Morten Lind

Open Real-Time Control and Emulation of Robots and Production Systems

Thesis for the degree of Philosophiae Doctor

Trondheim, February 2012

Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of Production and Quality Engineering

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Engineering Science and Technology
Department of Production and Quality Engineering

ISBN 978-82-471-3388-0 (printed ver.)
ISBN 978-82-471-3390-3 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2012:60

Printed by NTNU-trykk

To Trine, Sofus, and Oline

Acknowledgements

Thanks to The Research Council of Norway for funding my PhD scholarship through the IntelliFeed research project.

The IntelliFeed project was a research project with industrial cooperation, and I wish to extend my gratitude to the partners:

- Department of Production and Quality Engineering, Norwegian University of Science and Technology
- SINTEF Raufoss Manufacturing AS
- Scandinavian Business Seating AS
- Glen Dimplex Nordic AS

Thanks to my advisers at the Norwegian University of Science and Technology for support and interesting discussions:

- Professor Terje Kristoffer Lien, Department of Production and Quality Engineering
- Associate Professor Amund Skavhaug, Department of Engineering Cybernetics
- Adjunct Professor Trygve Thomessen, Department of Production and Quality Engineering

I am especially grateful for the support from my colleagues at SINTEF Raufoss Manufacturing AS, Department of Production Technology.

Abstract

This PhD thesis addresses the shop-floor control level in manufacturing and industrial production at some important points for the future. The main contributions are principles and software relating to the control of production systems and hereunder production devices. Specifically challenging at the production device level is the real-time, application-oriented and sensor-integrated motion control of industrial robots. At the higher level, the production control system for a shop-floor, considered as an ensemble of independently controllable production devices, is addressed. The challenge at this higher level regards the representation of all production devices of a shop-floor by suitable control interfaces for a distributed deployment of the control system.

The control of industrial robots and their integration into a production system is proposed to be troubled by each robot manufacturer developing their own native application platform. Each native platform offers some pre-chosen paradigm for programming and its own restricted set of technologies available for application development. For migrating the real-time motion control of industrial robots onto application platforms offering higher flexibility, or simply the right technologies, than that found in the native controllers, several software frameworks and platforms exist. The work underlying this PhD thesis has produced such a software framework, dubbed *PyMoCo*. It is entirely implemented in the high-level, interpreted programming language Python, and allows fast and highly flexible real-time motion control and application integration of industrial robots.

At the shop-floor control level, it is proposed by a segment of the research community dealing with production control, that holonic and multi-agent control architectures are suitable paradigms for bringing highly intelligent, computationally powerful resources into the direct control loops in production systems. The use of autonomous holonic or agent-based systems for distributed real-time shop-floor control presents some inherent difficulties regarding formal validation and verification. Hence, in the research community it has been suggested to use real-time, realistic emulators of the production systems as platforms for experimenting, developing, and validating holonic and agent-based production control systems. This PhD thesis presents principles for using the Blender Game Engine for implementing a physically realistic real-time emulator for the collection of production devices of a production shop-floor. An emulator has been designed and implemented for an extended version of a prototype shop-floor system set up in a laboratory, and an experiment control system has been implemented to validate the emulator. The presented emulator is developed to a state of being suitable for use as a realistic platform for development of a distributed, autonomous multi-agent system for control of the extended prototype shop-floor system.

Acronyms and Abbreviations

AGV	Automated guided vehicle, page 2
AI	Artificial Intelligence, page 1
API	Application programming interface, page 43
AS	Joint stock company (comparable to "Ltd.", "Inc.", and "Corp."), page iii
BGE	The Blender Game Engine, page 8
CNC	Computer numerical control, page 22
DES	Discrete event simulation, page 56
ERP	Enterprise resource planning, page 19
FireWire	The IEEE 1394 communication bus standard, page 5
GNU	GNU's Not UNIX, page 11
GPL	GNU General Public License, page 43
GPOS	General purpose operating system, page 37
HMS	Holonic Manufacturing System, page 6
IPC	Industrial PC, page 36
IPK	Department of Production and Quality Engineering, page 2
JIT	Just in time, page 20
LED	Light Emitting Diode, page 5
NC	Numerical control, page 22
NTNU	Norwegian University of Science and Technology, page 2
OTED	One-touch exchange of die, page 21
PAC	Programmable automation controller, page 37
PC	Personal computer, page 11

PLC	Programmable logical controller, page 17
PROSA	Product-Resource-Order-Staff Architecture, page 6
PyMoCo	Python-based robot motion control, page v
RTOS	Real-time operating systems, page 37
SBC	Single-board computer, page 7
SINTEF	The Foundation for Scientific and Industrial Research, page 2
Slerp	Spherical linear interpolation, page 216
SMED	Single-minute exchange of die, page 21
SRM-PT	Department of Production Technology at SINTEF Raufoss Manufacturing AS, page 2
TCO	Total cost of ownership, page 29
TPS	Toyota production system, page 21
UML	Unified Modeling Language, page 62
WIP	Work in process, page 4
$SE(3)$	Special Euclidean Group, page 216
$SO(3)$	Special Orthogonal Group, page 216

Contents

Chapter 1

Introduction

With respect to automation, much of the remaining industrial production in the developed countries may be characterized as somewhere between hard and flexible. A good share of the industrial production has been automated during the past three decades, but maybe even more have been moved to newly industrialized countries; notably China. Probably most of the remaining labour intensive production of goods in the developed countries are bound by local geographic requirement or by the need of a highly skilled labour force that would not yet be possible to find or educate in the newly industrialized countries.

What could have kept massive industrial production of ordinary goods in the developed countries from off-shoring to the developing countries was commonly thought to be the development of flexible, automated production systems. However, the off-shoring drove faster through the past two decades than the development of flexible and automated production systems. Furthermore, flexible production turned out to not meet today's volatile market requirement for manufactured goods. Agile, automated production systems seems to be a necessity to start up new, as well as keeping the remaining, industrial production of goods in the developed world.

Fast-paced development of agile, automated production systems should thus be emphasized to not lose the production and manufacturing competence and capacity remaining in the developed, high-cost countries.

1.1 Problem Domain

The main goals of the PhD work underlying this thesis is to advance further some possibilities for implementing advanced and computationally demanding control systems on high-performance computers at all levels of shop-floor control in production systems.

AI is a difficult concept to define. It is hard to define in a formal, let alone measurable,

way. AI is not explicitly used in the work presented here, nor is it claimed that the presented work implements any kind of AI. However, it may be argued that AI is a necessity for the realization of agile, automated production control. The agent-based paradigms for computing are generally agreed upon to be suitable for realizing AI, and they are extensively probed by the international research community to yield viable agile solutions in the field of automated production control.

This PhD thesis concentrates on three subjects, proposed to be of high importance in bringing AI into the control loop of agile, automated production:

- Making high-performance computational resources available in the real-time control loops of even the lower control levels in a production system.
- Development of flexible control frameworks to some of the more complex devices of production systems; notably robots and AGVs.
- Development of real-time emulation for the realistic simulation of production control systems, based on new paradigms of autonomous holonic and multi-agent systems.

1.2 Project Background and Outline

The work presented in this PhD thesis was part of the IntelliFeed project. IntelliFeed was a technology development and transfer project between research and development institutions and industrial production companies. The research and development partners were NTNU, IPK, and SRM-PT[1] while the industrial production companies which partnered were Nobø Electro AS[2] and HÅG AS[3]. All research and development activities were funded by The Research Council of Norway. The IntelliFeed project was active for the period from 2007 to 2010 inclusive. The PhD scholarship in the IntelliFeed project was given for three years in the period from 2008 to 2010 inclusive.

1.2.1 Project Background

Nobø Electro AS and HÅG AS are medium sized manufacturing enterprises producing mechanical goods; Nobø Electro AS produces electrical panel heaters and HÅG AS produces office and conference chairs. The two enterprises had many structural similarities in their enterprises and production systems, as well as the challenges with efficient automation of flexible assembly sections.

[1] SRM-PT was formerly the Production Engineering department of SINTEF Technology and Society.
[2] Nobø Electro AS has now changed name to Glen Dimplex Nordic AS
[3] HÅG AS is now a part of Scandinavian Business Seating AS

The IntelliFeed project was motivated by an earlier project, the SuperFlex project; with the same type of funding and the same partners, running for the period 2004 to 2007 inclusive. The SuperFlex project addressed the challenges of automating the assembly processes of mechanical goods. It was proposed, as a general tendency for small and medium sized manufacturing enterprises, that while most of the material processing involved with fabrication of the parts of final goods was close to fully automated, the assembly sections predominantly remained entirely based on manual labour. The challenge with automating the assembly section was accredited to the high flexibility required from it.

During the SuperFlex project a general consensus evolved that most of the assembly processes themselves were not especially challenging. Rather, the general transport for just-in-time deliveries of small, ordered numbers of component parts to specific assembly sites; the handling and feeding of complex component parts of the final goods to the assembly processes; and the real-time control system for operating the whole assembly system in detail were all identified as major challenges.

The main challenges addressed by the IntelliFeed project may be formulated as the following.

S/R Systems: If precise, small, measured numbers of ordered component parts must be delivered with low response time to the assembly sites ordering the parts, versatile and intelligent systems for storage and retrieval of parts is necessary as an integrated service in the production control system.

Stacking, Handling, and Feeding: Many components of final goods are of small and moderate size and shape. However, both partner enterprises have a good share of components that present severe challenges with respect to stacking, handling, and feeding; e.g. $2m$ by $20cm$ punched and pressed heater shells in sheet metal, wire harnesses for installing in heaters, and chair covers made from textile. Note that components involving textile or wires are not only a challenge for storing, handling, and feeding, but even more so for the assembly processes themselves; and no known, general solution exists to this assembly challenge.

RT Production Control: With smaller batch sizes and mixed production, towards single piece flow of parts, the planning and scheduling problem of production management transforms into one of real-time scheduling and control. To this end, it is not sufficient to deploy the ordinary implicit control of production systems, based on simple trigger signals. Rather, increasingly cognitive, distributed control systems based on information processing, advanced sensor systems, and communication are necessary for efficiently handling the increasing complexity.

The PhD work of this thesis addresses the challenge of real-time production control.

1.2.2 Project Evolution and Circumstances

This section brings an informal "eye-witness" account by the author, of how the PhD project was conceived and how it proceeded. It is intended to present a $10km$-altitude flyover of the central points of the development and evolution of the PhD project.

Preliminary Inspiration

Prior to any IntelliFeed activity, and hence prior to the presented PhD work, an activity led by professor Terje Lien as a masters project activity for engineering student Erik Haga, was centred around the simultaneous measurement and mechanical clamping of aluminium parts for welding. The mechanical development was undertaken by Erik Haga, but the development of the control for clamping, measuring, and correction submission to the welding robot system was undertaken by the author of this PhD thesis. The success of this project gave the first indication and confidence that an application developed in the interpreted language Python, confer [Python Website], from a standard personal computer system, in cooperation with dedicated peripheral devices and commonplace communication systems, could be reliably used for industrial application in production. A later publication on the subject is included as part of this PhD thesis.

SuperFlex, the IntelliFeed Predecessor

The predecessor project to IntelliFeed, SuperFlex, ended in 2007 and notably left Nobø Electro AS and HÅG AS with flexible transport solutions in the shape of palette conveyors, at the hearts of their assembly sections; almost all assembly processes were integrated up against the palette conveyors. Analysis of different low-response-time ($10s$ to $60s$) production transport solution for the flexible routing of single-piece WIP workpieces had indicated that the palette conveyors made the best compromise between flexibility and reliability. AGVs had been briefly considered for the transport and carriers of single-piece workpieces in-between and under assembly processes due to their potentially extreme agility, but under suspicion of being unreliable AGVs as solution principle was evicted.

The palette conveyors set up at Nobø Electro AS and HÅG AS were not, or only to a limited degree, extended to and integrated with intermediate component parts storage, so the transport of part batches to and onto the palette conveyors, as well as supply of parts to the assembly stations, were largely left as manual labour operations. These were, however, in grand total not as time and effort consuming as the transportation between assembly sites and associated preparation operations.

Free-Roaming, Prototype AGVs

Automation of the logistics operations surrounding the assembly conveyor systems was a leftover from the SuperFlex project and an overall motivation for the IntelliFeed project. The author of this PhD thesis suggested and undertook early on a short and limited project, which made an attempt at in-house, low-budget development of a prototype AGV. The goal of the prototype was to assess whether a very cheap, mechanically reliable and controllable free-roaming AGV could be built by consumer-grade, off-the-shelves electromechanical components.

The design target was for such an AGV to be able to drive at a march speed of at least that of human walking pace ($\sim 1\frac{m}{s}$) and to carry a load of about $100kg$ on a flat top plate. The ubiquitous issue with free-roaming AGVs is the localization problem, and to keep costs low while allowing for flexibility and redundancy, an accompanying localization system was based on AGV-mounted LEDs and cheap, ceiling-mounted FireWire cameras. The cameras were equipped with optical band-pass filters centred at the peak of the LEDs' emission spectrum.

The AGV prototype development was completed with success of the vague, qualitative goals, thanks to a very skilled masters student from Politecnico di Milano, Stefano Pedemonte. He worked intensely together with the author of this PhD thesis on the project for half a year.

The activity of AGV prototype and localization system development was not taken to industrialization, or even subjected to quantitative performance tests. However, the AGV was later used by a Holonic AGV System, mainly developed by Olivier Roulet-Dubonnet.

Onset of IntelliFeed Research

From its beginning, three PhD scholarships were announced in the IntelliFeed project, of which two were given to researchers already associated with the IntelliFeed project. The AGV development activity was to be carried over into a PhD project undertaken by Olivier Roulet-Dubonnet. Another PhD project, the one presented in this thesis, was undertaken by the author, and the project was addressing the development of real-time production control systems.

The first phase of research in the IntelliFeed project concerned itself with the identification of a suitable demonstration case, which could serve as a direct prototype, and later pilot, for one or both of the partnering industrial production enterprises. The case settled on was one of intense manual labour in both production enterprises: The up- and download of workpieces from carriers of the painting systems in the productions. A demonstration prototype based on a Power-and-Free overhead conveyor for fixing painting system carriers under up- and download of workpieces was scheduled for setting up with two industrial robots in the laboratory of IPK and SRM-PT. An overview of an extended version of

the IntelliFeed demonstrator, with the purpose of development of a shop-floor control system, may be observed in Fig. 3.1 (page 61).

Meanwhile, Olivier Roulet-Dubonnet together with the author started an analysis for selecting middleware technology for implementing an Ethernet-distributed control system, permeating from the highest level of management to the lowest level of logical, real-time device control. The selected technology was the quite new, free and open-source Ice™ middleware from ZeroC Inc. [ZeroC Inc. Website]. Together, Olivier Roulet-Dubonnet and the author analysed the general application needs of a shop-floor production system, such as the IntelliFeed demonstration case, and designed a holonic or agent-based facility framework for implementing and managing a distributed control system. This framework, which was successfully tested and taken to use in several other projects, was implemented by Olivier Roulet-Dubonnet. It was dubbed IceHMS, and much inspired by the PROSA reference architecture for HMS.

The first direct case of use of the IceHMS framework was for developing a holonic AGV system, working for the real-time management of the previously developed prototype AGVs, and integrating the developed localization system. This activity was also, in its initial phase, a highly collaborative effort between Olivier Roulet-Dubonnet and the author.

Real-Time Robot Motion Control

The robot control tasks in the IntelliFeed demonstrator case was not apparently complicated, and an ordinary task and application implementation on any standard industrial robot controller should suffice. However, it was clearly recognized from the demonstrator analysis phase that it was not so simple altogether. The painting system carriers, though geometrically well-specified, were made of soft steel and often bent several cms out of shape from dangling and entangling during transport in the painting system.

The most complex part of the robot system development thus turned out to be the 3D localization of carrier attachment points and attachment directions. In a separate project, owing much to Pål Ystgaard at SRM-PT, a stereo vision system from Tordivel AS [Tordivel Website] was set up for carrier identification and attachment point localization. Even with this advanced 3D vision system, and for just two specific carrier and workpiece types tested, it was a challenge to meet the tolerance of the attachment motion of workpieces.

This sparked the contingency planning of using sensor-based real-time motion control for the attachment of workpieces onto carrier sites. Standard industrial robot controllers, except the newest and the emerging ones, have very little or limited support for the real-time, intermediate frequency ($\sim 100Hz$) motion control required by sensor-servo control. Thus came about the desire for bringing the industrial robots under real-time motion control on a computer platform external to the native robot controller; a computer

platform with the necessary computational resources and technologies for creating and controlling the robot according to a model-based sensor integration.

This desire for achieving externally controlled real-time robot motion was as much sparked by the need as the opportunity: In an activity entirely unrelated to IntelliFeed, researchers at SRM-PT, Johannes Schrimpf and Thomas Ulleberg, were working on embedding an SBC into the Nachi AX10 controller, which provided interception of the communication between the high- and low-level controllers. This interception was controllable over the Ethernet port of the SBC, and thus allowed an external computer to completely take over the joint-position-level control of the low-level controller; running at $100Hz$ with approximately $150ms$ trajectory control delay.

In collaboration with Johannes Schrimpf, the author started the development of PyMoCo, a framework for real-time motion control of industrial robots in tool and joint space, over native controllers providing real-time joint-position-control. As part if the PyMoCo framework a set of "canonical" motion controllers were developed, comprising those provided by the application platform of typical industrial robot controllers, but also featuring real-time interactive motion controllers such as a joint velocity controller, a tool velocity controller, and correction tool and path controllers.

The PyMoCo framework never came to direct use in the control system for the IntelliFeed demonstrator, since the traditional approach with 3D vision eventually managed to meet the tolerances. However, in the presented PhD work, PyMoCo became a central and necessary ingredient in the activity for developing a real-time emulation setup for the IntelliFeed demonstrator, as well as for developing the experimental control system tested on the emulator. Further, PyMoCo has found its use in several other projects in need of sensor-based robot motion control.

Waiting for Equipment

At a specific point in time the following important status was reached within the component sub-systems:

- The PyMoCo framework for real-time motion control for the Nachi SC15F robots, controlled by Nachi AX10 controllers, was in place and working well with the robots. A simultaneous development by Pål Ystgaard had a more traditional robot application based on the native high-level controller platform meeting the tolerance for uploading workpieces. I.e. regarding the robot applications for the workpiece upload, everything was doubly ready.
- The IceHMS middleware framework for Ethernet-distributed, agent-based control was implemented and tested by Olivier Roulet-Dubonnet.
- the 3D vision system had been experimented and set up to perform at its best regarding time and accuracy.

- A micro-controller board had been setup and programmed by Johannes Schrimpf to integrate the control over Ethernet of a PnF-conveyor that had been specified and ordered for installation in the laboratory. This included the commanding of any of the PnF-valves for controlling the stopping states of the PnF-stops, reading the proximity sensors at any PnF-stop, and controlling the drive-train speed of the conveyor.

In other words, all that was needed for the overall laboratory production system as a demonstrator in the IntelliFeed project was the overhead PnF-conveyor itself. All other component sub-systems had been tried in individual testbeds. The waiting was not just impacting the progress of the IntelliFeed project itself, but it was a major delay for the PhD work presented in this thesis, as it was addressing exactly the integration and control of all component devices and sub-systems.

A natural remedy for this situation, ensuring the progress, of the central activity of the PhD project, was to break open the development of an realistic real-time emulation platform for the demonstrator system.

Real-Time Emulation Platform

The conception of using the Blender [Blender Website] Game Engine (BGE) for realistic real-time emulation of a production shop-floor came while waiting for arrival and installation of equipment for the physical IntelliFeed demonstrator system. The enabling experiences had been in place already from the early phase of development of real-time robot motion control, in which a simple but realistic real-time emulator of the low-level controller and its motion of the mechanical robot had been implemented. The robot control emulator development was motivated by three aspects which were equally valid for the emulator for the IntelliFeed demonstrator:

Safety: Experimental control of large, heavy, fast, or strong mechanical systems, e.g. industrial robots, AGVs, conveyors, always involve safety risks. These safety issues are entirely cut out by using pure software emulation.

Availability: A production system at shop-floor level comprises many individual devices that are interdependent in long operation chains and intertwined implicit states. In an physical system, such as a laboratory prototype, at any given instant in time there is a high probability that one or more necessary devices are

1. not arrived or setup yet,
2. experiencing downtime for repair or maintenance, or
3. borrowed or leased by other laboratory projects.

This impacts, often severely, the progress of the project.

Efficiency: The typical experiment cycle in a laboratory comprises the sequence of

1. stopping operations,
2. shutting down all devices necessary,
3. making changes or adjustments,
4. replacing or re-localizing workpieces,
5. starting up all devices,
6. starting operations,
7. recording and monitoring operations.

It is evident that for a pure software emulation system, this experiment cycle time may be vastly lower than for a physical laboratory system.

It is important to note in this connection that, however high the advantages with developing a control system over even a very realistic real-time emulated system may seem, validation of the control system over the emulated system does not imply validation of the control system in the physical system. While this seems to impair the principle of development over an emulated system, it is equally important to note that failing to validate a control system in a realistically emulated system almost always implies that the control system would fail on the physical system; at least for sufficiently realistic emulation. Hence it is suggested that control system development takes place with an emulated system until successful, and then proceed with the physical system.

Whether the benefit of the emulated system will exceed the cost of implementing it is a matter of analysis for each pertinent target system. When attempting to estimate the balance in such a cost-benefit analysis, it should be taken into account that in addition to the benefit of having an emulator for a target production system, the development of the emulator itself brings about valuable knowledge and detailed analysis also usable for the targeted physical system.

The above analysis assumes that a target physical system has already been chosen. Such was the case for the circumstances of the laboratory prototype system for the IntelliFeed demonstrator. But if an emulation system is being set up in advance of the selection and layout of the physical system, it may be a valuable integral part of the system design process. Such design-phase integration could have helped avoid a slightly unfortunate placement of the upload robots with respect to the upload stop on the PnF-conveyor; a placement resulting in some carrier sites being out of reach of the robot used for the demonstration scenario.

Experimental, Simulated Demonstrator Control System

The development of an emulator for an extended IntelliFeed demonstrator took up a considerable amount of time of the PhD project, and came to be a central and fruitful

part of it. However, it took much of the resources away from the planned focus, i.e. research and development of advanced production control systems.

Eventually a quite advanced, Ethernet-distributed control system for the IntelliFeed demonstrator emulator was developed. Initially this was an activity which was a natural and integral part of developing the emulator itself, since control of the emulated system was necessary for experimenting and testing. Later on, as the emulator was in its final phase and close to completed, focus switched from the emulator development and concentrated on transforming the pieces of control system code for experimenting and testing the emulator into a full-fledged control system in its own right.

The completed version of the control system is adequate for controlling the extended IntelliFeed demonstrator. Three circumstances prevented it from ever being deployed on the physical demonstrator system in the laboratory:

1. It was, as previously indicated, developed for a quite extended version to the IntelliFeed demonstrator, involving more devices, such as AGVs, more robots, more vision systems, turntables, order based workpiece providers, and exposed transport and re-supply issues.

2. A simpler control system, also based on agents, had already been implemented and used for demonstration in the IntelliFeed project on the actual, physical demonstrator set up in the laboratory. The control system was developed, tested, and demonstrated by Per Åge Nyen at SRM-PT.

3. At the time of completion of the control system developed in the presented PhD work, the IntelliFeed demonstrator system was no longer available due to other laboratory activities.

1.3 PhD Thesis Outline

The PhD thesis presented here has four major content components.

- Chapter 2 presents an essay of philosophical considerations assembled by the author during the period of the PhD work. First it introduces some terminological use in manufacturing and industrial production research, mainly to clarify some, to the authors opinion, frequent causes of slight confusion. It then presents a succinct and subjective selection of the history of manufacturing and production. The remainder of Chapter 2 is dedicated to a series of speculations over the paradigms for production control under development in some part of the research community, and relating it loosely to paradigms of previous eras and in contemporary use in manufacturing and industrial production enterprises. Chapter 2 presents many views that are associated with the philosophical background, and the very motivation, of the work presented in this PhD thesis.

- Chapter 3 gives a somewhat detailed account of most of the technical development undertaken by the author throughout the PhD scholarship period. The account is to a wide extent based on references to the publications included with the thesis. The majority of the effort is invested in describing aspects and development efforts which have not been adequately represented in the included publications. Notable subjects that have received much attention in Chapter 3 regards the use of GNU/Linux based PCs in real-time production and device control, and the development of an experimental shop-floor control system for the extended IntelliFeed demonstrator. The account of the developed control system goes into some details of the software design at the mechanistic level, and refers to code for the control system and interfaces of the emulator included in Appendix A.
- Chapter 4 is a selection of published and submitted papers prepared during the course of the PhD scholarship. Most of the work in the PhD project have been disseminated at scientific conferences and in scientific journals. The author has a considerable share in the underlying work of all the included publications. They have been selected among other publications, in which the author also contributed, for their coverage of the central parts of this PhD project. The collection of published papers is to be considered the main contribution of the PhD thesis.
- Chapter 5 concludes this PhD thesis proper by stating the main conclusions, giving a summary of contributions from the PhD work, and presenting a short account of planned and progressing future directions.

Finally a bibliography of cited publications, a webliography of referenced websites, and an appendix containing central interfaces descriptions and implementation code are given.

Chapter 2

Philosophical Considerations on Production Control

This chapter gives a presentation of views on philosophical themes and aspects of production automation and control. It shall not be considered as a thorough, corroborated, or comprehensive survey of production automation philosophies. The views and descriptions are representative for the philosophical considerations underlying the work of this PhD thesis.

Central to the presentation in this chapter are a series of propositions. The propositions are succinct formulations of some core concepts within the field of production and manufacturing engineering science. It must be kept in mind that about each and every one of the propositions, several to hundreds of books have been written and published. By no means the stated propositions are meant as a diminution of the extensive works of others, but an attempt at distilling their essence to as few as one to two sentences.

Thus, while it is the hope that the descriptions and the themes are recognizable, it is further hoped that some amount of fresh view on relations and new perspectives of the matters are also found.

2.1 Terminology

In research literature and correspondences, as well as in common everyday-use, there seems to be a lack of consensus about what concepts are covered by the terms *manufacturing* and *production*. While the thesis at hand is not a theoretical study on the scientific area of production- or manufacturing-engineering, a non-negligible amount of confusion is assumed to arise without the precision and motivation of the concepts behind those highly central terms.

First and foremost, it is natural to examine the etymology of the terms. A superficial investigation in standard dictionaries shows that both are composed words of Latin origin; manufacture meaning literally "to make by hands" and produce literally meaning "to lead forward". In modern concepts this transfers to an impression of manufacturing being process and machine oriented, and is clearly associated with something tangible and physical. Production literally gives the impression of being more concerned with management or conditions slightly peripheral to the real activity involved, and it does not necessarily imply something physical or tangible. The term production is further used in many different contexts, such as arts and theatre, computing, agriculture, etc. To limit the context of production to that which is relevant in this thesis, the term should everywhere be perceived in the context of manufacturing.

A short excursion into standard dictionaries of German, British, and US-American reveals that the concepts behind terms such as "Produktion", "production", "manufacturing", "Herstellung", "Fertigung", "fabrication" seem hard to dis-entangle, let alone finding an unambiguous and clear concept for each of them in their own right. They are highly context dependent in use; "production" more so than "manufacturing".

At online fora the exact question of the difference between manufacturing and production is abundantly posed. From observation of the answers given by various persons, the professions of whom are stated to be within the fields of manufacturing and production, are quite diverging and often contradicting. A conclusion from this is that the concepts behind may not simply be dependent on lingual culture, but even professional culture; e.g. with exactly which company a person is associated or employed, or from what department at which university a person graduated.

This PhD thesis adopts a terminology in which the concepts are used in consistence with their etymology, while not introducing or inventing new aspects. The following propositions defines the concepts as they are used throughout this thesis:

Proposition 1 (Manufacturing)
Manufacturing is the transformation activities and processes of physically creating goods from raw material or input parts, utilizing tools, machinery, and labour.

Proposition 2 (Production)
Production is the aggregation of activities, resources, and personnel utilized for, and directly involved with, the systematic manufacturing of any kind of goods with the intent of sale or later processing into final products. Production may be described as organized and formalized manufacturing, taking place at a factory, as part of a production enterprise in a production company.

These proposed definitions of central terms serve to clearly distinguish the associated concepts. By these definitions, it is possible to distinguish what is technically a producer from a manufacturer. For instance, most modern industrial robots are produced. Hence the companies producing and supplying the robots may be appropriately termed "robot producers". The same companies will typically develop new robot models as well, and

when making robot prototypes they will be manufacturing these but not producing them. If, however, they develop custom robots, and supply these to customers, they may, in that capacity, be appropriately termed "robot manufacturers" too. Machine builders are often performing manufacturing of the machines for very specific customer purposes. In contrast, simple goods like pencil sharpeners would typically be manufactured under highly optimized, formalized circumstances, and hence they are produced.

However, since "production" is a broad term, used in many areas of activity, the term "producer" is not very specific. A farmer, for instance, is a producer, but he has no manufacturing involved in his enterprise activities; natural processes take the place of manufacturing processes in this case. Hence, when referring to a producer which has manufacturing as its core activity, "manufacturer" is mostly used. This thesis adopts the common use of "manufacturing", and does not strongly or consistently distinguish between the use of the two terms "production" and "manufacturing".

It may be summarized that manufacturing is at the essence of physical creation of goods. Production, in contrast, is to do with control of the immediate circumstances of manufacturing, which enables it to be carried out efficiently and in a manner that allows replication of the end products. Mostly, the objective of production is mass manufacturing of goods of a controlled quality.

The definition of production make use of the term *production enterprise*. A definition is proposed as follows:

> **Proposition 3 (Production Enterprise)**
> *A production enterprise is an integration of the whole range of entities relating to, and including, a production department and a production system. This comprises business entities with functionality such as product development, production development, maintenance, planning, accounting, purchasing, sales, marketing, distribution, etc.*

In short, a production enterprise is the production domain part of a production company; i.e. the company may be thought of as the legal and financial entity, in which the production enterprise is embedded.

The essence of the concept of a production enterprise can be carried over to the concept of a *manufacturing enterprise*, which implies the absence of production-concerns around the manufacturing. As an example, consider the work in a joiner's workshop where wooden chairs are made with hand-tools, where details are added ad hoc as to the pleasure or idea of the joiner or customer at any given moment, with no formal specification of processes, and where no repetitiveness of processes have guaranteed a deterministic quality of any given finished chair. This would be appropriately called a manufacturing enterprise when considered together with the joiner's accounting, sales, purchasing, and general interaction with the customers. There is no particular production aspects involved. Such artisan-based manufacturing enterprises would typically begin their transformation towards a production enterprise with the purchase of milling

or turning machinery suitable for batch processing of chair seats, arm rests, legs, support structure elements, etc. This is where the more detailed planning and scheduling begins, as well as formal geometric specification of any element of the chair types that will have most of their elements made from machinery.

The work behind this thesis addresses problems mostly relevant to what may be called *complex production*. The results of the work may be applied to the complementary concept of *simple production*, but is thought to be less likely to be of need or to give profitable benefit. Complex production is not easily defined in a quantitative manner. The following is a proposition of a definition of complex production:

Proposition 4 (Complex Production)
A production which must schedule tight deadlines to fulfil the production plans, pressing the flexibility towards the limit of the theoretically possible in terms of changeovers; where the internal logistics of the production is complex; or where external control of manufacturing processes are complex, is said to be complex.

Some remarks regarding production enterprise, production, and manufacturing are in place in connection with this proposed definition of complex production. Production complexity is different from production enterprise complexity, which is a characterization of how sales, ordering, stock, purchasing, etc., interact and operate. Production complexity may be determined by the arrangement of production devices and machines; the real-time status of all ongoing processes and devices; the real-time coordinated, collaborative, or cooperative control of the aggregation of production devices; and the detailed progress and dynamical scheduling of the orders and products in progress. It is fully possible to have complex production in a simple production enterprise, or vice versa. This is analogous to the possibility of finding simple manufacturing in a complex production, or vice versa. Manufacturing may be called complex if the manufacturing processes are complex; typically implying high internal complexity of the manufacturing machinery or tools. Internally complex manufacturing processes or machinery does not necessarily imply that the external control of the machinery or processes need to be complex. A CNC milling machine, say, may have immense internal complexity by itself, and even the programs for a range of workpieces may be highly complex. However, once the machine complexity is handled well by the machine manufacturer, and the process control for the milling jobs on the workpieces have been handled by an experienced programmer, the production aspect of using the machine may be very simple. The example illustrates a case of separation between the manufacturing and the production processes.

Of central importance to automated production are the concepts of *production machine* and *production device*. Their use deserve separate propositions:

Proposition 5 (Production Device)
A production device is a controllable technical installation or equipment serving a purpose in the production system; and under real-time production control. It implies an obligation from the production control system to manage the device in a responsive, timely manner depending on the pertinent requirements to the device.

Proposition 6 (Production Machine)
A production machine is a production device that is internally complex. It may provide several control aspects and interfaces to the production control system, some of which addressing complex setup, monitoring, and information retrieval.

When used in the context of production, with no obvious ambiguity, the word "production" is dropped from these terms. The production devices and production machines may cover the production aspect of manufacturing machines and manufacturing devices. Specifically with respect to automated production, they are concerned with the external, exposed interface to such machines or devices with the purpose of automatically controlling them within the production system. Production machines and production devices may also be unrelated to manufacturing, and in such case they typically relate to processes involved in production logistics, i.e. handling and transport of parts and goods, or packaging of goods.

It should be noted that the proposition of a production device excludes equipment that may not be controlled, or installations which are purely mechanical in nature. Such installations may be adequately termed *passive production devices* or *reactive production devices*. A central theme of the PhD work for this thesis is fully automated production, and all devices are assumed to be controllable or interactive by digital communication. Most reactive devices with actuators may, when it serves a purpose, be brought to be controllable by associating a dedicated computing unit, such as a PLC, a micro controller board, an embedded PC, or just plainly a PC with adequate data communication.

Plain devices, i.e. devices that are not as complex as machines, are mostly simple, standardized equipment. Examples that frequently occur in automated production systems are conveyors, simple robot tools, valve controlled pneumatic cylinders, servo tracks and gantries, vibration feeders, etc. These examples represent simple devices with no hard, high-frequency real-time requirements by themselves, giving limited or no status data and with simple control signals. For a manufacturing company with in-house development of their production system, plain devices may be tightly linked to make up compound machines. Such linking may be of mechanical, electrical, or control-communication nature, and the component devices will typically be inaccessible to all but the associated devices in the compound.

Machines are often custom made by an automation supplier or machine builder for a very specific function; often in the form of customized, electromechanical assemblies of standard off-the-shelf components. Pallet transport systems may be considered examples of modular machines that are customized from standard elements. Industrial robots are

examples of monolithic machines.

The insinuated distinction between plain devices and machines is not a clean categorization into disjoint sets. It bears more resemblance to the two extremities of a spectrum of complexity and use. Clearly, an industrial articulate robot is a machine, while a pneumatic valve with a logically controlled solenoid switch is a plain device. However, an installed robot with a tested and proven programmed logic may support two simple signals for "pick" and "place" with no further complexities or parameters, whereby the machine is turned into a device from the perspective of production control. In a simple production this is the most common way of using robots, but it is notable that the online and real-time flexibility of the robot as a production machine is thus also removed.

Machines may be divided into two rough categories of logistical and transformational natures. Machines of a logistical nature are devices for handling, picking, packaging, and transporting products in progress. Machines of a transformational nature are the actual value adding devices; those that perform manufacturing processes. They are frequently highly custom made, with very complex internal control. CNC machining stations, fish fillet cutters, welding automatons, and thread rolling machines in screw production are examples illustrating the great variation in the plethora of transformational machines. When a drill or a welding torch is mounted on a robot, the combined system of tool, robot, motion control, and transformation process control may be treated like a compound transformational machine. When the same robot is equipped with a picking-tool for moving workpieces from a press to a box, it is to be considered a compound logistical machine.

The presented distinction and propositions for defining production devices and machines is partially in agreement with the concepts "intelligent device" and "intelligent machine" as used by Auinger et al. [2005]; though they incorporate software components explicitly with the concepts. Their concept of machine is, however, strictly reserved for machines supplied by machine builders or machine vendors. In the proposed definitions above used in this thesis work, the example of a robot with a mounted drill or weld torch represents a machine built by the systems integrator at the shop-floor, rather than by a machine shipped by a machine builder.

The concepts of devices and machines leads this thesis to the following propositions for the concepts of *production system* and *production control system*:

Proposition 7 (Production System)
A collection of machines and devices with their controllers, possibly organized into workcell areas or production lines, the associated labour and operators, and the supporting facility installations taken together for the controlled manufacturing of a certain spectrum of goods, is called the production system.

Proposition 8 (Production Control System)
A production control system is a system of one or more computers that interact over a digital communication medium with each others and with all machines, devices, and labour in the production system. Its main objectives are to handle, control, and orchestrate all aspects of production not covered by internal control of the production devices.

A production system, as considered in this thesis, is not active in itself, but has exclusiveness in the potential for performing any physical actions. It takes a production control system to give life to a production system, because it adds to the whole system the decisions; the actions of carrying out the decisions; the commands to machines and devices for configuration and operation according to the actions; and it determines and sets up the low level interactions among machine and device controllers.

Production control in a traditional sense may be understood as the execution of routines, formal or informal, written or in the minds of personnel, and the actions of the persons involved with the matter. A traditional production control system may be defined as the production control information together with the computer systems and the personnel involved with operating the production system in real-time. It is clear from the above proposed definition that this PhD thesis addresses fully automated production control, which distinguishes itself from traditional production control by the absence of informal descriptions of routines or actions from personnel.

The production aspects that must be addressed by any kind of production control system may be illustrated by the examples in the following list:

- Detailed, real-time, and reactive scheduling.
- Coordination of manufacturing processes.
- Production logistics.
- Supply from raw material stock.
- Changeovers
- Progress monitoring
- Reporting

In a typical production company with automation of both production and production enterprise, the main input to the production control system stems from the ordering and planning aspects of the production enterprise. The input may be expressed to the production department, or directly to the production control system, through an ERP system. The form of the input is in day plans and rough sequencing. The sequencing is formulated in terms of delivery deadlines of specific product types and quantities for shipments during the day.

2.2 A Brief History of Production

The works, methods, and principles developed and applied by Taylor [1913] and Ford in the early 20th century are renowned for the foundation of the engineering area that brought systematics to manufacturing, and from which emerged what is today referred to as industrial production. The main points may be crudely summarized by:

- Simplification of manual transformation processes by deploying specialized machines.
- Reduction of product complexity while maintaining or increasing product quality and functionality.
- Structuring the management and information of production operations and transformation processes.

Taylor's scientific approach to management of a manufacturing enterprise aimed at raising the efficiency by structuring and systematize manufacturing operations, workshops, machinery, and human labour activity. It became key to mass manufacturing of merchandise in manufacturing companies; thus evolving to production companies. Production of identical cars based on standardized components enabled Ford to lower the costs of the individual products sufficiently that the working classes could afford them. The affordability of the products, and the market they thus opened, justified and made sense of systematic mass manufacturing. Ford's insight into, and courage to exploit, the symbiotic relationship between mass consumption and mass manufacturing, may have contributed a considerable share of the spreading of wealth from production in the industrialized world, throughout the 20th century.

Ford's principle of reduction of product complexity aimed at lowering the material costs of the product, and simultaneously simplifying the manufacturing processes for the production and assembly. Structuring the end-products from standardized components made product upgrades easier regarding the production system, because their impacts were confined to proportionally few affected production lines, workers, and machines. Another achievements which is, contrary to common beliefs, accredited to Ford is the JIT principle, as argued by Petersen [2002]. The JIT principle was applied to the ordering of material, such that no material was stockpiled far in advance of its need in production. The principle was used not only on an external supply-level, but also production lines, and even each process-site on the production line, were supplied only the amount of parts for a foreseeable future of planned operations. The replenishment would even, with advantage, be delivered by the external supplier directly to the transformation processing site on the line, and thus even cut out a central storage. The main benefit that comes with the principle of JIT is the release of investments in stockpiled supplies and the associated valuable shop-floor spaces it takes up. The benefit of applying the JIT principle will outweigh the more vulnerable complex and tight logistics only if the latter can be handled in a sufficiently reliable manner.

With centralized production, Ford faced a serious spatial problem of transportation capacity for the sheer amount of cars; cars being not exactly easy to package tightly in large amounts. The solution to the problem was to geographically distribute the final assembly of the components, making the shipments from the factory much more compact. The final assembly would then take place near wholesaler or retailer. Thus, the final assembly was done to individual order (*assembly-to-order*), and the specific variant of the final car would be determined by selecting among compatible components, specified with the individual order (*delayed product differentiation*).

The Ford-T model is an eminent example of the achievement of production in the form of mass manufacturing. However, the post war increase of consumers wealth became the driver for diversity in the requirements, demands, and wishes to the products. The multitude of final products, the product spectrum that a manufacturer had to provide to remain competitive, became a direct challenge for the streamlined, optimized, highly tuned production lines. Methods and principles for accurate sales projections was not well developed at that time, and in the name of supply security for the wholesalers, final goods began to pile up at the manufacturer. Though expensive on the investment in final goods stock, it was possible for US-American automotive manufacturers to implement such a space-consuming principle, but in Japan problems arose. Being an automotive manufacturer in a densely populated country, Toyota had to seek a way of simultaneously meeting the requirements of a broad product spectrum, and not fill up the precious stock space with cars that were not certainly known to be sold and shipped in a short time. For Toyota, producing only what was ordered, and only close to delivery time, became the solution. This was another application of the JIT principle; in this case supplying the final goods storage with only what was ordered.

The problems faced by Toyota resulted in the development of TPS, notably by Taiichi Ohno. It is widely recognized as the forerunner of the lean production philosophy, which has permeated to many production companies today. Much of TPS is aimed at the business management around production. However, some elements of TPS are aimed directly at the production control and strategy. SMED and OTED were key technical achievements for enabling the shorter lot size, which was important for TPS; such technical key achievement for the success of TPS are well described by Shingō [1989]. The names derive from the main problem of changing die in stamping presses for the car bodies. Such die-change operations posed the major challenge to reducing the change-over time, which was urgently necessary to lower the lot size while remaining efficient. Only by lowering the lot size could Toyota produce the cars close to the delivery times.

Where SMED and OTED were directly aimed as principles of inter-transformation process operations, the production control strategy they were part of enabling is referred to as *Pull Strategy*. The pull strategy broke with the predominant *Push Strategy* of production control. In a production controlled by a push strategy, the production planners first have to be certain of what to produce in the end. Then a corresponding amount of raw material, corresponding to the total bill of material, are pushed into the produc-

tion system, every segment of the production pushing its products in progress further downstream in the production. When the situation of a production company is such that prognostics is reliable and planning is optimized, the push strategy may be more efficient and productive than the pull strategy. However, especially if the prognostics is highly uncertain in estimating the later demand-situation, the push strategy may result in large, long-term stocks of end-products. The technical development at Toyota enabled a pull strategy to be developed, and the solution was based on cards that were passed upstream in the production to notify the need of products at specific downstream points. A card, or "Kanban" in Japanese, thus triggered upstream activity directly from a downstream need. In a well-implemented "Kanban"-system, the production of final goods is simply ordered by placing the desired amount of cards at the shipping area of the production.

The direct technical achievements in the production at Toyota, as well as the philosophical achievements in production management, may be considered unsurpassed. With introduction of NC and CNC in machinery, for automated precision transformation processing, and later robots and robotics for handling and assembly, the levels of efficiency, automation, and transformation process capabilities increased. In TPS, the concept of *autonomation* was established for describing and organizing the cooperation of a human operator and an automated machine.

With the onset of fully automated production systems, implying the elimination of machine operators from the shop-floor, system control computers was introduced for taking over the supervisory and communicative level of the operators. The application and control software deployed onto the various machines and devices was becoming increasingly complex and interdependent between machine and system controllers. The total reliability became an issue as a result of the control paradigm and the sheer number of potential failure points.

To deal with such new complexity in software and application control, a number of ideas and paradigms have been proposed, debated, investigated, and experimented with. The earliest, most successful, and the only one to come into use, is the hierarchical control paradigm. Its success in automated production control hinges on a number of qualities, such as amenability to analytic verification, computational efficiency, and clear and static responsibility delegation. If a hierarchical control system is also centralized, i.e. fully synchronized in execution, the highly desirable quality of determinism may be attained.

In spite of an increasing level of criticism against the hierarchical and centralized production control paradigm, due to changed circumstances and requirements to production, it is safe to say that it is the de facto pinnacle of achievement within the realm of fully automated production control.

2.3 A Battle of Paradigms for the Future

The competition, and controversy, among paradigms for the fully automated production control and management systems of the future can be inferred from the following two propositions:

> **Proposition 9 (Traditional Evolution)**
> *The traditional production engineers insist on remaining with a proven, well analysed, hierarchical or centralized control system. They will attempt to extend and re-factor it to accommodate current and future requests for flexibility. Every small step of the way must present a stable and reliable solution, adequate for continued deployment in production.*

> **Proposition 10 (AI Revolution)**
> *When considering the most extreme requests for manufacturing and production flexibility, the AI-minded engineers propose that only by distributed deployment of AI and allowing autonomy into all levels of manufacturing and production control may the requested flexibility be fulfilled in a scalable manner. By allowing autonomy in distributed, intelligent entities, emergent behaviours may have to be tamed and tethered to prohibit inadequate, spurious control decisions.*

Both of these paradigms are generally applicable throughout the spectrum of manufacturing and production. Being paradigms, they do not lay down hard constrains on the architectures, designs, or implementations of production control system. Rather, they may characterize the architecture, design, and implementation of a given production control system, or the state of minds of the developers during its development.

The paradigms are conceived to be efficient in different regions of the spectrum of production characteristics, and should be expected to have very different profiles of efficiency with respect to production performance, development effort, and maintainability.

Hatvany [1985] details how analogies to the controversy of these paradigms of hierarchy and centralization versus distributed autonomy apply in society, in corporate structures, and in scientific tradition. It demonstrates that this controversy is neither new, nor limited to production management and control. Raymond [1998] gives a good overview of how the controversy of paradigms apply to the development of large software systems.

2.4 Hierarchic, Heterarchic, and Holarchic Control

Where a hierarchy may be described as a simple, rooted, acyclic graph, a heterarchy is a dynamic graph with no restrictions on connections; i.e. a general graph. Trying to tame the inefficient anarchy of such a heterarchic structure with its unlimited freedom while retaining some of its flexibility, the concept of holonic control was introduced.

This section will state and discuss some loose and succinct propositions of three different paradigms of control systems in the following. Van Brussel et al. [1998], Hatvany [1985], Leitão [2009], and Blanc et al. [2008] all gives comparative descriptions of the paradigms from the perspective of heterarchic or holonic systems for production and manufacturing control, and with emphasis on the shortcomings of hierarchical or centralized approaches. Dilts et al. [1991] dedicates a more detailed treatment on the evolution of hierarchical and centralized control systems towards heterarchic control systems in production and manufacturing. In many ways, the holonic architecture resembles the "modified hierarchic" architecture of Dilts et al. [1991], but with the possible difference in dynamics of superior-subordinate and inter-subordinate relations.

The traditional, hierarchic control system may be described as:

Proposition 11 (Hierarchic Control System)
A hierarchic control system is a statically, hierarchically organized system of control entities where each control entity is subordinate to exactly one superior control entity, and exclusive superior to a number of subordinate control entities. Interactions are strict with respect to commands going from a superior to a subordinate, and status information in the opposite direction.

It is apparent from this proposition that designing and implementing a hierarchic control system is a task that may be immediately undertaken and started, provided that an analysis and ideas exist about how the overall logic of operation should proceed. This is a situation often met in a production subsystem of limited scale or complexity. The notion of "statically" mentioned in the proposition, is not a strict requirement, but allowing dynamics in the hierarchy opens a whole range of complicated failure modes, whereby one of the major benefits of the paradigm vanishes.

When considering larger or more complex subsystems, the immediate overall logic of operation, which presents itself to the developer, tends to be somewhat more vague, since the scale or complexity of the system reflects on the logic. Trying to capture a very complex logic while maintaining consistency and completeness is an essential problem; for certain cases it is not even solvable. This motivates the relief of the specification from such global completeness and consistency, concentrating rather on the individual component control entities, leading to the concept of heterarchic control. Heterarchic control may be captured shortly by

Proposition 12 (Heterarchic Control System)
A heterarchic control system is a flat organization of control entities. Each control takes on its own responsibilities and initiates peer relations to other control entities. Control responsibilities and peer relations are autonomously and continuously updated according to internal logic and negotiation with peers.

The architectural concept of a heterarchic control system is very aligned with the paradigms of agent and multi-agent systems. The central common concept is that of ultimate autonomy in every entity. Heterarchic control systems share the advantage of agent systems when dealing with large or complex systems, which is that of an unconstrained decom-

position of the problem into a representation of numerous aspects; all of which being easily comprehensive both computationally and logically. This is in sharp contrast to the reductionist decomposition in hierarchical control, where a pure separation of concerns combined with a full coverage of all aspects of the strict operation specification is required.

Development, maintenance, and improvement of a heterarchic control system, where everything is characterized by dynamic connections and absence of architectural constraints between entities, has its apparent advantages. Systems implemented along the heterarchic or agent-based paradigms are viable wherever "best effort" or "intelligent response" are the only requirements. Such are typical of complex problems where no external requirements can be reasonably posed, and where just reasonable solutions or answers are requested of the systems. The problems of using the heterarchic paradigm for production control should thus present itself immediately. While it is easy to start implementing control entities that covers and take responsibility for different parts or different concerns of a production system, meeting the external expectations to overall performance gets lost in the process.

It is not the least difficult to imagine heterarchic systems for controlling very large and complex production systems, achieving full flexibility in principle, but where the lack of coordination, or inefficient execution, leads to vastly deficient performance, or even zero overall performance. In a multi-agent system, there is, by definition, no way of overruling the internal autonomy and decision competence of individual agents, and no agent may thus be controlled into submission by overall real time planning and production performance concerns. The shortcomings of the otherwise tempting paradigm of heterarchic control when met with external performance and coordination criteria, motivated the paradigm of holonic, or holarchic, control systems.

In a holonic system, a hybrid and dynamic mixture of the aspects of heterarchic and hierarchic systems is allowed. For a given instant in time, a holonic system may seem to operate as a heterarchic system, with all entities taking on responsibility deliberately and seeking meaningful peers for cooperation. At a later instant the same system, or parts of it, may have switched character, and then exhibit a hierarchic behaviour with entities acting like slaves for other entities, and thereby having their autonomy and integrity suppressed in certain respects. This is expressed in the following proposition

Proposition 13 (Holonic Control System)
A holonic control system is an dynamic organization of control entities, called holons. A holon is an agent of which a certain set of control facets must exhibit the ability to cooperate, be coordinated, and act autonomously. The precise dynamics of the modes is entirely up to the surrounding environment of holons and to the timely state and status of the local and global tasks, goals, and objectives.

As such, this is a very precise proposition, which basically just relieves constraints that are in effect in hierarchic and heterarchic systems. While it is clear that choosing a holonic

paradigm will enable the developers of a control system to work around the inherent limitations and problems with the heterarchic and hierarchic paradigms, it comes at the cost of a higher inherent complexity of the holons that must be consistently designed and implemented.

The presented propositions on control paradigms are quite clear on the architectural aspects. But in practise, it is doubtful if designers and implementers of hierarchic and heterarchic control systems are strictly following the paradigms. It may be conjectured that in almost any successful implementation of a complex, real-time control system that claims to be heterarchic, violation of the absolute autonomy and integrity of some agents take place. Likewise, it is hard to believe that an intelligent, hierarchic system for the real-time operation and control of a complex production system with numerous local uncertainties, is never in need of delegating considerable autonomy or decision control to subordinate components and allowing them a high degree of self-organization in a peer-to-peer manner.

In light of this, the paradigm of holonic control may simply express and extend what is more or less common practise. It may not be useless for that matter, since it will clearly help developers if they are not constantly believing that they are violating the overall paradigm or architecture of their solution system. It have also kicked off an entire community and forum of researchers that develop strategies and techniques for the design and implementation of holonic control systems.

2.5 From Flexibility and Batch to Agility and Chaos

Production control, considered as a service in a production enterprise, has traditionally been required to put dependability, predictability, and efficiency above everything else; e.g. flexibility, scalability, and maintainability. When facing these requirements in isolation from other concerns, the hierarchical or centralized control paradigm will stand out as an almost obvious choice for the organization of a production control system.

The gradual transition, during the past two decades, towards requiring production flexibility has left a cleft between production enterprise aspects and production systems. Flexibility is, to the production control system, for most part a matter of handling the increased number of static configuration states of production equipment and flow of materials. The states are static during the whole operation of a given setup of the production system for a given product; though a possible set of dynamic co-states may introduce, when manageable, an extra complexity of a dynamic nature for handling related variants of the pertinent product in the same setup.

Flexibility may be classified as a quality of a combined production and production control system as follows:

Proposition 14 (Flexible Production Quality)
Production flexibility is the quality to produce the whole or a large part of the product and variant spectrum of the production enterprise, in the same production system with massive reuse of production equipment across products and variants.

Central to the concept of flexible production is the batch production strategy:

Proposition 15 (Batch Production Strategy)
Batch production is a production strategy characterized by system-wide, time-separated periods of setup and producing. During setup, no producing takes place and all equipment is prepared for the next product type. During producing, nothing is changed in the equipment setup and a high number of goods of one specific type is produced.

Typical production in today's advanced production enterprises may be qualified as flexible batch-production. The notion of advanced production discounts the companies with trivializing extremities in their production systems, product spectrum, or transformation processing complexity:

- companies that have an extremely sparse product spectrum, e.g. one single product with no variation and simple transformation processes, and
- companies where the transformation processing and handling variations across the whole product spectrum are possible on the same equipment and within one and the same setup; e.g. only colour or inscription variations with no or little geometric variation.

As even the complexity of flexible batch-production may make a hierarchical approach to production control struggle to achieve stability and efficiency, it does not promise any better for the future. The last decade of development of the concept of agile manufacturing and agile production is beginning to break its way into the requirements to contemporary production systems.

Proposition 16 (Agile Production Quality)
Agility is the quality of a production control system and production equipment when any entity is able to, without prior or with only short notice, undertake the meaningful or necessary processing from the whole, or a part, of its capability-share of the product and process spectrum. All reconfiguration of the entity must be automatized and initiation must be triggered autonomously, without the requirement of stopping any other part of the entire production system.

Agility is a much stronger quality than the quality of flexibility. It is not hard to understand why agility is a quality to strive for when considering the desires from the following production ordering rules:

- Issue only internal production orders for items that can be associated with effective external customer orders.

- Immediately start internal ordering of production upon the issue of an external customer order.

The first of these rules express the vision of complete eliminating stockpiling based on prognostics, and the second express the desire for avoiding delivery delay due to high lot sizes. Only agile production may operate in simultaneous fulfilment of these rules while retaining some level of efficiency. On the other hand, a flexible production which fulfil these rules may be phenomenologically re-classified as agile.

If production agility can be efficiently achieved, the entire production planning and prognosis activities may be severely reduced or entirely eliminated. A direct technical beneficial effect implied by efficient agility is the ease with which load balancing of equipment usage may be introduced.

A desirable production strategy that may be deployed with attained efficient agility is chaotic production:

Proposition 17 (Chaotic Production Strategy)
Chaotic production is a strategy of production control characterized by absence of a global production plan. Production orders are created on the fly, scheduled on-line, and with routing, re-scheduling, and transformation processing decisions created in real-time during the course of the individual work orders.

The term "chaotic" intuitively carries a negative impression. In the context of agile production control, however, it is to be perceived as the successful mastering of chaos. An analogy may be drawn to intentional "inherent instability" or "relaxed stability" in aircraft design, allowing for increased manoeuvrability by an advanced control system. Likewise, allowing chaos in the planning, scheduling, and execution of a production system may be possible for a sufficiently advanced production control system. The production control system which can master a chaotic composition of work in progress and entangled material flows will likely be less disturbed by sudden changes to order priorities, order types, or breakdowns in devices, machines, cells, or lines. I.e. it will be agile.

The property of chaos may pertain to something internal to a system, emerging out of even quite simple and strict production control logic. This poses a challenge to flexible production systems that assume that the production operation can be maintained under strict control according to plans and schedules. For a discussion in mathematical terms, and examples of conditions for, emerging chaos in controlled production confer Haghighirad et al. [2008].

In a different sense, which is the one intended with the proposed definition here, chaos may pertain to the external expectations imposed on the production control system from the production enterprise. The production enterprise may simply reflect the fluctuating circumstances of customer demands, customer ordering patterns, and supply chain opportunities to the production control system. This meaning of chaos in production control is discussed by Rio [2007].

A chaotic production strategy, as proposed here, implemented with an agile production system is a vision and desire of most production companies that either have highly complex product spectra; high uncertainty or downright unpredictability in the pattern of customer orders; very short lifetime of product types or families; or all of these.

A hybrid of the chaotic and batch production strategies is called mix production. It may be characterized as a chaotic production strategy, but with high penalties, in terms of temporal delays, for the change of product type at some or several devices, machines, cells, or lines. The intelligent scheduling system of an agile production control will know this a priori, or experience it a posteriori for later application. Thus for certain situations and periods the production control will operate according to the batch production strategy but with changeovers flowing with the product type boundaries. By arrangement and orchestration of the changeover operations in a flexible production control, the same may basically be achieved. Thus, the mix production strategy is approachable from both flexible and agile production control systems.

The immense complexity that arises when changing from batch-, over mix-, to chaotic production strategy is very hard to capture and manage in a control system. Ultimately, to become efficiently agile, the flexible production control system must be evolved to automatically deal with the full complexity of the whole spectrum of products and variants, at any time, at any single device, machine, cell, and line.

At the historical advent of fully automated production some decades ago, the natural and superior agility of a manually operated or assisted production system evaporated. The agility was traded off for extremely "stiff", hard-automated, but efficient production lines. Basically, a production line could produce one single product variant, but at a high rate, controlled quality, and at a low cost. Increasingly varying customer demands induced the desire for regaining agility, but widespread production automation seems stuck with only flexible batch production in hierarchical or centralized production control systems.

As the hierarchical control paradigm is struggling even with flexible control of batch production, it is not far-fetched to project for the future that it either must rely on some basic innovation or be replaced by some new paradigm. The holonic production control paradigm may turn out to be a viable successor, but still has some way ahead, in terms of analysis and proof of concept, before being considered. Specifically, as long as no wide scale holonic production control system is known to exist, any TCO analysis will be more or less guesswork. Besides a TCO analysis, a wide scale prototype or pilot system further has to clearly indicated that such a production system can in fact operate satisfactorily; i.e. that the eventual loss of productivity is adequately balanced by the increased stability, flexibility, and agility. Until then, and as long as production companies are unwilling to or incapable of risking loss of productivity at the potential gain of agility, they are, it seems, stranded in the current production crisis in countries with high labour costs.

2.6 Automation Principles of Agile Production

The flexible and the agile strategies break with the focus on immensely stable, monolithic, costly, high-productivity solutions of past production strategies. In past paradigms of production control, and still widespread today, it was considered mandatory that any subsystem was of high quality, high reliability, and yielding high productivity, disregarding the cost. In the future it will be much more important that commissioned subsystems present highly flexible platforms for hosting intelligent or cognitive control systems allowing for automatic reconfiguration and operation.

As a traditional example, consider the installation of an assembly cell centred around one large, expensive robot, with long reach for covering voluminous magazines of component assembly parts. This would typically be the only installation for doing exactly these pertinent assembly operations on all products of the production. Hence the high requirements to quality, reliability, and productivity. These requirements lead to the development and tuning of the setups of the installation into an extreme level of complicated parameter interdependence and low tolerances. Very tight demands on the operation tact-time to the installation will leave no slack for sensory or cognitively based decisions and control. The final setups of the cell ends up in a set of "stiff" parameter configurations, for which no engineer will stand a chance of changing a single bit in any parameter without knowing every aspect of the control system to the lowest level of detail.

Such a system is naturally wanted for its resulting reliability and productivity, but unwanted when it comes to product upgrades, cross-product reuse, change of products and parts flow control, changes to magazine geometries, and every other aspect that change the circumstances for any production system entity. In the future, such a choice of subsystem principle must be preceded by an extremely careful analysis, showing with high certainty that the requirement on the productivity, the parts produced, and the immediate production environment for the system are kept fixed for some years ahead.

In today's increasingly fluctuating market-demands a positive outcome of the aforementioned analysis is decreasingly likely to be met. Rather than considering an alternative system principle, the traditional project described is simply rejected, and automation is never introduced for the assembly tasks. This situation may account for some of the general decline of production in the western economies with high labour cost.

A possible future alternative to the traditional production automation principle calls for less emphasis on requirements to quality, reliability, and productivity of individual installations. Instead, other qualities, less amenable to direct analysis regarding impact on productivity, may be gained. At the same total cost as commissioning of one large assembly cell based on expensive high quality equipment, five to ten smaller systems may be designed for solving the same assembly tasks, each having the same capability as the traditional system, but with much lower tact time. The combined productivity of the five to ten low-cost systems, however, may greatly surpass the productivity of the single-system design, and even at a lower total cost.

Several immediate beneficial impacts of such a change in automation strategy, mainly derivatives of the properties of redundancy and low cost, are given in the following list:

1. **Delayed emergency maintenance** is possible when any single redundant system may be automatically taken out of production on failure, with the other systems taking up the combined instantaneous productivity requirement.
2. **Live experimentation** for upgrades and changes on a real, on-line, complete hardware systems which has been partially taken out of production is a major benefit of having redundancy.
3. **Decoupled and unconstrained maintenance** scheduling becomes possible, since any one system may be taken out of production at any time, without disrupting the entire up- and down-stream production.
4. **Incremental commissioning** by replication of individual hardware and software setups allows gradual ramp-up to a desired level of productivity or redundancy.
5. There will be less pressure on optimization and hard geometric coupling, yielding room for more **intuitive and comprehensible system implementation**.

Intuitively it would seem an immediate problem that the engineering and maintenance burdens are proportional to the number of installations. This is naturally not so for replicated, near-identical installations, and this objection is further rejected by the property of comprehensible implementation (impact 5). By allowing for a much more intuitive implementation of the control systems of the individual installations, the processes of error seeking, debugging, and fixing of the systems will be much less cumbersome and time consuming.

The principles of delayed emergency maintenance (impact 1) and decoupled and unconstrained maintenance (impact 3) affect not directly the effort required for maintenance, but maybe more importantly the circumstances surrounding maintenance. In an emergency on a traditional productivity-critical installation, there is no maintenance scheduling; it has to be fixed immediately. For a solution based on several cells and redundancy, the failure of any single cell, which is automatically switched out of production, emergency maintenance may be deferred for hours, weeks, or even months, depending on the level of redundancy in the solution or the impact on overall productivity. Scheduled maintenance in a traditional system will have to be carried out in coordination with the surrounding production system and most frequently implies the total stop of production in the cell under maintenance. By decoupled and unconstrained maintenance (impact 3) in an agile system with redundant cells, any cell will be possible to schedule for maintenance based exclusively on conditions internal to the cell and availability of maintenance personnel.

Specifically with respect to total cost of ownership and performance optimization of the agile solution, the concepts of live experimentation (impact 2) and incremental commissioning (impact 4) are highly important. In a traditional productivity-critical setup,

there is very little room for experimentation on the live system for upgrading; either to meet product changes or for optimizing in the face of inferior productivity. By having a sufficient redundancy in terms of number of cells, or a period of lower requirements on productivity, any single cell of the agile solution may be taken out for live experimentation. Regarding incremental commissioning, in case of the final agile solution proving to give too little redundancy or too low productivity, or if at a later stage the overall productivity is desired to be increased, replication of the single cell systems may enable the minimum investment for meeting the immediate demands or desires. This is contrary to the situation faced with solution along the traditional principle, where a replication requires a doubling of the shop-floor spaces and equipment costs, while the required or desired productivity increase may be as low as 10%.

Chapter 3

Account of PhD Work

This chapter presents an account of work and results throughout the duration of the scholarship for this PhD thesis. The account is mainly thematically organized across the included publications in Chapter 4.

The fundamental themes that make up the core of the work in this thesis is succinctly captured by the following list:

- The use of GNU/Linux and standard computers in real-time and on-line distributed control in production.
- Real-time emulation of complex production devices; notably robots and AGVs.
- Device-level, real-time emulation of production systems for development of distributed, intelligent production control.
- Implementation of an experimental production control system for verification of the production emulation system.

3.1 Linux-PCs in Production Control

A common PC is considered as any computer based on, or developed from, the IBM PC architecture. For a couple of decades, since it's conception in the early 1980's, the standard PC has been generally considered too unreliable and fragile for use in control application by industry. Industrial control during those decades were predominantly based on the PLC hardware and logical programming platforms, offering much simpler programming and operating facilities. Much of the automation in manufacturing industry today is still based on PLCs.

Due to the inferior general computing facilities and performance in the typical PLC, PCs have been gaining ground on the shop-floor, adding comparatively tremendous computing

powers needed for the heavy information and data processing associated with intelligent and advanced control systems. This section is devoted to discussing how deep into the control loops the PCs may go, and to review some possible trends of the past decade in this regard. Specific attention is devoted to common PCs with GNU/Linux, since this has been the consistent and fundamental control and development platform for the work underlying this PhD thesis.

3.1.1 Issues with common PCs in Manufacturing Control

In comparison to simpler, rugged computing or logic devices such as PLCs, three common issues with standard PC may be stated succinctly as:

Software System Unreliability: Too high flexibility of technologies and programming paradigms in the development environment and at the operating system level.

Real-time Unreliability: Lack of real-time support in the operating system, as service to control applications.

Hardware Fragility: Sensitivity to the physical environment, i.e. moist, dust, temperature variations, vibrations, electromagnetic disturbances, etc.

On Software System Unreliability

The software system unreliability is here suggested to have nothing much to do with the PC platform itself. This thesis suggests that it rather has to do with the very flexibility and vast amount of available technology available with standard operating systems for PCs.

Unreliability emerge or evolve when developers build complex applications; especially when bad software design choices are made, the resulting software complexity may greatly exceed the fundamental solution complexity. The wide possibilities provided by the operating system, the development environment on the platform, and the technologies available for the development sometimes muddle an otherwise good solution design of the control application, simply by unfortunate mixing-up of technologies at the implementation and deployment level. Such a spaghetti-implementation will contain a fair amount of implicit mechanisms and constraints. While the initial system may be developed to become reliable in operation, it may exhibit a perceived unreliability during later software evolution stages, as the aforementioned hidden, implicit constraints and mechanisms surface. As indicated, this may be a *perceived* unreliability, whereby the original programmers may successfully evolve the system, but where any attempt by a non-initiated developer will fail. However, in severe cases not even the initial developers of the software system may grasp the implicit mechanisms, and hence the software system design, the operating system, and the PC platform are left open for being victimized as inherently unreliable.

It is thus suggested that it is the highly flexible software development facilities and the wide range of technologies that lure the programmers into building unreliable applications. This problem will rarely arise when founding the control application on PLCs, where the lack of advanced features in principle inhibit the design and implementation of complex applications. Furthermore, besides being quite simple, the restricted facilities for application development on PLCs are quite standardized. Therefore neither the initial developers, nor the new, non-initiated developers will find it difficult or complicated to perceive the mechanisms of a PLC-based application.

Of course, it is possible to restrict the development of control applications on PCs to the functionality provided by various PLC development and operation platforms. This is the essence of a *Soft PLC*, and the principle was indicated by Rullán [1997], though the case was purely for demonstrating how easily it was done on a standard PC with common development tools.

A principle of restricting the flexibility and technology of the facilities available to the control system designers and programmers may imply better reliability of the control systems produced. However, it is definitely not a principle that will help a production enterprise to evolve towards advanced and agile production control.

On Hardware Fragility

When considering standard PCs or personal laptops, these are mostly only shielded and protected for reliable operation in the friendly environment of an office. The standard office PC or personal laptop being the predominant form of the common PC, it is not strange that industrial production engineers often consider PCs too fragile to be commissioned in the shop-floor environment of a production system.

A short description of the operation-environmental issues for a standard PC is given in the following list:

Dust: Air-suspended solid particles, such as arising from burning, welding, wood-cutting, etc., or simply dust, will enter with the air-flow into any standard PC. Accumulation of such particles may eventually create short-circuits, or reduce ventilation and heat conduction. Given enough time, any PC will get a hardware failure from such effects.

Shock/Vibration: No part in the PC is suspended by a damped elastic mechanism, and thus shock or vibration may lead directly to fracture in fragile components. The common rotating disk hard drive is possibly the weakest point of the PC in this respect.

Electromagnetic: While some PCs have grounded metal plating all around the cabinet, mostly the components inside are quite unshielded from electromagnetic interference from large motor drives, welding equipment, etc., in the immediate vicinity in

an industrial environment.

Thermal: The operating range of the hardware components in a common PC is quite narrow. Having a standard PC operate at high CPU load for a moderate duration at an environment temperature beyond $30°C$ will bring it to the brink of failure; the CPU core temperature will rise quickly towards the limit of specification and the hardware monitoring system will either shut down the computer, or simply cut the power supply.

Humidity: There is no air-tight sealing of any kind in a common PC cabinet, and inside the cabinet, no component is sealed in itself in any way; with the possible exception of the CPU being encapsulated between the socket plate, on which the die is mounted, and the cooler. Any component within the cabinet is thus susceptible to the humidity of the air surrounding the PC itself.

Water/Moist: However high the humidity tolerance of any PC component, it will typically not be liquid tight. It is easily possible to shield a PC from trickling water from above, but randomly splashing water, or water impinging at high speed is a challenge. In addition, in an environment with any kind of water or moist, it is almost certain that the humidity will approach saturation.

In any shop-floor of a given factory a certain fraction of the floor area, or volume, may be classified as tolerable for standard, common, office PCs. This fraction may vary violently, from 0.0 to 1.0, depending on the industry or the nature of the manufacturing processes of the shop-floors.

To bring this factor closer to 0.0 motivated the conception of an IPC[1] as an environment-hardened version of the common office PC. An IPC has good electrical shielding, and a powerful, dust-filtered ventilation system. This meets three of the issues with the common PC mentioned above; the dust, electrical, and thermal issues. High grade of all hardware components may also be shock and vibration tolerant, dealing with that issue. The equilibrium of humidity between the inside and outside of an IPC can not be avoided, and is a natural consequence of the air exchange of the cooling system of most PCs. However, the individual components of an IPC are typically of a higher grade than those of the common PC, and an increased overall moist tolerance may be a grand result of the higher moist tolerance of all constituent components.

Stepping slightly away from the office-type PC system, there exist a large number of types of PC-compatible SBCs with all peripherals integrated on the single board, or even on the CPU chip. Most of the embedded versions, i.e. with a low power consumption and small form-factor, are entirely fanless systems with only passive cooling, allowing for complete sealing of the entire device. Only connectors are left as a potential vulnerability to water and moist. SBCs are mostly based on solid state or flash disk drives, and thus highly shock and vibration tolerant. A well-encased SBC may thus provide much of

[1] IBM were probably the first to release an IPC already in 1985, but called it an *Industrial Computer*; confer [IBM 1984 Archive Website]

the flexibility from the software technology, the operating systems, and the performance known from the common PCs, and they may be mechanically hardened to meet most or all of the physically demanding requirements for commission in the harshest of shop-floor environment.

Although SBCs are typically moderately inferior to the common mid-range PC regarding computational power, they outperform the standard PLCs in computational power. When they do not, it is probably because such PC-compatible, SBCs are found inside some PLCs today. An embedded computing device offering both the flexibility of a PC and the ruggedness and communication interfaces of a PLC is called a PAC; confer e.g. Josifovska [2004].

On Real-Time Unreliability

The association of PCs with absence of real-time performance may stem from the GPOSs predominantly used with PCs. These operating systems were multi-tasking systems without any real-time scheduling or preemption support, and thus no guarantee of determinism could be given to any process.

Without going into details of the vast literature on the matter of RTOSs for the PC platforms, it is safe to state that most RTOS-approaches with PCs have worst case latency well below $100\mu s$. This should be sufficient for most applications where PLCs are being used today, with the possible exception of control applications of very low timing-tolerance.

Already in the early 1980's, hard real-time systems were developed for the PC platforms. As an example, the hard RTOS OS-9 was initially released for the Motorola architectures around 1980, and towards the end of the 1989's it was ported to the Intel x86 architecture in a version called OS-9000. Of newer, proprietary, pure RTOS systems that are easier to learn and bring to use may be mentioned the Windows CE and QNX.

OS-9, Windows CE, and QNX are entire and pure, "self-contained" RTOSs. VxWorks is another dedicated RTOS, which is not self contained. It requires a host system for development and building, with the built image being deployable on the target system. This approach to RTOS highly resembles the development and deployment method for common microcontrollers, where the built or compiled binary program is flashed on the target microcontroller. This may render VxWorks more efficient in operation on computers with scarce memory and computational resources, but increases the development burden and run-time flexibility.

During the past decade various extensions, such as RMX and INtime, to some of the Microsoft GPOSs bring hard real-time capabilities into the operating system, side by side with the GPOS system. Likewise, dual-kernel approaches for Linux will host the user-level GNU/Linux GPOS system side by side with an RTOS environment. Such approaches makes it easier to integrate GPOS facilities relating to high-level operation logic with

RTOS facilities relating to low-level device or process control. However, systems based on separate concurrent kernels require the separation of real-time tasks from the general software, at every level from source code to deployment. And the development and run-time environments provided by the RTOS may be very different from those known from the GPOS. Hence, it still requires special expertise and a different set of skills to exploit the RTOS capabilities, like was the case for development for pure RTOSs.

Recent development effort are closing in on hard real-time capabilities from normal user-space processes in ordinary GNU/Linux systems. This effort is based on real-time pre-emption patches in progress of being integrated into the main line Linux kernel. This will be discussed shortly in Section 3.1.3.

In summary, the state-of-the-art operating systems and hardware for the common PC, should present no hindrance of use in applications tolerating up to $100\mu s$ worst case latency in the control loop.

3.1.2 PCs to Take Over or Inter-Operate with PLCs

As PCs are entering the control loops at the factory floor, it is interesting to shortly review some of the visions and emerging standards, as well as the progress towards implementations, regarding more sophisticated control systems. The main effort in this area is motivated by allowing for much more advanced computational performance and integration at the production control level. This is assumed to be a necessity for developing highly flexible or agile production control systems, and naturally directs attention towards the use of PCs.

It was already suggested by Rullán [1997] that PCs may eventually entirely take over the functionality of PLCs in production automation, for all but the simplest PLCs. The simpler PLCs may remain cost effective for distributed applications where a remotely located control component is to implement a very simple IO mapping.

The IEC 61499 standard for distributed control in manufacturing aims at formalizing and specifying the flexible inter-operation of the distributed control elements, on platforms such as PCs and PLCs, at the control system level. Thramboulidis [2009] gives a critical overview of the results of the standard and the relation with its ancestor IEC 61131, accounting for the absence of adoption by industry. This standard clearly aims at the co-operation between PCs and PLCs, rather than PCs fully taking over the domain of PLCs. The standard was developed to meet the limitations of previous standards and engineering practises in the face of the requirements to higher flexibility in the manufacturing control systems.

O³neida [O³neida Website] is an open, cooperative network under IMS[2] for supporting and furthering the development of distributed automation and control, based on open

[2] Intelligent Manufacturing Systems [IMS Website]

standards. Auinger et al. [2005] gives a good description of the views and philosophies on which O^{3}neida is founded. Hegny et al. [2008] developed an IEC 61499-compliant run-time system within the 4DIAC project [4DIAC Website] for integrating an agent-based control system with low-level device control.

In conclusion, there is not yet any clear tendency regarding the PC versus PLC battle for the future of advanced production automation. The following extreme scenarios may all be realistic to some level:

- The PC may take over all control aspects and situations on the shop-floor; as hinted by Rullán [1997].
- The successful adoption of either the IEC 61499 standard or the IEC 61131 standard with advanced extensions may help separate the control domains for inter-operation and cooperation among PCs and PLCs; as per the purposes of these standards according to Thramboulidis [2009].
- The PLCs, based on the IEC 61131 standard, will evolve into increasingly powerful platforms with stronger and more flexible operation and programming facilities. The production control systems will then evolve with the PLCs and their facilities will gradually meet the demand for higher flexibility and agility on the shop-floor.
- A new hybrid between hardened PC and advanced PLC, such as the PAC mentioned by Josifovska [2004], may emerge as the predominantly used standard computing device in production automation. This may squeeze itself in-between the PCs and the PLCs, pushing them back to where they came from; PCs back to the enterprise and office systems and PLCs relegated to the simple IO-signal control.

There is, however, nothing contradicting the co-existence of all of these outcomes, and many more on top of them. Perhaps in-house competences will once again. in the near future, prevail in manufacturing companies, reducing the need for sticking to strict standards and a uniform practices of automation engineering.

3.1.3 Real-Time Control with GNU/Linux

As touched upon in Section 3.1.2, there exist a wide range of GPOSs and RTOSs suitable for different objectives. Mostly these are disjoint sets; an RTOS being ill-suited for user and office applications, while a GPOS is ill-suited for real-time control applications. For many control tasks in a production control system the timeliness provided by a GPOS, i.e. mostly none, is not sufficient.

In addition to the pure RTOSs mentioned in Section 3.1.1 the GNU/Linux based platforms RTLinux, Xenomai, and RTAI are worth mentioning. They are based on an approach where a micro- or nano-kernel is the fundamental OS on the host computer, ensuring the timeliness of the real-time tasks while hosting the GNU/Linux system with

low priority as a preemptive process on top. The tasks developed and deployed as real-time tasks in the micro/nano-kernel is thus guaranteed hard real-time performance, while the Linux kernel and user system itself may be scheduled as one low-priority, preemptive task in its entirety. This dual-kernel approach clearly distinguishes the programming, compile-time, and run-time environments of the real-time programs from that of an ordinary GNU/Linux system. Some system level features of the approach facilitates the interaction between the real-time tasks and the processes of the GNU/Linux system, but at the programming and deployment level, they are two separate worlds.

A different approach, now known as Real-Time Linux[3], has been an ongoing activity for the past decade, where patches to the Linux kernel makes it, almost, fully preemptible. Together with advanced developments, e.g. in scheduling and high resolution timers, the Linux kernel can today be easily patched to become an RTOS. It is noteworthy that this RTOS-property comes with almost no impact on the programming, compile-time, or run-time environment as presented to the application programmer, and without losing any of the features of its GPOS nature, and at the cost of only a minor general performance loss. Of course, it is then up to the application designers and programmers not to ruin the real-time performance by bad choices of technologies, design, and implementation.

At the onset of the presented PhD-work in this thesis, it was generally assumed that many control tasks would require low-latency real-time execution; e.g. development of external robot motion control and low-level control of AGV motor controllers. Therefore, some effort was invested in studying the real-time performance of Real-Time Linux, and with good inspirations by work such as that of Dallefrate et al. [2005]; though they used RTAI and QNX.

Eventually it turned out that the standard Linux kernel, and even the Python interpreter platform, [Python Website], was adequate for meeting the latency and real-time requirements for all experiments and developed applications. In spite of many runs of control applications with the standard Linux and Python setup without any instabilities accredited to lack of real-time performance, the lack of worst-case *guarantees* may not stand when taking such applications to industrial use. Therefore some valuable reference and thought on the matter is presented in this section.

Real-Time Linux, the RT Patch

The mainstream Linux kernel is currently almost to characterize as an RTOS. By configuration options in the build-system for the standard kernel, it is possible to select almost all required kernel features for making it an RTOS kernel. This is due to an effort in the past few years, where most of the "RT patch" has been merge into the mainstream kernel. A few features making the kernel fully preemptible are still left out, due to their

[3] Not to be confused with the older RTLinux.

compromising the general computational performance of the kernel. Thus, to achieve a high quality RTOS with standard Linux, it is still necessary to patch the kernel source.

The core source of information on the RT patch for the Linux kernel is the Real-Time Linux Wiki [Real-Time Linux Wiki Website]. This is the home of the RT patch with good instructions on how to apply the patch and build the kernel, reported benchmark results, applications, guides, etc. The patch itself is (mainly) developed and maintained by Ingo Molnar and Thomas Gleixner, but with a long history of the principle and many contributors working on different forerunners to the RT patch.

The "Open Source Automation Development Lab" [OSADL Website] is an industry-oriented organization for the further development of the real-time Linux kernel and its industrial application; notable for process control and automation. The website and the organization itself are highly relevant sources of information and resources for getting to terms with Real-Time Linux, especially for production automation.

Specifically on Debian-based systems, it is possible to install pre-built, RT-patched Linux kernels from Pengutronix [Pengutronix Website]; a company providing services for embedding Linux in industrial applications. Such pre-built kernels makes it very easy for production control engineers to quickly start developing applications with Real-Time Linux, without the trouble of first understanding the mechanisms for patching, configuring, and building a custom kernel from source.

Dietrich and Walker [2005] and Rostedt and Hart [2007] present good, thorough overviews of the issues met with the development of the fully preemptive Linux kernel, and how they were dealt with; the latter going into some details closer to the code. The mechanisms involved with the RT patch are well described.

Arthur et al. [2007] present some benchmark measurements of Real-Time Linux of high relevance to control in automation. The most important measurement setup is one where a pin on the parallel port of the PC must be kept high for $50\mu s$ at a cyclic rate of $1kHz$. The results for the highest performing hardware setup, based on a $2.8GHz$ Intel P4 processor, with the RT patch show a scheduling latency in the range of $5\mu s$ to $10\mu s$ and a jitter of $24\mu s$. For comparison they made the same measurements for RTAI and RTLinux, both based on micro/nano-kernels, showing no latency and around $15\mu s$ symmetric jitter.

The measurements of Arthur et al. [2007] illustrate that Real-Time Linux still has some way to go to become a high-quality RTOS. They emphasize that the biggest problem for Linux to become a hard RTOS is that of absence of guaranteed worst case latency. The complexity of the Linux kernel makes it impossible to formally prove a bound to the worst case latency, and only trust may be gained by extensive testing under very varied conditions. However, as also indicated by Bruzzone et al. [2009], Real-Time Linux has come a long way and is usable for many real-time applications in its current state.

Embedding

Embeddability, defined as the degree to which an OS lends itself to embedded applications, plays a crucial part for GNU/Linux to becoming widely accepted in industrial applications of production automation control. Fortunately this is one of GNU/Linux' major advantages over many other GPOSs. Thousands of hobby and research projects favour GNU/Linux as their embedded OS. Many consumer electronics devices, such as Ethernet routers and media players, are based on customized, embedded GNU/Linux systems; confer e.g. [Linux for Devices Website] for a comprehensive list of such examples.

Techniques, methods, principles, and guides for embedding GNU/Linux are abundantly represented in literature. Sally [2010] presents a comprehensive and general work on embedding Linux. He states, in the opening sentence of Chapter 1, that

> *"Linux is an incredible piece of software. It's an operating system that's just as at home running on IBM's zSeries supercomputers as it is on a cell phone, manufacturing device, network switch, or even cow milking machine."*

The OS quality of being embeddable relates to a series of aspects which may have various importance depending on the application. Following is a non-exhaustive list of aspects included under embedability:

- system miniaturization of installation in terms of memory and disk footprint, as well as CPU performance requirement;
- systems robustness;
- range of supported computing hardware platforms;
- range of supported peripheral devices;
- real-time performance;
- power consumption and heat generation.

Clearly, many of these aspects are not just pertaining to the OS, but rather shared among OS, application environment, and hardware platform. It is noteworthy, and not coincidental, that on most of these aspects standard PLCs score high marks; which is the very reason for their widespread use. However, as previously mentioned, PLCs have a relatively limited application and execution environment, low computational power, and do not support the wide range of flexible application technologies that come with GPOSs like GNU/Linux.

Sally [2010] gives a quite comprehensive guide for embedding GNU/Linux, covering many aspects from download and configuration over application development and debugging to cross-compilation and deployment.

Industrial Communication

An important part of production control is the direct, external control of the machines and devices, which is often handled by PLCs in contemporary production automation. To actually perform the external control of the devices and machines, it is obviously necessary to have the means of communicating with them. To this end, one of the major advantages of PLCs are their support of industrial communication, i.e. fieldbusses. If GNU/Linux systems on PCs should pervade down through the PLC-level of device and machine control, rather than remain in a higher level layer, it is necessary to establish that GNU/Linux is indeed capable in a wide range of industrial communication hardware and protocols.

GNU/Linux supports a lot of the prominent and widespread industrial communication protocols, such as EtherCAT, PROFINET, PROFIBUS, Ethernet POWERLINK, CANOpen, ARCnet. OSADL has a separate activity for unifying the major fieldbusses in a layered structure[4]. The German companies Pengutronix and Steinhoff [Steinhoff Website] are very active in the development of fieldbus support on Linux.

For illustrative purposes the following give examples of reports on support of a couple of prominent fieldbusses in GNU/Linux.

Kastner et al. [1999] claim to have implemented the first PROFIBUS DP master driver for GNU/Linux. The emphasis of their work, however, is on a developed gateway on the GNU/Linux host connected to the PROFIBUS DP network, giving access to Internet-connected control clients.

Khanh et al. [2009] present the "PBMaster" project, which aims at a software implementation of PROFIBUS DP. It is an entirely open implementation under the GPL. Many details of the API to the fieldbus data link, kernel modules, as well as example applications are presented. It is reported as a work in progress towards certification, supporting the basic DP-V0 specification for cyclic data exchange. A gateway is implemented for Internet connectivity of remote control clients to the PROFIBUS network. Graphical interfaces for monitoring and analyzing the PROFIBUS communication are provided by the project. For the ease of testing and evaluating, or even using, the implementation, the project provides a complete OS system on a Live CD.

Baumgartner and Schoenegger [2010] present the use of openPOWERLINK, an open implementation of the Ethernet POWERLINK stack on Linux. It takes advantage of the RT patch and the High Resolution Timers in Real-Time Linux to realize a POWERLINK master node. A test environment is detailed, and performance results in terms of correlated cycle times and system loads on the master node are presented for two different master node PCs, and with varying number of control nodes. It is concluded that Real-Time Linux is indeed suitable for hosting a POWERLINK master node.

[4] Confer e.g. the presentations from the "OSADL Fieldbus Framework Meeting", November 26, 2008, at [OSADL Website]

Soft Real-Time Control with Python

All the experimental control activities reported in this PhD thesis were started by prototyping with the Python Programming Language, confer [Python Website], on a stock kernel from the Debian GNU/Linux distribution. The contingency plan for each activity, in case of inadequate performance or timing issues, was to migrate first to a C/C++ implementation. In case of persisting timing issues it was further planned to move to an Real-Time Linux kernel; e.g. a kernel from Pengutronix[5].

As physical devices were to be integrated in control systems, the project activities would need to address the fieldbus interfaces provided by the physical devices. It was expected that some or all of PROFIBUS, PROFINET, and Ethernet POWERLINK were necessary during the course of the work.

As it turned out, Python programs executing on moderate-performance PCs running a stock Debian GNU/Linux system fulfilled all performance and real-time requirements of the control tasks. Remarkably, this was even achieved without any optimization, such as shutting down unused standard services and applications, and while running advanced desktop managers on top of the X Window System. None of the devices controlled eventually imposed the need of special industrial communication. Only UDP and TCP Ethernet, and UART/RS232 communication have been used.

It is inferred that the hardware and software technologies used, with the notable bottleneck expected to be the Python Interpreter processes, are capable of performing well in a $100Hz$ control cycle. Approximate figures for the real-time performance is suggested to be $1ms$ fixed latency and $1ms$ asymmetric jitter (towards delay) over an UDP connection; amounting to some $2ms$ to $3ms$ average latency, but without guarantee. As all applications have required only soft real-time performance, it has not been possible to observe any critical glitches in control deadlines. As no performance deficiencies have been observed, the actual real-time performance of the hardware and software configurations were never quantitatively measured.

3.1.4 Examples of PC Control for Devices

With reference to the included papers in Chapter 4, a some examples of PC control of industrially relevant setups and equipment is given in the following.

Example: Fixture and Welding Robot Operation

Section 4.1 [Lien and Lind, 2008] presents an industrial setup where a PC is integrating and controlling a welding fixture with a welding robot. The fundamental problem

[5] Pengutronix provides pre-compiled, RT patched Debian GNU/Linux kernels from their website, [Pengutronix Website].

addressed by the work was the offset correction of weld seams to be measured simultaneously with the controlled automatic clamping of the workpiece.

The clamps, acting from the top and two orthogonal sides of the workpiece, were pneumatically operated, with the valves controllable through a simple service program on a micro-controller board. For the measurements, high-resolution incremental encoders were mounted on the axes of two arms clamping in the horizontal direction. The micro-controller was continuously keeping the pulse-count of the encoders and made these accessible for reading out.

An RS232 connection between the PC and the micro-controller enabled the PC-software to read instantaneous encoder pulse-counts and command the states of the pneumatic valves. At the other end, the robot controller was also setup with a program communicating with the PC on an RS232 connection. The robot controller program listens for submission of weld seam corrections and for a command for starting the execution of the welding process.

The entire setup and operation of the welding cell was handled in an application on the control PC. The application was implemented Python and graphical interfaces for operator interactions were based on the Qt framework [Qt Website]. It covers the following tasks:

- calibration of encoder readings to geometrical offsets;
- specifying and executing the clamping procedure;
- computing offset corrections from sensor readings;
- sending corrections and welding commands to the robot controller;
- and presenting several graphical interfaces for operation and monitoring.

The solution emphasized the removal of production control and strategy from lower level devices, and placing it in a more accessible software application on a PC. The implementations in the robot controller and in the micro-controller were stripped down to essential process control signalling and sensing.

The application control software on the PC is kept with a clean separation of functional code and graphical interface. The implies that the system may easily be beheaded of its graphical interface, which could then be replaced by a network-exposed interface. This interfaces would then allow the welding cell control to be directly with the wider production control system; on-line and in real-time. Such an automated, remotely controlled commissioning will eventually be desirable, as confidence in the welding cell control application has been established.

For efficient development, refinement, extension, run-in, and integration with the production control system, the flexibility of the PC platform, the GNU/Linux operation system, and the Python programming platform may play a major role.

Example: External Robot Motion Control

Section 4.4 [Lind et al., 2010] presents some measurements of achievable control characteristics for three different industrial robot controllers. The measurements were obtained by real-time control from an external PC connected to the industrial robot controllers, over regular Ethernet.

It is suggested how it is possible to take over the entire application and motion control from the proprietary robot controller platform, presenting a framework for motion control in a much more flexible computing and programming environment to the application developer.

The results show that the real-time characteristics and performance of the addressed low-level controllers in the native industrial robot controllers are very different. The best performance obtained was a measured tracking delay of about $12ms$, suitable for many sensor-based motion control applications. The worst performance showed a tracking delay of about $120ms$, disqualifying the robot from applications demanding fast physical response to sensor system readings.

The paper in Section 4.5 [Schrimpf et al., 2010] presents an application using the principles of external motion control that was described in Section 4.4 [Lind et al., 2010]. The application uses vision-based identification of a point on a marked line together with the inclination of the curved surface surrounding the point. The control task of the application is to move the robot tool centre at a specified speed along the line, while maintaining a normal orientation of the tool approach direction to the local plane of the surface.

The application did not involve mechanical contact, and no external metrological systems was used for measuring the precision of tracing the marked path or the deviation of the orientation from normal. While using a robot controller with a high tracking delay, this did not obscure the performance towards instability. As a system for testing of principles for sensor-based motion control, and for taking advantage of a flexible framework for motion control, it can be characterized as successful. However, as it was set up with a robot with long tracking delay, the specific system as a whole would only be useful in highly tolerant applications.

In the cardinal paper included in Section 4.10 [Lind and Schrimpf, 2011], the principles and design of the PyMoCo motion control framework are detailed and described. This is the motion control framework which is also underlying the papers in Section 4.4 [Lind et al., 2010] and Section 4.5 [Schrimpf et al., 2010].

The main results from the work with PyMoCo are capitalized in the following list:

- External motion control from an ordinary PC with an Ethernet connection to an accessor element in a native controller for an industrial robot is possible.
- For prototype or pilot installations the presented framework, PyMoCo, based en-

tirely on the Python programming language and platform, may boost development and integration.

- In non-critical production installations, PyMoCo will be an adequate choice for robot control in terms of performance, and a preferable choice in terms of maintainability, flexibility, comprehensibility, and integrability.
- For high-performance real-time applications, a more comprehensive motion control framework should be chosen; e.g. Orocos as described by Bruyninckx et al. [2003].

The PyMoCo framework as presented in Section 4.10 [Lind and Schrimpf, 2011] is not in its final version, being under active development and improvement. As mentioned in the paper, a refactoring and slight redesign of the communication mechanisms is in progress.

PyMoCo was initialized as a development effort in direct consequence of a project for gaining access to the low-level controller of the Nachi AX10 robot controller. The software that started off as a testing system for the low-level control of the Nachi SC15F robot eventually became a more general software framework for robot motion control implemented in Python. The latest development has resulted in the successful motion control of the Universal Robots UR-6-85-5-A industrial robot.

The technical development of robotics control software, such as PyMoCo, is ultimately relying on fundamental mathematics, modelling, and applications of robotics; e.g. the seminal works by Craig [2004] and Schilling [1990].

Example: Custom Built, Free-Roaming, Low-Cost AGV

The AGVs and AGV systems for production logistics are often closed for deeper integration by proprietary protocols, just like industrial robot controllers. For AGVs as for industrial robots, this may be an advantage in the simplest forms of use, where the scenarios of application control considered by the system supplier are sufficient for a given production use. But this rarely covers the wish for deeper integration that takes advantage of the full flexibility of their electromechanical capabilities.

Free-roaming AGVs must be unleashed, as must general industrial robot manipulators, when intending to bring a production system towards agility. For a fully agile AGV system, there should not even be soft-coded, fixed routes in the production area. In principle, routes should not even be represented a priori, but rather inferred by experience by the AGVs themselves, and gathered by the distributed AGV system for sharing among all associated AGVs. With the holarchic control paradigm as key to bringing agility to production systems, redundancy and intelligent cooperation of the AGVs become conflicting with the centralized control and high unit-costs of commercially available solutions.

Such considerations spurred the development effort that led to the design ideas presented in the paper included in Section 4.3 [Roulet-Dubonnet et al., 2009]. It presents a pro-

totype AGV which is inexpensive and where the task, application, and motion control systems at the individual AGV level are hosted on-board on a standard laptop PC. The laptop communicates with the surrounding AGV system over standard wireless Ethernet.

3.2 Real-Time Production Device Emulation

When implementing the control of a device, or a system of devices, it is a tedious, time-consuming, and often a risky process to work with the hardware system itself. When separable from the low-level device control, it is highly efficient for development and testing of certain aspects of the higher level control to run it on real-time, emulated low-level control.

The realistic emulation of low-level device control has been an activity in which much effort has been invested as part of this PhD work, and fortunately it has yielded a proportionally great success. It has been fruitful as an independent activity in itself, and the results have been serving their purposes in helping or enabling other activities in the PhD work.

3.2.1 Considerations for Real-Time Device Emulation

When developing a control system for an electromechanical device, comprising actuators and sensors, it is typically arranged in a cascade of control loops integrating various levels of sensor inputs with various abstractions of control. Such are the circumstances of control systems around motors in AGVs, servos of robots, and arrangements of pneumatics, motors, and sensors in machines. For motors, at the lowest level, the control objective regards the current passing through the motor windings. By knowing the characteristics of a motor, the torque that the motor exerts on its axle at a given current may be computed. By knowing the gearing on the axle a rough output torque exerted on the external mechanical system may be computed. In case of an AGV, knowing the diameter of the wheels will give a good estimate of the force exerted by the floor on the AGV at the contact point of the wheel. The knowledge of the geometry and inertia of the AGV may then lead to good estimation of the resulting acceleration of the AGV. Of course, unmodelled and non-deterministic effects, such as time-varying friction, enters at every level, making it impossible to obtain acceptable accuracy with pure feed-forward control.

Rather than explicitly controlling the full acceleration of the AGV by motor currents, it is more common to have an encoder on the wheel axle and then control the motor current to a given rotational velocity of the wheel. By knowing the encoder resolution, and the AGV geometry, the AGV may be controlled to a trajectory in terms of velocity commands to the motor controllers. Development of an advanced velocity controller may be based on realistic emulation of the physical system, where correct or realistic quantitative properties of electrical, mechanical, and geometrical nature will help experiment

with the workings of an implemented controller. At the level of the velocity control loop over motor current, emulation and experimentation is typically performed with high frequency, high-performance software simulation systems based on advanced physics-based mathematical system models.

If this level of control of the AGV is established, and when such limits as slipping of the wheels are respected in the controlled region, the trajectory controller simply has to command updates to the velocities in a timely manner. Specific for the problem of AGV control is the problem that it is non-holonomic. As a consequence, errors accumulate, and the feed-forward velocity scheme of trajectory control diverges. The trajectory controller needs feedback on the position and orientation of the AGV, in manner to incorporate error compensation into its velocity commands.

It can be argued that much of the development of an external control system may be performed with a simulation of the controlled device, which can be described as a coordinated scenario of interactions which are played back for the controlling system. Such a scenario would be set up in expectancy of the specific commands and events from the controlling system, while it is fed back the consistent measurements, outputs, events, etc., from the controlled device. When a given system-aspect of both the controlling system and the controlled device may be isolated from all other aspects for testing purposes, it can be realistic and efficient to use such a system simulation for development. However, the highly distributed and autonomous nature of complex devices, such as AGVs, and advanced control systems, such as AGV system and AGV controllers, renders isolated simulation of specific aspects a very questionable endeavour.

It may be more efficient and more flexible to implement a consistent and complete set of aspect in a realistic emulation, which is not locked to a complex prepared scenario of playback. For control system simulation with realistically emulated devices, the scenario rests entirely with the logic operation of the controlling system, e.g. AGV controllers and the AGV system, and the implemented behaviour of the emulated devices, e.g. the AGVs.

Several techniques exist for global localization of AGVs. If implementing any one of these, it may be done as an emulated system together with the emulated velocity controllers of the AGV. The trajectory controller is now the target for development, and for a sufficiently realistic, real-time implementation of emulators for the velocity controllers and localization system, the real trajectory control system may be efficiently tested in the emulated setup. Even the connection technology of the trajectory control system may be the same for the emulated as for the real systems, and the trajectory controller is left entirely oblivious of the fact that it is not connected to the real devices.

Having developed a satisfactory trajectory control system for the single AGV, a collection or fleet of AGVs may then be considered as an emulated device. Over a real-time emulation of the collection of AGVs, real systems for task scheduling, traffic control, and collision avoidance may be the next targets of development and experimentation.

Much the same scenario can be achieved for robots, conveyors, and other machines. It is of cardinal importance to find the correct level of separation between the controlling and the emulated systems. This level of separation is determined by such factors as

- the facilities of the underlying technologies used for making the emulator;
- the communication types and frequencies between the different cascaded control systems;
- and the target control system to be experimented.

The emulated system for a device may in principle be anywhere from individual motor current control to the highest level of motion planning.

A true emulation of a standard industrial robot controller, as commissioned by the end-user, would contain the implemented code for the robot application control, and the target control system is the external application control system which is controlling the deployed robot application. In such a simple setup, the external orchestration of a pick-and-place operation could be experimented with, if the application control in the emulated robot controller is set up to listen for positions and orientations to hold a mounted gripper, and triggers for opening and closing it. This scenario may even be possible with an emulated controller supplied with the robot by the manufacturer.

For more advanced control scenarios of industrial robots, where application and motion control is tightly integrated with sensor and process control systems, the emulation level of the robot has to be lowered to the low-level control of the robot. This is where the interface between the emulated device and the external control system starts to put up real-time constraints. In case of hard real-time requirements from the emulated system, the controlling system must meet these, and in case of soft real-time requirements, the performance of the controlling system depends on the level to which it meets the requirements. The development efficiency of advanced control systems, integrated with sensor and process control, will benefit highly from initially using realistic, real-time emulation of the lower level systems. Naturally, the development with the emulated system should only proceed along any given aspect to the extent with which that aspect is realistic in the emulator. The stage where the control system performs well enough, and all that can be realistically tested on the emulated lower level system has been tested presents a breaking point. At this point, either the emulation must be refined to become more sophisticated and realistic or the experimentation must be taken to the real, physical system.

3.2.2 Real-Time Emulation with the Blender Game Engine

The animation studio Blender [Blender Website] integrates a game engine which has advanced physics capabilities via the Bullet Physics Library [Bullet Physics Website].

The modeller of Blender allows for advanced mesh modelling, and integrated setup of game and physics properties of all objects in a scene.

The applicability of the Blender game engine lies with good performance, visual real-time display, realistic physics interaction, and an extensive run-time API exposed to Python. The run-time API enables an almost unrestricted control of the game engine by embedded Python code. The unrestricted execution of the embedded Python code presents a very flexible mechanism for presenting a realistic real-time emulation system to an external control system.

Every object modelled in the scene in Blender is turned into a game object when the game engine is started. A game object is characterized by:

- The **object type**, such as static, non-colliding, dynamic, soft body, and rigid body. The type of a game object is highly determining for its run-time behaviour in the game engine.
- A set of **physical properties** such as inertia, motion damping, interference radius, collision boundary object, friction quotient, collision elasticity, etc., depending on the kind of object.
- A configurable map of general numerical or string **properties**.
- A set of **game object sensors** for triggering controllers under various circumstances such as collision, proximity, timed delays, message reception, property changes, keyboard inputs, mouse events, etc. Many sensors are mostly useful when implementing the entire logic through the Blender interface. For advanced control setup, however, mostly the sensors of a temporal or geometric nature are useful.
- A set of **game object controllers** which are activated by triggering from the game object sensors connected to it. A game object controller is not restricted to be connected to game object sensors from its own game object. There are various simple sensors for Boolean operation on the inputs from connected game object sensors, which when evaluating to true triggers the associated game object actuators. However, for advanced control, the type of game object controller which invokes a function in an externally implemented Python module is the most heavily used.
- A set of **game object actuators**, which may perform various operations on the game object such as object destruction and creation, change of properties, change of motion, change of forces, change of position and orientation, etc. Most of what can be performed by game object actuators is also possible from Python code invoked from the game object controllers.

These are the primitives with which an emulated system can be set up with the Blender game engine. These are, however few in number or limited in extent they may seem,

together with the flexibility and technologies available to Python itself and the large and comprehensive API to the game engine run-time system, enough that for most part it is the fantasy that limits the devices and systems that may be realistically emulated. Of course, there are always performance limits of communication bandwidth, CPU resources, GPU resources, RAM, etc., which may ultimately limit the level to which a device may be emulated.

So long as the frequency of the logic ticks is adequate for the internals and the control of the device emulated, there are very few restrictions on the nature of the devices that can be emulated. The most notable limitations are concerning the Bullet Physics Library, which has its natural limitations in precision and its natural limitations in the natures of physical interactions. Some of these limitations of the Bullet Physics Library simply stems from lack of computing power, to which there is an easy remedy in buying more powerful CPUs. Other limitations pertains to the absence of certain physics features or effects, such as static friction, which wholly excludes a range of emulation applications from the category of being easy and straight forward to implement.

3.2.3 Examples of Real-Time Emulated Systems

As part of the presented PhD work a robot motion control framework has been developed. Some work effort was also contributed to designing and building an AGV prototype with simple vehicle control, and the development of a prototype AGV system for operating a collection of AGVs. The robot motion control framework and the development of the AGV system were both greatly boosted in efficiency by the early establishment of realistic emulation models.

Example: Robot Controller Emulation

Experimenting with external, real-time motion control of an industrial robot is hazardous for the robot involved and for surrounding and mounted equipment. Every live run with the robot controller must be preceded by thorough preparations; as always when developing robot applications. However, when interacting with the low-level controller of the robot in real-time with experimental code, even more thorough and comprehensive precautions should be made.

This factor of cautiousness presents an immense hindrance to efficient and quick development, and may be greatly alleviated by realistic emulation of the low-level robot system. The robot motion control framework described in Section 3.1.4, and in detail in the papers included in Section 4.10 [Lind and Schrimpf, 2011] and Section 4.4 [Lind et al., 2010], owes much in terms of development efficiency to the early setup of a realistic emulation model of the lower level of the industrial controller.

The initial development of the PyMoCo framework for motion control was targeting the interface of an accessor component deployed on a single-board computer in the Nachi AX10 controller. From an external control PC, through the accessor, the low-level controller can be commanded in $100Hz$ with position updates, while synchronously receiving emitted actual joint positions from the accessor.

The implemented emulator system for the low-level controller and accessor is, as with almost any emulation, realistic only up to a certain limit. Some deficiencies from fully realistic emulation are intentional for reasons of simplicity. An emulation for experimenting with a certain aspect of the external control system may be oblivious to a whole range of complex features of a highly realistic emulation. Other deficiencies are limited by hardware performance of either the node running the emulation or the node running the external control system.

Examples of deficiencies for the low-level robot control emulation implemented to support the development in the paper in Section 4.10 [Lind and Schrimpf, 2011] are listed in the following.

- The tracking delay and response time discussed in the paper in Section 4.4 [Lind et al., 2010] are not modelled. Modelling these would not have presented any serious challenge, but was left out since the purpose of the emulation was to achieve a simple validation of the behaviour of implemented motion controllers, and to get visual hints to software error localization.
- Mostly the setup for emulation was done on standard laptop computers, hosting both the emulation in the Blender game engine and the external control system. While lowering the network latency in the communication by a small amount, it played a greater role in lowering the frame-rate of the emulation, and hence the communication frame-rate between the emulated low-level controller and the external motion controller. This was not an intentional deficiency, but rather one induced by the hardware performance in the pertinent development situations. With the right computing hardware and setup, frame-rates up to $200Hz$ frame-rate were no problem to achieve.

As indicated, the emulation was mainly set up for ease of visual inspection and validation of the motion controllers in terms of accelerations, speeds, directions of motion, and axes of rotations; which may be described as behavioural debugging and error finding. To this end, the simple emulation was a tremendous success. For more advanced validations and motion controller development, the emulation model should be extended to encompass the known tracking delay and response time.

Example: AGV Collection Emulation

The development of a prototype AGV was described in the paper included in Section 4.6 [Roulet-Dubonnet et al., 2010]. The same problem of cautiousness with developing a

robot control framework is not present to near the same extend when developing an AGV. The lower power of motors, the lighter mass for the given AGV, and the generally lower accelerations, make the risks of person injury or equipment damage much lower when operating experimentally with the live AGV device. Still, the development of the motion-level control of the AGV would have benefited from a low-level device emulation, but it was not considered at the time of development.

The real-time emulation of a collection of AGVs at their individual motion- and task-level control became a necessity when the development of the AGV system started. The AGV system was described at the analysis and consideration levels by the publication included in Section 4.3 [Roulet-Dubonnet et al., 2009], and with some more design and technical detail in the paper included in Section 4.6 [Roulet-Dubonnet et al., 2010]. The collection of the described mechanisms for matters such as transport order handling, AGV reservation, task scheduling, traffic control, environment mapping, cooperative tasks, etc., that are not entirely attributed to any individual AGV in the collection, is what may be collectively referred to as the AGV system.

The AGV system is hard to develop and experiment with, without having a considerable amount of AGVs. Certain aspects may be tested on just a single operational AGV, but most of the mechanisms associated with the AGV system regards the central theme of managing and coordinating a fleet or collection of AGVs, and it makes no real sense to consider a fleet with a cardinality of one. This is where realistic emulation enters not just as an efficiency enhancer for the realistic development, but downright as a necessity for experimenting with systems that are not yet available.

An AGV collection emulation was easily set up for experimenting during development of the AGV system. Two different AGV emulations were set up for developing AGV systems with two separate purposes.

One AGV system was developed over simplified AGVs with motion-level emulation. By motion-level emulation is meant, that there is no motor-level control and sensing. Thus, direct linear and angular velocities can be commanded to the individual AGVs, and their instantaneous velocities can be read off. The localization system also took an unrealistic advantage of the emulated world, and the full knowledge of the AGV poses in the Blender game engine; and thus the highly problematic aspect of global localization was disregarded. The work with developing this AGV system and the simplified, emulated AGVs for use in the production system emulation is described in the paper included in Section 4.9 [Lind and Skavhaug, 2011]. The AGV system itself was first developed with a stand-alone emulation of the collection of AGVs. The AGV collection appeared as a complex "emulated device" to be controlled by the AGV system being developed. Both the emulated AGV collection and the AGV system was later incorporated into the production emulation system and the production control system described in Section 4.9 [Lind and Skavhaug, 2011]. The simple AGV trajectory control, order handling, and AGV reservation system provided by the AGV system was adequate for the targeted aspects of the production emulation and control system. However, a more realistic AGV emulation

and fully developed AGV system will present a more realistic level of both uncertainty and flexibility to the production control system.

Another, much more realistic, AGV system is under development by Olivier Roulet-Dubonnet. The AGVs are emulated at the motor velocity-control level, with individual wheel velocity sensors; though not physically modelled. One of the objectives for the emulation system is to develop the global localization and tracking system, based on localizing LEDs mounted on the AGVs by vision systems associated with ceiling-mounted cameras. The emulated vision systems are taking advantage of the global knowledge in the Blender game engine of all LEDs in the emulated scene, but provide the controlling AGV agents and the AGV system with realistic data pertaining to the locations of the visible LEDs. Other aspects of development are the overall ordering system, the traffic control system, and the individual AGV motion control systems. This is still work in progress.

For both AGV emulations it is only the lower control levels of the AGVs that are emulated; pertaining to instantaneous velocities and sensor, odometric, and localization data. External to the emulation, each AGV device is controlled by an AGV agent, undertaking trajectory control, low-level ordering, AGV status, inter-AGV communication, and other higher level activities. These AGV agents are representatives of the AGV devices in the AGV system.

3.3 Real-Time Production System Emulation

The ultimate goal of the PhD work underlying this thesis was to investigate and experiment with distributed real-time production control, based on autonomy in the device control entities. The emphasis on PC-based device-control in Section 3.1 is an important step towards deploying tremendous computing power in direct association with the controlled production devices. Virtual experimentation of the real control system on realistic emulated, real-time responsive production devices, as described in Section 3.2, brings about an invaluable tool for developing the device control systems. Before being able to start an efficient development of a real production control system, the production system must be realistically emulated and be real-time responsive. This section is dedicated to describing the advances towards realistic, real-time production system emulation.

3.3.1 Simulation vs. Control of Emulated Devices

In traditional production simulation a distinction can be made between what may be dubbed *production logistics-oriented* simulation and what may be dubbed *device/process-oriented* simulation.

Device/process-oriented simulation covers the detailed operations of a few devices or the

processes in a machine. It is used for either determining or validating the possible abstract control logic for the operation to succeed, or verifying that an operation is withing bounds of restricted resources; such as time, power, geometric clearance, transformation process tolerances, etc. Device/process-oriented simulation bears a high resemblance to the combination of the real-time device emulation and its control system, as described in Section 3.2. However, while the real-time emulation and control setup respects the real-time nature of the interaction and aims at using and developing the *real* control system for the devices, the device/process-oriented simulation concerns itself neither with any relation to the real time clock nor with the concrete, deployable, detailed, realistic logic control of the devices. Being constrained by real-time requirements, any emulation may only obtain a certain quality of process realism. Determining high precision internal stress-development during welding, forging, casting, etc. are examples of process-oriented simulations which may not be expected to be obtainable in real-time emulation.

Production logistics-oriented simulation takes as fundamental data the processing times of devices and machines, transport times, waiting times, buffer sizes, changeover times, etc., and possibly the statistical characteristics of all of these. With the use of DES, as the typical method, a model of the overall internal ordering is set up for the production system containing all the devices, machines, buffers, transport units, etc. By running the DES, a multitude of statistical characteristics of the performance of the production control system as a whole may be immediately derived. With careful analysis, tuning, and rerunning the DES, the abstract logistical control-logic can be developed and enhanced. Furthermore, introspective analysis of the statistical data collected from a DES-run will also reveal the locations of bottlenecks, giving hints to where effort should be directed in increasing productivity or where production capacity should be increased by additional production equipment. A good survey of DES for production systems is given by Smith [2003].

The production logistics-oriented DES, has its abstraction level of device and transformational and logistical process description primarily based on cycle-times characteristics. It leaves no room for a complex, real-time interaction between higher level production logistics control and the detailed individual device or process control. The separation of the device and logistics control perspectives, on which production logistics-oriented simulation relies, stems from a traditional view on reliability, repeatability, repetitiveness, and simplicity in device and machine operation in manufacturing. Many of the problems arising from demands for shorter batches and quicker changeovers in production today may be conjectured to origin from the remnant of this traditional principle of disconnecting production logistics and device control.

If the future of production control brings highly autonomous control systems, which are cognitive in a broad sense and distributed over the production devices and machines, the simple divisions of computation and analysis underlying production logistics- and device/process-simulation breaks down. The focus must be switched towards realistic real-time emulation of the entire collection of production devices, and executing the *real*

production control system in the virtual world created by the production emulation. This is largely what has been proposed and described in the papers included in Section 4.7 [Lind and Roulet-Dubonnet, 2010] and Section 4.9 [Lind and Skavhaug, 2011]. Support for such arguments may be found in the literature of agent-based production control; confer e.g. Vrba and Marík [2005].

3.3.2 From Device to System Emulation

In exact analogy with the division of the device control systems and the emulated, controlled devices presented in Section 3.2, the emulated production system is cleanly separated from the production control system. The full production control system may be a distributed system of control entities at very different levels, or it may be a highly centralized, near-sequential program running in a single process and with a very limited number of threads of control. There are two requirement of paramount importance for the emulation system to be realistic:

(Interfaces) that it provides realistic, real-time responsive, accessible control interfaces to all devices involved in the production system;

(Constraints) and that it implements and enforces all the lower-level constraints and interactions among the devices, between devices and uncontrolled objects, and among uncontrolled objects.

The requirements on interfaces is natural and obvious. It is in principle what is provided by a simple aggregation of the independently developed device emulators as per the description in Section 3.2. The requirement that the interfaces be realistic is fundamental, and should be simply carried over from the development of individual device emulation and modelling. For the interaction between the production control system and the production system emulation to be real-time responsive to an adequate level, the implementations for the individual devices may need simplifications or distribution of functionality; as explained in Sections 3.3.3 and 3.3.4. The last requirement associated with interfaces regards accessibility. The production system emulation must set up for all devices to be accessible over their control interfaces such that they do not clash on underlying communication interfaces. For instance, using serial communication ports to access some type of emulated device in the production system emulation, the setup routines must ensure that each device gets a dedicated serial port. Similar considerations applies to addresses, ports, channels, etc., of any other communication hardware and technology; e.g. USB, CAN, FireWire, and Ethernet.

The requirement on constraints is much more subtle and complex, since it regards not only the low-level device interactions, but also the natural, physical behaviour of objects that are not under explicit control; such as workpieces. The implications on developing a realistic production system emulation from such emulation-internal activity is described at great length in the papers included in Section 4.9 [Lind and Skavhaug, 2011, Section 5.1]

and Section 4.7 [Lind and Roulet-Dubonnet, 2010, Section 3.1].

A further aspect that enters when emulating a production system, which is not present in device emulation to the same degree, is the interaction with external systems; such as process-control and factory-wide information services; e.g. ERP systems. Though much of this interaction is expected to be addressed by the production control system, rather than the emulated production system, some external systems may be in need of addressing some level of device control or behaviour, which is implemented in the emulation system.

In an emulation system based on the Blender game engine it may be advantageous, for this reason and, as indicated in Section 3.3.3, for performance reasons, to move as much complexity and functionality out of the game engine platform as possible. The functionality will remain emulation specific, and the computation entities that contain it will as such still be part of the emulation system. However, the surrounding run-time presented to the emulation components external to the game engine may be easier to integrate with such external systems as process-control and factory information services. This principle is not to be considered specific to the Blender game engine, but rather a general principle of re-deploying functionality away from centralized, monolithic-core systems.

3.3.3 Performance Considerations

Performance of an implemented system emulation in the Blender game engine is indicated by the achieved frame-rate, or its inverse, the frame cycle-time, of the running game engine. For any production emulation, there will be a lower bound on the frame rate, below which a certain device or a certain transformational or logistical process will become unreal, un-natural, or simply fail. An arc-welding process, for instance, having a trajectory position-tolerance of $1mm$, executed by a robot at the speed of $10\frac{mm}{s}$ may meet insurmountable control difficulties when the frame-rate drops towards $10Hz$. On the other hand, many production applications with no timely related tolerances will operate quite well even below this frame-rate.

As indicated in Section 3.3.2, the Blender game engine is not a distributed platform itself, but rather centralized and monolithic. This sets a limit to the size of the emulated system in terms of number and complexity of devices contained, with no possibility to achieve good scalability by complete separation and distribution. The limitations may arise from any kind of resource limit on the computer hosting the Blender game engine executing the system emulation. It is a consistent observation throughout the PhD work that it is the computational power of the host computer that is always the limiting resource for the performance. In different system setups than the ones that were experimented with it might be the memory, the graphics card, or the network bandwidth or latency which will be the limiting resource.

As an increasing number of production devices are added to an emulation scene to make up a production system, the nature and applicability of the emulation changes. The emphasis switches from physical realistic emulation, being the case for individual device emulation, towards a simplified implementation limited to the sufficiently realistic features of behaviour; in order to retain real-time responsiveness.

Performance of a system emulation is dominated by the fact that the emulation frame cycle-time has an approximately linearly increasing component with the number of any given independent device; as indicated in the paper included in Section 4.9 [Lind and Skavhaug, 2011, Section 6]. If, in addition, devices may have direct interactions with a number of other emulated devices in proportion to the total number of such devices, this will contribute a super-linear growth in frame cycle-time, and thus a considerably rapid decrease in performance of the system emulation with the number of devices.

The observation that it is always the computational power of the computer hosting the emulation game engine spurs a principle to retain only the core functionality of the emulated devices inside the game engine. For some devices this is the natural thing to do. For other devices it becomes a violation of the principle that the real production device is entirely embedded in the emulation system in the game engine, while the production control system provides the external control of the device. Fortunately it is not difficult to keep the principle of a clean separation while gaining some level of distribution of the otherwise monolithic emulation system in the game engine. It is a matter of re-deploying a certain amount of emulation control code at a computation node remote from the game engine node, and establishing an interface to the remaining control code inside the emulation system in the game engine. This separation must remain transparent to the production control system, however, since otherwise it will become emulation specific.

3.3.4 Distribution of Emulation Functionality

An example which provides several natural options for the division of both the external control to device emulation interface as well as the division of the device emulation between an external computation node and the game engine is presented by a robot with the control system based on PyMoCo; confer Section 3.2.3 and the paper included in Section 4.10 [Lind and Schrimpf, 2011].

At the top level, there is the option of presenting the production control system to a fixed control logic; the typical principle of integration supported by standard industrial robot controllers. Alternatively the emulated robot presents a prepared set of motion controllers, which may be logically controlled from the production control system. The ultimate alternative is to present the production control system to the low-level, real-time interface of controlling the robot joints directly; yielding the maximum of flexibility by integrating a motion control functionality directly with the production control system.

The division and distribution of the emulation system over external computational nodes

and the game engine node also presents several alternatives. The concern of the lowest level, which must be deployed inside the game engine, is the controlled positioning in each time cycle of the geometry parts of the pertinent device; i.e. links, tools, sensors, etc., for a robot. The minimum computational load on the game engine is achieved by exposing an interface to the internal emulation entity associated with the device, over which is provided pose data for all geometry parts in each time cycle from the external emulation entity. The external emulation entity for the device may then provide the required interface of the emulated device to the production control system. In the case of a robot, the external emulation entity must perform the forward kinematics computation in each time cycle, resulting in sending the poses of all robot parts to the internal emulation entity. Additionally, depending on the interface required by the production control system, the external emulation entity may have to undertake the motion control step required by the task, and possibly even the logical task handling.

Alternatively, the computational load of the full forward kinematics for finding robot part poses may be left inside the game engine, while leaving task logic and motion control with the external emulation entity. For a robot device, this latter approach results in a much cleaner and generic interface of data exchange, in the form of joint positions, between the external emulation entity and the internal game engine part of the device emulation. Finally, there is the option of implementing the entire emulation of the device inside the game engine. This design choice leaves the full computational load on the game engine node, but simplifies the software and computational system setup by eliminating the external emulation entity.

Development, in reality, of any advanced production system and production control system will experience a very similar phase of determining boundaries and interfaces among deployed subsystems. Hence, besides being not particularly specific to emulation of production systems based on the Blender game engine, this exercise is not even to be considered specific to the setting up of an emulation system.

Similar considerations of division of functionality between the game engine and external emulation was touched upon for the emulated AGVs described in Section 3.2.3.

3.4 Experimental Production Control System

Throughout the PhD work of this thesis, the IntelliFeed project has been concerned with a specific prototype production setup, and the development of distributed, autonomous control for that setup; confer Fig. 3.1 for an overview of the scene as modelled in Blender. The papers included in Section 4.2 [Lind et al., 2009], Section 4.3 [Roulet-Dubonnet et al., 2009], Section 4.7 [Lind and Roulet-Dubonnet, 2010], Section 4.8 [Lind and Roulet-Dubonnet, 2011], and Section 4.9 [Lind and Skavhaug, 2011] dedicates a considerable amount of explanation of the production and manufacturing related circumstances; the devices and logistical processes involved with their individual control; and the production

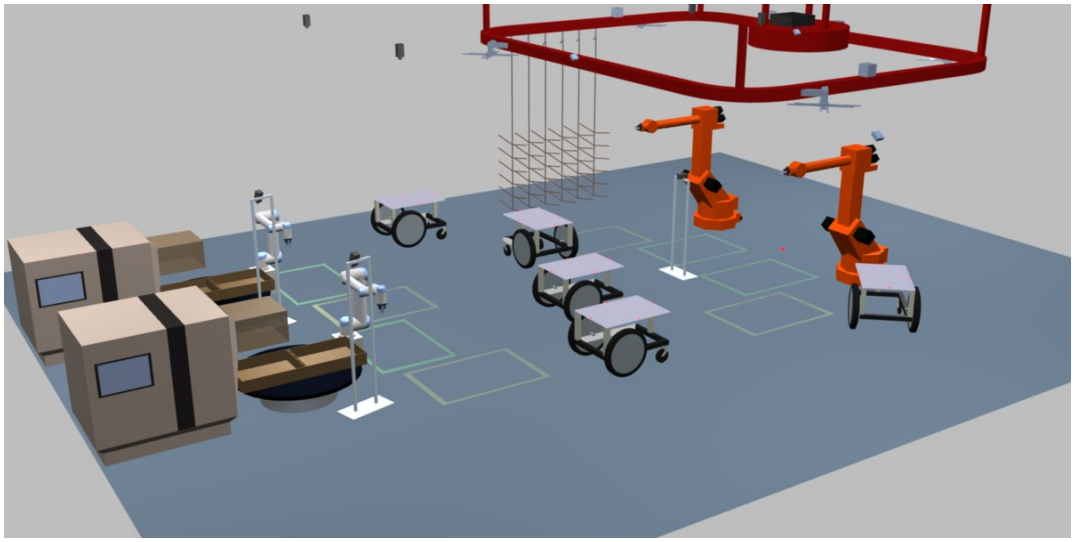


Figure 3.1: *Overview-snapshot from the emulated prototype production system modelled in Blender. Supply cells are seen on the left, each comprising a CNC machine for producing workpieces; a top-view camera for bin-picking, a picker-robot, of type Universal Robots UR-6-85-5-A, associated with a bin-picking vision system; and a turntable with boxes for switching produced workpieces from the CNC machine to the picker-robot. Upload cells are seen on the right, under the PnF-carrier, each featuring a Nachi SC15F robot and a camera and vision system for picking workpieces off a docked AGV. The carrier management is based on the side-view camera under the PnF-conveyor, focused on the upload position of a carrier between the upload-robots. The PnF-conveyor comprises the carrier trolleys, on which the paint-system carriers are suspended, and the controllable PnF-stops located on the track of the PnF-conveyor. The paint-system carriers are seen as skeletal structures at the far end of the PnF-conveyor. A total of five AGVs are seen around the scene on the floor and on the floor near the cells are shown some marked rectangles, symbolizing docking poses for the AGVs.*

scenario to be supported by a production control system.

For testing purposes, a simple production control system was developed along with the production system setup and emulation system. The developed production control system was designed to be centralized, for keeping setup, testing, experimentation, and run-time inspection comprehensible; these being the initial goals for the experimental production control system. It was thus experimental in the sense that it was used to perform experiments with the emulated production system. However, over time it evolved to some level of sophistication and will be easily distributed in its current state. The level of autonomy is not impressive, though, since it has remained a central goal for the control system to be a facility for testing and debugging the emulation system.

The sophistication level evolved naturally from the attempt at making the emulation system realistic. Thus, like in a real production control system, there are many details with many timely constraints and a high degree of concurrency. Though quite sophisticated in its present state, there is no current support for orchestrating changeovers. To

demonstrate changeovers, first of all the production system must be slightly extended to handle different workpieces and use different tools and carriers. Since there is currently only one set of tools, carriers and workpieces in the emulated production system, the production control system handles only that specific setup.

It is the purpose of this section to give an impression of the design, implementation, and operational logic of the experimental production control system; which has been almost neglected in the publications. The description is based on UML diagrams as a descriptive tool, and reference to included interface and implementation code in Appendix A.

3.4.1 Overview

Fig. 3.2 gives a structural overview of the central production controllers, and their association with important emulated devices. The diagram is a UML object diagram, and thus illustrates the objects and their associations at a given instant of time. In the illustrated prototype system in Fig. 3.2, the only associations that change over time are those regarding the AGVs.

In addition to the displayed emulated devices, but not shown, there is a collection of dock-devices for the AGVs. These play no major role for the overall workings of the system, and have been left out of the diagram for reasons of clarity.

There are four distinctly identifiable clusters in the control system:

- Supply of workpieces.
- Upload of workpieces.
- Management and execution of transportation.
- Management of upload-carrier and control of PnF-conveyor.

The responsibilities and operations of each of these clusters are the target for descriptions in the remainder of this section.

3.4.2 Supply Operations

An object of the SupplyCell class is the hub of a workpiece-supply cell. There are two workpiece-supply cells in the laboratory prototype system, so two SupplyCell objects are found in Fig. 3.2; lower left and lower right. Appendix A.2.1 includes the code implementing the SupplyCell class.

A SupplyCell object associates and orchestrates a workpiece producer, a bin-picking vision system, a tool-linear controller for a robot with picking tool, and a turntable. It is further responsible for acquiring AGVs to fill with workpieces. It has an associated picker object of class Picker, which is itself associated with the vision system and the robot

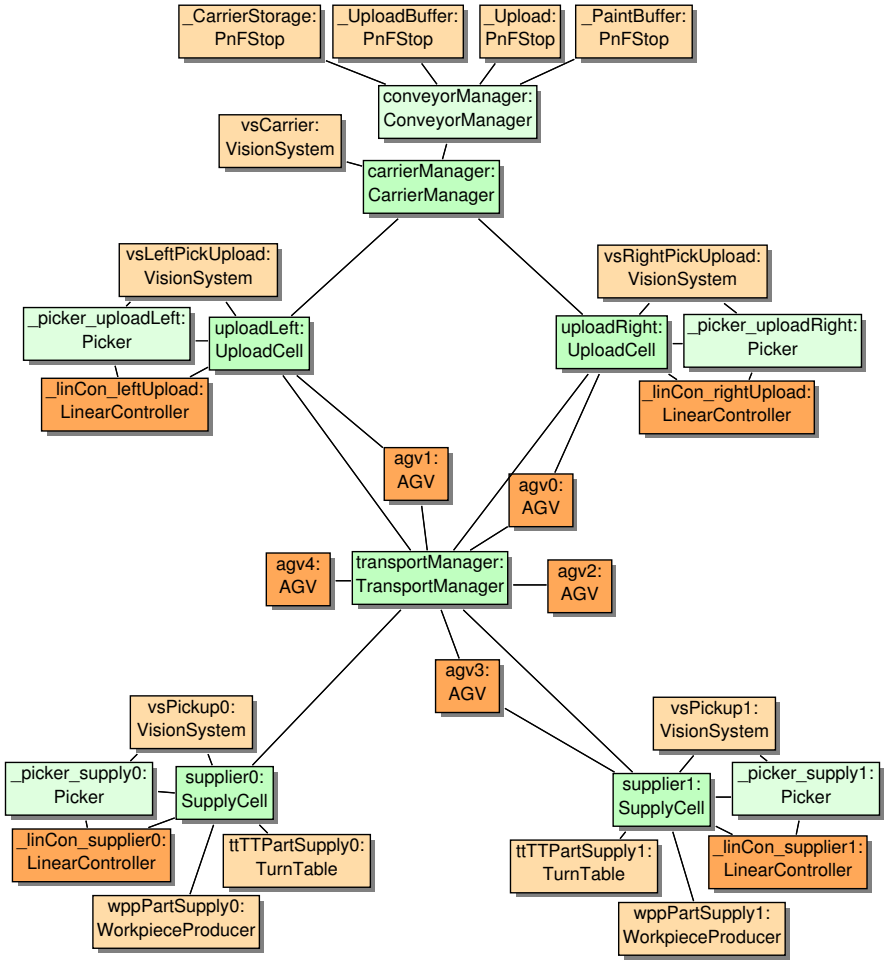


Figure 3.2: *UML object diagram for the most active and important objects from the production control and production emulation systems in the prototype production setup. Production system controllers are shown in shades of green and interfaces to emulated devices are shown in shades of orange. Important controllers from the production system are shown in strong green, while more peripheral controllers are shown in pale green. Distributed controllers for emulated devices are shown in strong orange, while controllers deployed purely in the game engine are shown in pale orange.*

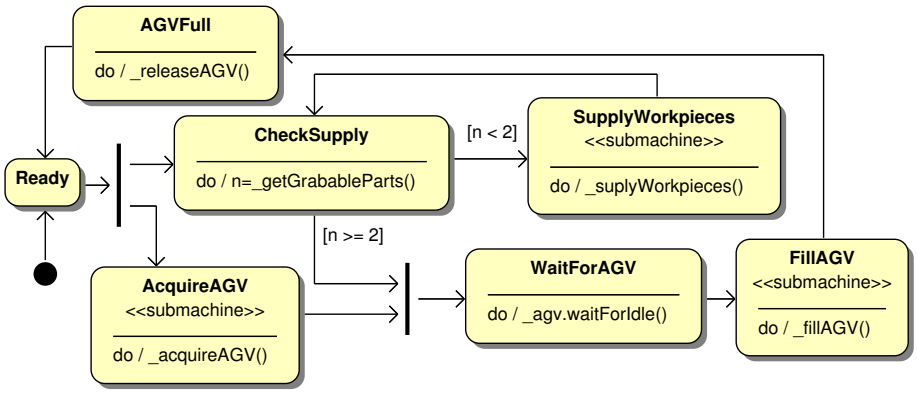


Figure 3.3: *Overview of the operations of a supply cell modelled as a UML state machine diagram. The main cycle is based on acquiring an empty AGV, and filling it with workpieces. The main operations are in the sub-machine for filling workpieces on the AGV. Re-supply of workpieces from a workpiece producer is represented by a separate sub-machine, which is invoked while waiting for an AGV to arrive, and as an integral part of filling the AGV.*

controller of the supply cell. The picker is delegated the general operation of selecting and picking a workpiece when ordered. Both the PICKER and the SUPPLYCELL objects are associated with the tool controller for the tool mounted on the robot. The tool controller is, for reasons of clarity, considered as an integral part of the robot controller in Fig. 3.2. The operation of a PICKER object will not be described in detail. However, since it plays an important role for both the SUPPLYCELL and UPLOADCELL classes, as evident from the descriptions in the following and from Fig. 3.2, the code for the PICKER class is included in Appendix A.2.3 for reference.

The overall operation of a SUPPLYCELL object is shown by a UML state machine diagram in Fig. 3.3. Note that the UML state machine formalism allows sub-machines; confer e.g. the *OMG Unified Model Language™ (OMG UML), Superstructure* [OMG, 2010]. The purpose of the supply cell is to fill a docked AGV with workpieces; that the AGV may then supply to any production facility in need of the parts. The supply cell does not operate to order, but simply fill AGVs as fast as possible; limited by availability of unloaded AGVs, workpiece production rates, and pick-and-place operation times.

A necessary precondition to filling an AGV is the reservation and docking of an empty AGV at the loading dock of the supply cell. In parallel with the waiting for arrival of a reserved AGV, the supply cell checks that there are indeed available parts in the box on the turntable facing the picker-robot. While the number of pickable parts are below some threshold, a resupply-operation is ordered, comprising an order to the CNC-machine to produce a certain amount of parts. At the join of transitions in Fig. 3.3 an AGV has arrived, and a minimum number of pickable workpieces is guaranteed.

There are three sub-machines involved in the overall operation state machine of the supply cell in Fig. 3.3. Each will be explained in the following.

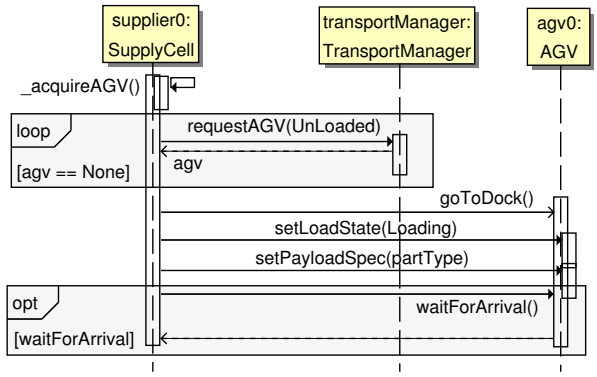


Figure 3.4: *UML sequence diagram for the interaction between a supply cell, the transport manager, and an AGV, for obtaining a reservation for an AGV, and commanding it to the dock. An infinite loop is initially entered, wherein each iteration requests an unloaded AGV from the transport manager. The loop runs as long as no AGV object is returned. As a reserved AGV is returned, it is immediately sent to the loading dock of the supply cell.*

Acquisition of an AGV

In the supply-cell state machine, Fig. 3.3, the acquisition of an AGV is a separate sub-machine; `AcquireAGV`. The operation of acquiring an AGV is an interaction between a supply cell (or an upload cell; confer Section 3.4.3), the transport manager, and an AGV. The interaction is best described by a UML sequence diagram, as shown in Fig. 3.4.

The scenario of acquiring an AGV may be recognized in the protected method *_acquireAGV()* of the SupplyCell class in Appendix A.2.1. The transport manager is repeatedly asked for an available AGV until one is returned. When a reserved AGV is returned it is guaranteed to match the requested load state and payload type. In case of a supply cell request, the AGV is requested to be un-loaded and with un-specified payload type; the payload type being irrelevant since the AGV is requested to be un-loaded. In the equivalent operation of acquiring an AGV for an upload cell, the AGV is requested to be loaded, and match a given payload specification for the pending upload process. The AGV returned from the transport manager is for an AGV which has been reserved to the requesting cell, and it may be immediately commanded to the cell's operational dock.

As reservation of the AGV is obtained, and it is commanded to the operational dock, its load state and payload specification is set according to the planned operation. For the supply cell the load state is set to `Loading` and for the upload cell the load state is, in the corresponding situation, set to `UnLoading`.

Optionally, though not used in the current implementation, the caller to the *_acquireAGV()* method may be blocking to synchronize with the arrival of the reserved AGV to the cell's dock, as seen in the lower fragment of Fig. 3.4.

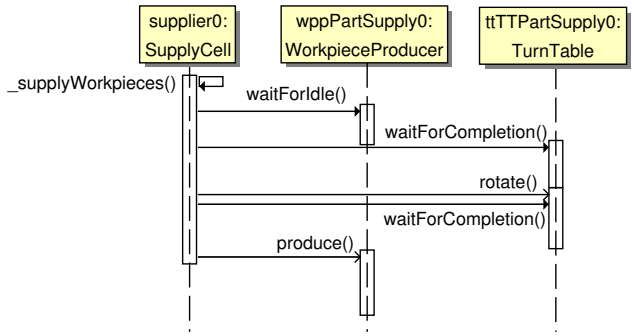


Figure 3.5: *UML sequence diagram for the operation of re-supply of parts to pick. First any previously ordered, ongoing operations of the turntable and CNC machine resources are allowed to complete. Then the turntable is rotated, and after completed rotation, a box filled by the previous produce operation is facing the picker-robot, while a new produce order is allowed to commence asynchronously.*

Re-supplying Workpieces for Picking

Re-supplying workpieces for picking by the associated picker is an operation involving the workpiece producer (the CNC machine), the turntable and the bin-picking vision system. The supply operation always leaves the box under the CNC machine being filled with workpieces, while immediately rotating a previously filled box on the turntable towards the picking robot. Thus, the actual producing of new workpieces proceeds in parallel with the main operation of the supply cell.

The operation of re-supplying workpieces for picking was shown as a sub-machine in the state machine of the overall operation of the supply cell in Fig. 3.3. Fig. 3.5 shows the details of the re-supply operation by a sequence diagram for the `supplier0` supply cell. The corresponding method in the implementation of the SupplyCell class in Appendix A.2.1 is found in the protected method *_supplyWorkpieces()*.

Filling an AGV

The filling of an AGV, at rest in the operational dock of the supply cell, is the operation of the supply cell fulfilling its ultimate purpose. The operation is illustrated by a UML state machine diagram in Fig. 3.6. The operation is implemented by the code for the protected method *_fillAGV()* in the code of the SupplyCell class in Appendix A.2.1.

The main operation consists of interleaved operations of picking a workpiece from a supply box, deploying the picker associated with the supply cell, and dropping it at a layout-site on the AGV. The picker, when requested to perform a pick of a workpiece will indicate failure if no pickable parts could be found in the supply box, and this results in a re-supply operation. After re-supplying, the picking and dropping operations are resumed, proceeding thus until the AGV has been filled to the specified layout.

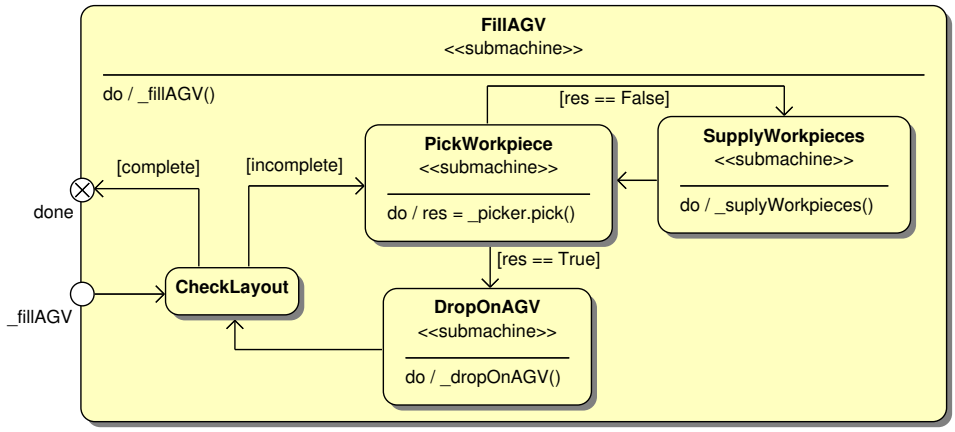


Figure 3.6: *UML state machine diagram for illustrating the operations involved in filling an AGV. The main iterative cycle between the states* `PickWorkpiece` *and* `DropOnAGV` *proceeds as many times as there are workpiece sites in the layout on the AGV; each cycle filling one site. The pick operation in the* `PickWorkpiece` *sub-machine may fail, when no workpieces are found by the bin-picking vision system of the supply cell, triggering a re-supply operation.*

In the current implementation the supply cell contains directly the layout of the workpieces to put on the AGV and the logic for expressing the layout by place-operations with the robot controller. In a real, advanced production system, the specifications of layouts and motion involved with adding workpieces to an AGV may be of a great variety. Such specifications would typically reside in an off-line generated database, but they may also, in very advanced production systems with highly customized products, be generated on-line, and thus unknown prior to the operation. The supply cell operation should be flexible with respect to layouts and motion specifications for adding workpieces, and this aspect could be adequately based on the strategy software design pattern; as described by Gamma et al. [1995]. This current lack of flexibility also pertains to the upload process of the upload cell, and with the general picker class for picking workpieces identified by a vision system. All these aspects must have their motion and identification computations and controls encapsulated with a strategy support-object for given workpiece types and operation conditions.

3.4.3 Upload Operation

An upload cell is operationally and logically very similar to a supply cell. Two UploadCell objects are found in the system overview in Fig. 3.2, named `uploadLeft` and `uploadRight`; to the upper left and upper right. An upload cell is persistently associated with a robot linear controller, a bin-picking vision system, a workpiece picker, the transport manager, and the carrier manager. An upload cell is further transiently, periodically associated with an AGV which is acquired from the transport manager. The AGV brings workpieces from the supply cells for upload on a paint system carrier, and thus represents an implicit

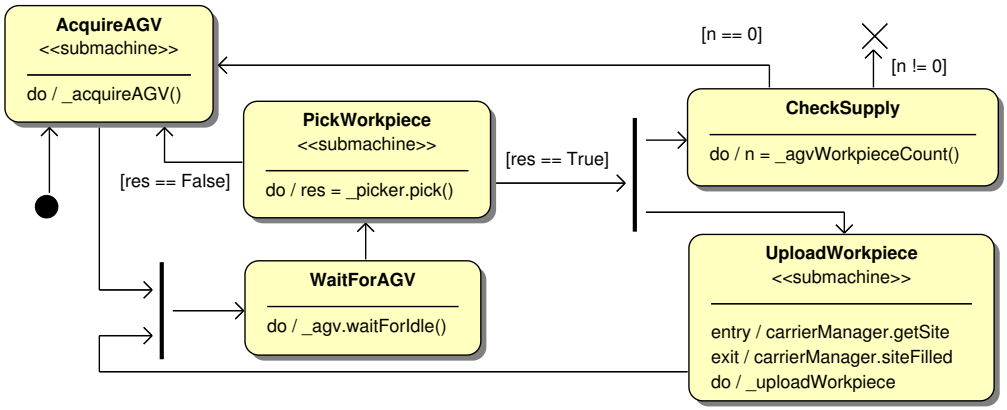


Figure 3.7: *UML state machine diagram for the upload cell operation. The basic operation cycle is to pick a workpiece from the AGV by using the vision system, and uploading the workpiece to a site on the paint system carrier. Before a pick is performed, it is ensured that an AGV is docked. If a pick operation fails, the AGV is deemed empty, and a new AGV is requested. Following every pick, the AGV is checked for workpieces, and if none are found, a new AGV is requested. The upload operation proceeds concurrently with this check. The upload operation may block on requesting a new site on the paint system carrier, and remain blocked until a new carrier is in place.*

association with the supply cells.

The objective for the upload cell is to fill its share of a paint system carrier with workpieces for painting, at the highest possible rate. Compared to a SupplyCell object, an UploadCell object have less devices to orchestrate while another production system controller to interact with. The interaction with another, high-level, production system controller is, in this case, less interaction-complex than that of commanding a device controller. Of course, generally there are easily imaginable, highly complex interactions between higher-level controllers in an advanced production control system; the complexity of which may vastly supersedes that of mere device control.

The operation of an upload cell is illustrated by a UML state machine diagram in Fig. 3.7. The code for the UploadCell implementation is included in Appendix A.2.2. The sub-machines `AcquireAGV` and `PickWorkpiece` of the UML state machine diagram in Fig. 3.7 are identical in operation to the correspondingly named ones shown for the supply cell operation in Fig. 3.3. The most pronounced difference in a real system lies with the vision system. In the supply cell, workpieces are produced and fall freely into boxes, and hence the bin-picking vision system can not assume anything about the structure of workpieces to identify. The vision system of the upload cell may assume a structured, specified layout of the workpieces on the AGV supplying them. This difference is specific to the setup in the prototype production system.

In the implementation for the upload cell, in Appendix A.2.2, the protected method *_acquireAGV()* implements the `AcquireAGV` sub-machine. The `PickWorkpiece` sub-machine is implemented by a call to the method *pick()* of the Picker object associated with the

pertinent UploadCell object.

The main operation cycle for the upload cell is to let the associated picker pick up a workpiece from the AGV, acquire a free site on the paint system carrier from the carrier manager, and command the robot to place the picked workpiece on the acquired site on the carrier. This operation cycle flow is interrupted in two situation:

1. When there are no more workpieces on the docked AGV.
2. When there are no further upload sites on the carrier.

The handling of these two situations in the upload cell are of two quite different natures. The acquisition of a new AGV in the `AcquireAGV` sub-machine is, like for the supply cell, directly handled by the upload cell. I.e. the transport manager is repeatedly requested for a loaded, available AGV, and the AGV, which eventually gets reserved, is commanded to the operational dock of the upload cell.

The acquisition of a new carrier is not observed from the state machine for the upload cell. It is not even expressed the code in Appendix A.2.2, since it is handled integrated with the call to the method *getSite()* of the the carrier manager, `_carMan`, in the `UploadWorkpiece` sub-machine.

The operation that clearly distinguishes the upload cell from the supply cell is the upload of a workpiece to the carrier. It is symbolically illustrated in Fig. 3.7 by the `UploadWorkpiece` sub-machine. The `UploadWorkpiece` sub-machine is implemented by the protected member *_uploadWorkpiece()* of the UploadCell class. Its operation is simply that of acquiring a free site for upload on the carrier from the carrier manager, computing the approach position, and commanding the robot linear controller to the approach and to the attachment pose of the site, attaching the workpiece, and registering with the carrier manager that the site has been filled. The logic for computing and executing the actual upload is hard-coded into the *_uploadWorkpiece()* member function. This exposes the lack of support for different workpieces and carriers, and in a flexible, more realistic system will be implemented by a strategy software design pattern. Both the pick and the upload motion processes are depending on specific workpiece, tool and carrier types, and should be designed as dynamic strategies.

3.4.4 Transport System

The transport system is the coupling between the supplier and upload cells. It partially administrates the AGVs, which perform the physical transport. The TransportManager is a singleton class in the current design, and the object, `transportManager`, of the experiment control system is seen at the centre of Fig. 3.2, with the AGV objects scattered around it. A third central concept of the transport system is represented by the Dock class; which was left out of Fig. 3.2 for clarity. The Dock objects serve as registered

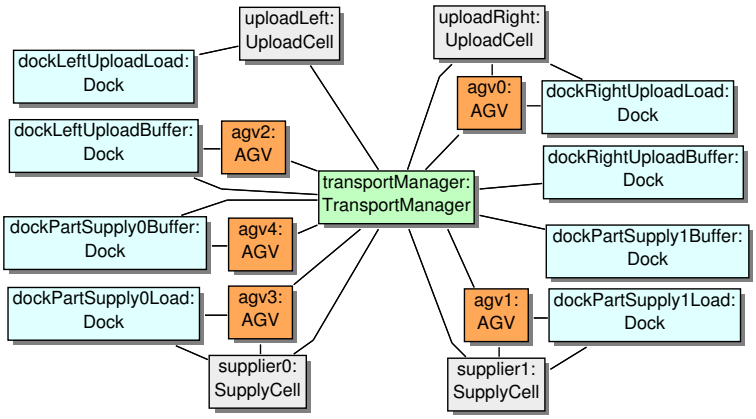


Figure 3.8: *UML object diagram for an overview of objects and associations involved with the transport system. The singleton* TransportManager *object is central, and connected to almost any other entity of the transport system; shown in green as a production system controller. The* AGV *objects represent the aggregation of the low level devices for an AGV, such as velocity controller and localizer system, with the production system aspects of handling individual transport tasks; shown in orange as distributed emulated devices. The* Dock *objects represent the information service connected to specific physical areas, where AGVs must have reservation to pose themselves; shown in light blue symbolizing production system information services. For completeness, the* SupplyCell *and* UploadCell *objects are included since they are clients to the transport system; they are here shown in grey since they are not a part of the transport system.*

targets and restricted zones for AGVs in the transport system, and the docks of the prototype production system are included in the transport system overview in Fig. 3.8.

Fig. 3.8 illustrates the associations among objects that make up the transport system and its clients in the production control system. The associations between AGVs and supply cells, upload cells, and docks are transient, while the all remaining associations are persistent. Generally the operational docks are associated with the operational cells, while the buffer docks are associated with the transport manager. The AGVs are taken under appropriate ordering of the transport manager when they are idle, whereas they are commanded by the supply and upload cells when under operation. The geometric layout of the objects in Fig. 3.8 reflects the geometric layout of the associated physical entities on the shop-floor in the laboratory prototype.

Docks

A primitive Dock object is an informational service for handling the reservation of a particular area around a given position and orientation on the shop-floor. It is part of the logic of all AGVs in the production control system to not enter a target dock without having obtained reservation first, and when entering it must register this with the dock service object. In a more advanced system, with, for instance, mechanical fixing or local sensor guidance of a docking AGV, the Dock class may be much more involved, and may

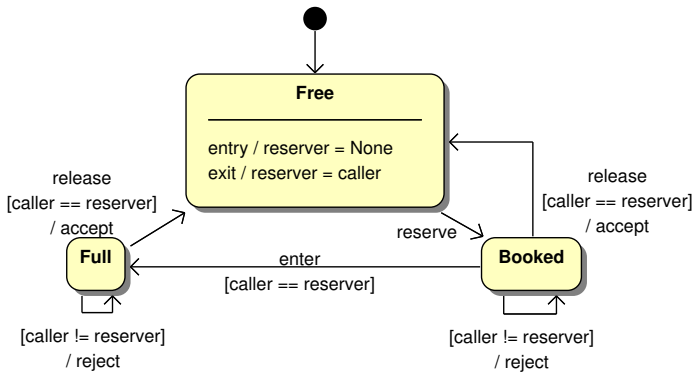


Figure 3.9: *UML state machine diagram for a* Dock *object. The standard operation cycle is seen as the central transitions from* `Free` *to* `Booked` *to* `Full` *and back to* `Free` *state. Whenever in the* `Booked` *or* `Full` *states the* Dock *object has a reserver, and only the reserver is allowed to do operations that change the state.*

even temporarily take over the direct control of some low-level AGV devices.

In an advanced AGV system for production system transport tasks, the individual AGVs will ensure general collision avoidance among themselves, while also avoiding immediate dead-locks. This may be based on central localization services, inter-AGV communication, and, for safety and stability reasons, proximity sensors on the individual AGVs. Certain situations or conditions, however, may render such autonomous and cooperative mechanisms inefficient. An example of this is the case of an operational dock, where the AGV under operation must not suddenly, autonomously, decide to yield to another, passing, AGV. While standing still, or under very controlled motion, under operation with a production system controller, the AGV must not be hindered in its operation. Two other examples where autonomous AGV-behaviour should be suppressed for reasons of efficiency and safety are high-speed corridors and congested intersections. A separate or integrated, distributed traffic control system may take responsibility for controlling or monitoring such features of the transport system. These features are control-wise reminiscent of the primitive dock, in the respect that AGVs, based on certain conditions and priorities, must achieve some level of reservation, and once obtained they may lower their internal level of safety awareness, allowing for more liberty in motion planning and control.

The primitive Dock class used in the prototype system in this work is, in its nature, a representative of such traffic-control features. The code for the interface provided by a Dock object, the *Dock* interface, is included in Appendix A.1.5. The code for implementing the Dock class is, as is evident from a quick glance at the interface, quite simple and not included in the appendices.

The operation logic of a dock is illustrated by the UML state machine diagram in Fig. 3.9. It shows that a dock in the `Free` state will acknowledge any AGV-request for reservation, registering the AGV as the reserver of the dock. Reservation implies a transition to

the `Booked` state, whereby any subsequent requests not originating from the reserver are rejected; likewise in the `Full` state. The reserver is obliged to notify the dock at the time of arrival to the dock, triggering a transition to the `Full` state. From either of the states `Full` and `Booked`, the AGV may trigger the dock to make a transition to the `Free` state. From the `Full` state, this symbolizes that the AGV has left the dock, whereas from the `Booked` state it is equivalent to a cancellation of a reservation.

More complex logic may be implemented in the DOCK classes. For instance, an AGV may notify that it leaves the dock but retains its reservation. Another realistic example of more advanced logic is cooperative docking, where two or more AGVs may bring together various, simultaneously necessary parts for assembly. The cooperating AGVs must then be given coordinated priority for obtaining simultaneous reservation of the neighbouring, operational docks for the assembly cell.

Transport Manager

The transport manager is a central element, serving two main operational purposes:

- On request from a production control entity, all AGVs are searched for an appropriately loaded, unreserved AGV according to the request. This is a reactive task in response to requests.
- Periodically searching for idle AGVs, and sending them to buffer docks flagged to be appropriate for the load state of the pertinent AGVs. This is an active, internal task of the transport manager.

In addition, the transport manager has an internal task of regularly refreshing its list of active AGVs in the production system. The mechanism employed in the current implementation is suitable for adding AGVs, but there is no mechanism for handling AGVs which disappear from the system. I.e., the transport system may handle dynamically deployed AGVs but not AGVs being dismissed or disconnected from the system. This is not a severe issue for testing in the scenarios of the prototype production system, which simply deploys a fixed set of AGVs initially, an no AGV is ever dynamically deployed, disconnected, or dismissed during operation.

The code for implementing the TRANSPORTMANAGER is included in Appendix A.2.4 and the interface provided to the production control system, the *TransportManager* interface, is found in Appendix A.1.5.

The core service of the transport manager is to support the request for reservation of an AGV to any production control system entity. The interactions that unfold when the method *requestAGV()* of the TRANSPORTMANAGER class is invoked is illustrated in the UML sequence diagram in Fig. 3.10. The illustrated scenarios are taken from an invocation from the `uploadLeft` upload cell, requesting an AGV loaded with workpieces for upload to the paint system carrier.

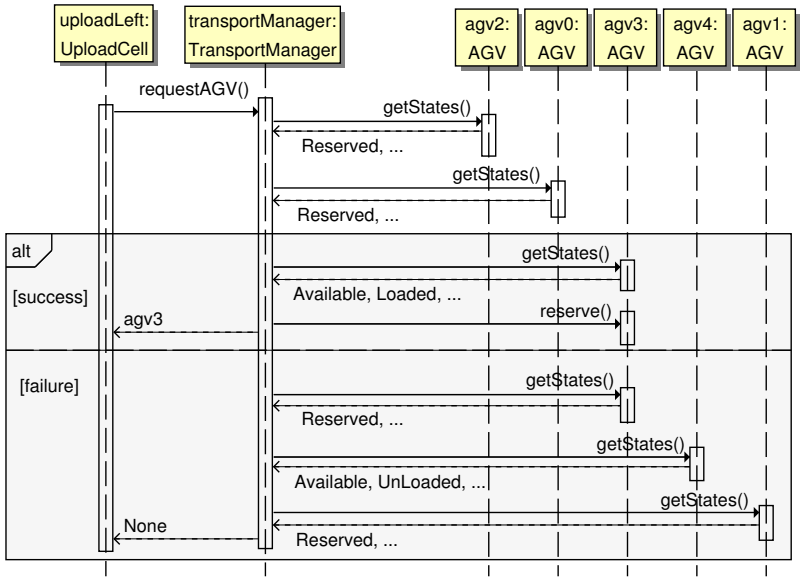


Figure 3.10: *UML sequence diagram for illustrating the interactions of the transport manager with the AGVs, when an operational cell requests an AGV. In the specific scenarios illustrated, an upload cell is requesting a loaded AGV. The transport manager queries all active AGVs in turn for their states regarding reservation and load. For the request from the upload cell, a match is made when an available, loaded AGV is located, the AGV is reserved to the upload cell and an interface to the AGV is returned; as illustrated in the "success" alternative. The "failure" alternative illustrates that if all AGVs have been queried and no state-match is found, the transport returns an empty, "None"-interface to the requester.*

The diagram illustrates two alternative scenarios; one where an appropriate AGV is successfully identified and one where all AGVs are queried but without identifying an appropriate AGV for the request. A sweep is started for querying all AGVs, and it stops when either an AGV is identified which matches the request or if all known AGVs have been queried and failed to match the request. In case of success, the matching AGV is reserved to the invoking production system controller, and an interface to the reserved AGV is returned. In case of failure to find an appropriate AGV, an empty interface is returned.

The internal, active operational purpose of the transport manager, i.e. the continual identification and appropriate disposal of idle, available AGVs, is a production control task. It is a centralization of functionality for the production control system, relieving AGVs and operational cells that use the AGVs from the complex production knowledge about where to send AGVs after operation. In the prototype system, as an AGV has been filled with workpieces at a supply cell, some entity must identify the situation and know where to send the loaded AGV. A similar situation arises at an upload cell, when a docked AGV has been depleted of workpieces. For the supply cell it is a natural part of its operation to request for a new, unloaded AGV for continuing operation, as well as it is natural for an upload cell to send out requests for an AGV loaded with workpieces. But

both operational cell types share the problem of how to dispose of the currently docked AGV.

The transport manager solves the AGV disposal problem by having a number of buffer docks registered. A buffer dock is not directly associated with any operational cell, but is physically located in the vicinity of an operational dock. A buffer dock is registered into the transport manager with the load state of the AGVs relevant for it. Thus, a buffer dock near an operational dock for the upload cell should be registered as accepting AGVs loaded with workpieces, whereas the buffer docks near the supply cells should be registered for accepting empty AGVs. The disposal task of the transport manager thus serves two purposes:

- It solves the problem of getting AGVs out of the way from operational docks, when the AGV is dismissed from the pertinent operation.

- It tries to optimize the transport tasks of the production system by a pre-fetch mechanism, by which idle AGVs are sent to appropriate buffers in the vicinity to their next expected usage.

An endless number of further optimizations is possible as extensions to the currently implemented mechanism for pre-fetching AGVs. The current mechanism simply selects a dock at random, from the set of appropriate buffer docks for an idle AGV. The appropriateness of a buffer dock for an AGV is based entirely on whether the AGV's load state matches the load state for which the dock is registered. Many immediate optimizations may be implemented as strategies for inspecting and computing at various sophistication levels, where a buffered AGV is likely to have the shortest time to wait for operation in the associated operational dock. Such strategies may depend on the nominal or estimated rate of "consumption" of AGVs in the operational cells serviced by the buffer dock, and, of course, the operational status of the associated operational cell.

The transport manger is the strongest point of centralization in the current prototype production control system. In a control system based on more autonomous agents, the solution may be that the supply cells would be implemented to obtain awareness of and association with an upload cell in near-future need of workpieces, and thus inferred where to send the AGV. Alternatively, such awareness could have been located with the individual AGVs or with the upload cells. The centralized approach taken in the current implementation, by which the transport manager eliminates the need for autonomy and awareness, was motivated by its simplicity. This approach is known to result in poor scalability and low flexibility; for instance, all AGVs are handled equally and with very little intrusiveness. An approach allowing more autonomy and diversity in the requesting, operational production system entities and in the AGVs themselves may lead to much better system scalability and higher production flexibility; at the price of more complexity in design and implementation, however.

AGVs

The AGVs are software representatives and controllers of the physical transport devices. In essence they are the entities which bind together AGV velocity controller devices and AGV localization services with the production control aspects of physically transporting workpieces.

An AGV software object has, as its central control task, the responsibility for the low-level trajectory planning and control, which leads to achievement of a desired gross motion of the physical device. The achievement of the endpoint pose of the motion is the production system task associated with transport by the AGV.

In the current prototype system, the design does not encompass such features as traffic control, prioritization of transport tasks, or AGV interference avoidance. The trajectories of AGVs are thus simply planned and controlled along the direct paths to target pose. The implemented AGV system only works for testing purposes by optimally fulfilling the direct transportation tasks, disregarding anything directly or indirectly involving interferences and collisions of the AGVs, except for access control to docks; which is considered part of the production system constraints.

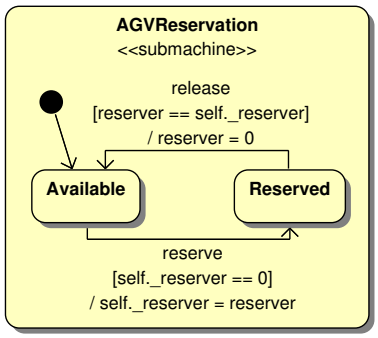
AGVs are on the conceptual border between production controllers and production devices. Their main control objective, being motion and trajectory control, is device-oriented, and thus may be classified as production devices. However, much of the motion and trajectory planning, encapsulated in the AGV software, touches upon production control aspects. Continuing this trend in future design, i.e. putting more advanced production control related responsibilities into the AGV software, will render the AGVs increasingly as production system controllers.

Due to the quite low amount of production system aspects in the current, simple AGV software design and its unrealistic state for real deployment, the AGV software will not be further discussed. Still, the AGV software is some of the more complex code in the production control system. Therefore, its implementation in the AGV class is included in Appendix A.2.5 for reference. The *AGV* interface provided by the AGV class to the production control system is included in Appendix A.1.5.

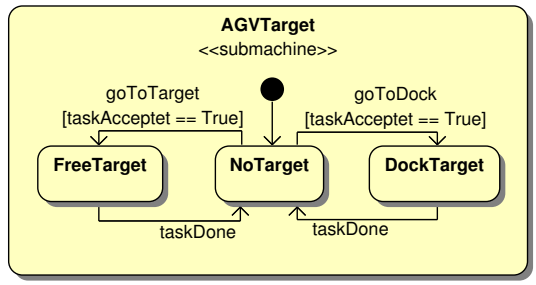
Certain central features of the AGV software design is expected to be usable also in a future, more realistic, implementations. This regards the design of three concurrent, internal state machines illustrated in Fig. 3.11.

The AGVReservation state machine in Fig. 3.11(a) takes precedence for all logic operations in the AGV object, which basically rejects all requests originating from a requester which does not have reservation of the AGV.

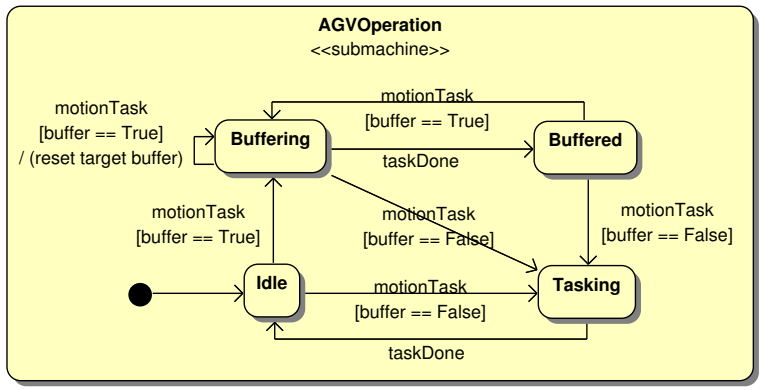
The AGVTarget state machine, in Fig. 3.11(b), holds information of the kind of motion task, and hence the kind of motion target, is currently in progress. The target type is determining of some internal tasks and activities of the AGV object. The states and



(a) *UML state machine diagram for the reservation states of an AGV.*



(b) *UML state machine diagram for the target states of an AGV.*



(c) *UML state machine diagram for illustrating the dynamics of the operation states of an AGV.*

Figure 3.11: *UML state machine diagrams illustrating the dynamics of the internal state machines of an AGV. The three state machines are concurrent in the* AGV *objects, the logic of which depends on and manages the state transitions of the state machines.*

dynamics of this state machine is supposed to be generally extended, with corresponding extension of motion control strategies, and customized for the detailed use in production control systems. In the prototype production control system, only the `DockTarget` is in active use, since all requests to the AGVs regards transport to docks; the `NoTarget` state is used when there is no target for the AGV.

The AGVOperation state machine is controlling the operational states of an AGV object. As seen in Fig. 3.11(c), it is the one with the highest number of states and the highest complexity of transition dynamics. It mainly controls, and is controlled by, requests of motion tasks. The `Tasking` state is associated with a real production system task, such as moving to service of, or staying posed at, an operational cell. The `Tasking` state naturally takes precedence above all other states, and transition out of it is only possible with the completion of the task. Correspondingly, all other states allow transition to the `Tasking` state, in recognition of their lower importance to the production system control. The two

states `Buffering` and `Buffered` are associated with motion and posing for the pre-fetch optimization of the transport system. The AGV is in the `Buffering` state while moving to a buffer dock, and at the end of such motion, the state changes to the `Buffered` state; which resembles the `Idle` state by signalling that the AGV is at rest. The important difference between the `Idle` and `Buffered` states, and which justifies the existence of the `Buffered` state, is that in the `Buffered` state the AGV has a purpose with being posed where it is; in the `Idle` state, the AGV has no special purpose with being anywhere in particular.

3.4.5 Carrier and Conveyor Management

The carrier management is a service for the upload cells to acquire 3D information about upload sites for workpieces on a painting system carrier. The conveyor manager is a service to the carrier manager, for orchestrating a shift of carriers in which a filled carrier at the upload PnF-stop is sent on to the painting system and a new, empty carrier from a carrier storage or buffer is sent to the PnF-stop at the upload area. The carrier and conveyor management objects, of classes CarrierManager and ConveyorManager, together with the PnF-stop objects, of class PnFStop, are found topmost in the prototype control system overview of Fig. 3.2.

Carrier Management

The carrier manager is implemented by the CarrierManager class in the code file included in Appendix A.2.6.

The carrier manager serves two purposes in the production control system:

- To support all upload cells in their requests for free upload sites for workpieces on the painting system carrier. This is a reactive service provided by the carrier manager.
- To keep account of free sites on the painting system carrier with potential to be filled by some upload cell, in order to timely command the shifting of carrier. This is an internal, continuous task undertaken by the carrier manager.

The design of carrier management includes the issue with having several concurrent upload cells, each of which may cover only a subset of the upload sites of a given painting system carrier. For instance, the two upload robots in the laboratory prototype system are limited in reach to the proximal sites of a particular painting system carrier. Therefore the regions in the carrier management problem in the UploadCell and CarrierManager classes are referred to as "sides". The painting system carrier sites are thus, following the vision analysis for upload site localization on a newly arrived carrier, partitioned into the left and right sides. The `uploadLeft` upload cell should register with the carrier manager

that it covers the left sites of a carrier, and likewise the `uploadRight` upload cell registers to cover the right side.

Like for many other detailed mechanisms in the design of the prototype control system, this is an instance where the detailed logic for a particular configuration is directly implemented in the code of the control entity. The separation of sites into relevant regions, as well as the specification of installation of a particular workpiece type on a site of a particular painting system carrier, must, for the system to be agile, or at least flexible, have its design and implementation based on dynamic strategies and plug-in structures.

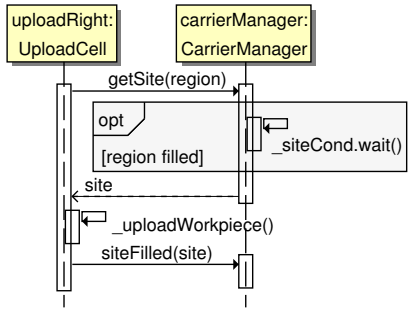


Figure 3.12: *UML sequence diagram for the request from an upload cell to get an upload site on the painting system carrier. Initially the carrier manger checks if any more sites are available for the region addressed by the getSite() request. If no more sites are available, the requester is blocked while waiting for the carrier to be shifted, and new sites to become available. As an upload site is eventually returned, it is marked in the carrier manager as reserved. The site will be cleared from the site-accounting when the upload cell have performed its upload operation; signalling this by the call to the siteFilled() method of the carrier manager.*

The details of the operations involved for supporting an upload cell in getting a site for uploading a workpiece is shown by the UML sequence diagram in Fig. 3.12. At the request for a site for a given side the carrier manager checks if there are remaining free sites on the given side. If not, the caller is blocked until notification of new sites, in the advent of arrival of a new carrier. A free site is eventually marked as reserved and returned to the upload cell. As the upload cell completes its upload operation to the site, it notifies the carrier manager that the site have been filled; which removes the site from the accounting of sites for the pertinent carrier.

The task of monitoring the upload status of any carrier is an internal, continuous task for the carrier manager. It involves monitoring the total number of reserved sites and free sites with potential to be filled. A site that has potential to be filled defined as a free site in some region of the carrier for which an upload cell is registered. It means that some upload cell has taken the obligation to fill the site, when it is returned in a *getSite()* request. When the sum of the number of reserved and free sites goes to zero, the carrier is defined to be full. It is so in the sense that any remaining site which does not have had a workpiece attached has no registered upload cell which may fill it. The condition of a carrier being declared full triggers the internal task of disposing of the current carrier, and providing a new, empty carrier from the PnF-conveyor. This is achieved by using

the service of the conveyor manager. After arrival of a new, empty carrier, it is analyzed by the vision system for localizing upload sites, which are then partitioned according to the site regions for the carrier. Finally, all upload clients awaiting new carrier sites are notified that a new carrier is in place.

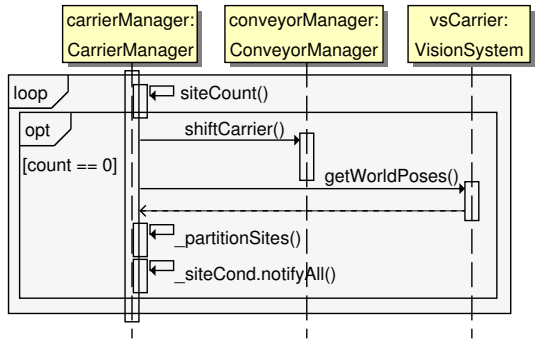


Figure 3.13: *UML sequence diagram for illustrating the active loop internal to the carrier manager. Its main purpose is to monitor the potential number of free sites on the current upload carrier. At a count of zero, the shift of carriers is ordered to be handled by the conveyor manager. As a new painting system carrier has arrived, the associated vision system is requested to analyze for the locations of all free sites for upload. These are partitioned according to the "sides" or regions for which upload cells have registered their coverage. At the end, all waiting upload clients are notified that there are now new, free sites available.*

The detailed interactions of the carrier manager, involved with shifting the carrier, is illustrated in the UML sequence diagram in Fig. 3.13.

A minor flaw with the mechanism used to obtain and store carrier sites in the current design of the carrier management is that all sites are identified and localized only once, on the arrival of the new carrier. Sites are thereafter referred to by their 3D pose in world coordinates. For this to work in reality, the mechanical stability and fixture of the carrier must be ensured. However, in the current laboratory prototype system, the carriers are free to rotate around a horizontal axis normal to the plane spanned by the carrier structure. This means that uploading a workpiece on one side shifts the equilibrium angle of the carrier, and hence shifts all upload site in world coordinates by a rotation around the carrier attachment axis. Since such a shift in position of any site is outside tolerance for the attachment motion process for the particular workpieces and carriers used, the operation based on the current design of the carrier management and upload process will fail. However, the emulated version of the laboratory prototype system assumes that the carriers are mechanically stabilized throughout the upload process, without explicitly including a fixture mechanism. It is possible, and feasible, to model and control a manipulator for fixing the freely hanging bottom of a carrier, as an integral part of the arrival of a new carrier at the upload site. This solution is possible and realistic to implement in the real, as well as in the emulated, laboratory prototype system.

Changing the design of the implemented carrier management is possible, and involves use of vision localization of sites in every request from the upload cells. For stability

reasons, such that no site doublets are registered, structural recognition of the upload sites is necessary for matching the cached, reserved sites to the right sites from a vision localization.

Conveyor Service

The conveyor manager provides a simple service to the carrier manager for shifting a painting system carrier filled with workpieces out and shifting a new, empty carrier in at the PnF-stop at the upload cells. The production system aspect is that the trolley of a carrier filled with workpieces is to move on to a transfer area, where carriers are transferred from the PnF-conveyor to the main conveyor through the painting system. This section details the rather fixed orchestration involved with shifting the carrier at the upload cells.

The code for implementing the CarrierManager class is included in Appendix A.2.7. The *PnFStop* device interface used to control the PnF-stop devices is found in Appendix A.1.4.

The PnF-conveyor has three PnF-stops involved with the shifting of carriers. The PnF-stop associated with the upload area is called `Upload`. It is located where the carrier manager has its vision system for identifying carriers and localizing upload sites on the carrier, and such that it is withing reach of the upload robots of the upload cells. As close as possible to, and upstream on the conveyor from, the `Upload` is another PnF-stop, called `UploadBuffer`. It is positioned as close as possibly allowed by the geometrical extent of the carriers not to interfere, for giving the shortest shift time of the upload carrier. A distant PnF-stop, called `CarrierStorage`, is located upstream from the `UploadBuffer`, at an appropriate location for many trolleys with empty carriers to stand in line, blocked against each other. This is, as the name indicates, a mass storage of carriers. Whenever the `UploadBuffer` has released its trolley to refill the upload area with an empty carrier, a new carrier should be supplied from the `CarrierStorage` to the `UploadBuffer`.

The detailed sequence of interactions of the conveyor manager with the PnF-stop controllers is illustrated in the UML sequence diagram in Fig. 3.14. The detailed logic of the orchestration involves three conditional fragments, which serve the purpose of starting up normal operation from transient states; such as entirely depleted `Upload` and `UploadBuffer` stops at initialization. In understanding the logic of the sequence diagram, it is important to note that the semantics of invoking the *releaseTrolley()* method on a *PnFStop* interface is that the calling thread is blocked until a trolley has passed the associated stop, regardless of whether a trolley was blocked at the stop prior to the invocation. This gives a potential deadlock, if *releaseTrolley()* is invoked without a trolley is blocked or on its way towards the stop. The logic sequence in Fig. 3.14 is safeguarded from this deadlock potential.

Some of the checks in Fig. 3.14 also serves to correct errors in the real, as well as the emulated, conveyor system. Errors may arise if two trolleys are released when only one was ordered. In a certain sense, these checks and their associated actions may not recover

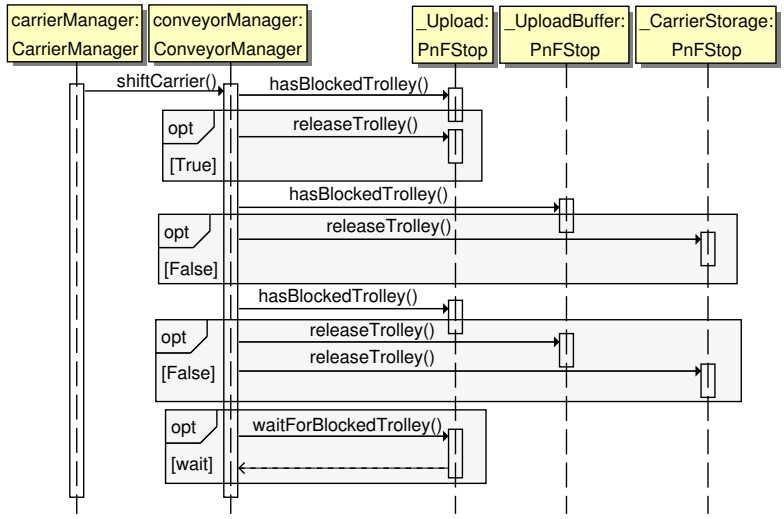


Figure 3.14: *UML sequence diagram for illustrating the interactions of the the conveyor manager in response to the request from the carrier manager with to shift the current, filled upload carrier with a new, empty carrier. It is a simple, conditional orchestration of the PnF-stops of the conveyor to ensure that a carrier at the upload site is released, a new carrier is moved to the upload site, and that the carrier buffer site for upload is replenished from the carrier storage PnF-stop.*

the real system from such types of errors. If, for instance, two trolleys with carriers are blocked at the `Upload` stop, the entanglement of the pegs between the two carriers will certainly make any upload attempt fail, if the vision system is indeed able to actually localize the entangled upload sites. It may even be the case that two entangled carriers may stick together when one trolley tries to leave the PnF-stop, with unpredictable results. For these reasons the error recovery included with the logic displayed in Fig. 3.14 should rather be perceived as operational verifications, with emergency stop as the result of failure of any of the checks.

Chapter 4

Included Publications

This chapter is a compilation of pre-prints of selected published and submitted papers prepared during the period of the PhD scholarship. The compilation is to be considered the main contribution of this thesis.

In the following is found a bibliography of the included papers, and following that the pre-prints of the papers themselves are included in order of publication, each in their own section. Each section for a paper opens with the bibliographic information for the contained paper and a short declaration of contributions.

Bibliography of Included Publications

Terje Kristoffer Lien and Morten Lind. Instrumented fixtures for on-line correction of welding paths. In Mamoru Mitsuishi, Kanji Ueda, and Fumihiko Kimura, editors, *Manufacturing Systems and Technologies for the New Frontier*, Engineering, chapter 11, pages 435–438. Springer London, May 2008. ISBN 978-1-84800-267-8. doi: 10.1007/978-1-84800-267-8_89.

Morten Lind, Olivier Roulet-Dubonnet, Per Åge Nyen, Lars Tore Gellein, Terje K. Lien, and Amund Skavhaug. Holonic Manufacturing Paint Shop. In Vladimír Marík, Thomas Strasser, and Alois Zoitl, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin/Heidelberg, September 2009. ISBN 978-3-642-03666-8. doi: 10.1007/978-3-642-03668-2_20.

Olivier Roulet-Dubonnet, Morten Lind, Lars Gellein, Per Nyen, Terje Lien, and Amund Skavhaug. Development of a Holonic Free-Roaming AGV System for Part Manufacturing. In Vladimír Marík, Thomas Strasser, and Alois Zoitl, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, pages 215–224. Springer Berlin/Heidelberg, September 2009. ISBN 978-3-642-03666-8. doi: 10.1007/978-3-642-03668-2_21.

Morten Lind, Johannes Schrimpf, and Thomas Ulleberg. Open Real-Time Robot Controller Framework. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 13–18, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4.

Johannes Schrimpf, Morten Lind, Thomas Ulleberg, Chen Zhang, and Geir Mathisen. Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 19–23, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4.

Olivier Roulet-Dubonnet, Morten Lind, and Terje Kristoffer Lien. Development of a Low-Cost Prototype AGV. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly*

Technologies and Systems, pages 25–29, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4.

Morten Lind and Olivier Roulet-Dubonnet. Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 67–72, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4.

Morten Lind and Olivier Roulet-Dubonnet. Holonic shop-floor application for handling, feeding, and transportation of workpieces. *International Journal of Production Research*, 49:1441–1454, 2011. ISSN 0020-7543. doi: 10.1080/00207543.2010.519115.

Morten Lind and Amund Skavhaug. Using the blender game engine for real-time emulation of production devices. *International Journal of Production Research*, 0(0):1–17, 2011. ISSN 0020-7543. doi: 10.1080/00207543.2011.601772. Online available, iFirst.

Morten Lind and Johannes Schrimpf. PyMoCo – Python-Based Robot Motion Control. *Journal of Software Engineering for Robotics*, 2011. ISSN 2035-3928. In preparation.

4.1 Instrumented fixtures for on-line correction of welding paths

Terje Kristoffer Lien and Morten Lind. Instrumented fixtures for on-line correction of welding paths. In Mamoru Mitsuishi, Kanji Ueda, and Fumihiko Kimura, editors, *Manufacturing Systems and Technologies for the New Frontier*, Engineering, chapter 11, pages 435–438. Springer London, May 2008. ISBN 978-1-84800-267-8. doi: 10.1007/978-1-84800-267-8_89

Declaration of co-authorship

The idea and project setup was entirely a contribution from Terje Lien. The working principles and conceptual design was a cooperative effort from Terje Lien, Erik Haga (at that time, a master student at IPK) and Morten Lind.

The detailed mechanical and geometrical analysis was a cooperative effort by Erik Haga and Morten Lind. The detailed mechanical design and machine construction is entirely a contribution by Erik Haga. Morten Lind contributed the design and implementation of the computational software and control system. In close collaboration, Erik Haga and Morten Lind performed comprehensive laboratory testing of the fixture. For the final validation Erik Haga and Morten Lind brought the fixture to a welding laboratory at Hydro Aluminium Structures Raufoss AS, where several test welds were performed in collaboration with personnel there.

Terje Lien is the main author and writer of the paper. Morten Lind contributed with writing the Sections 4, 5, and 6, and produced the graphics therein; not the photos. Morten Lind's written contribution was thoroughly reviewed and revised by Terje Lien.

Instrumented fixtures for on-line correction of welding paths

T.K.Lien[1] and M. Lind[1]
[1] Department of Production and Quality Engineering,
Norwegian University of Science and Technology, Trondheim, Norway

Abstract
Robotic welding of thin-walled aluminium structures is very sensitive to deviations in parts' positions. Position deviations give inferior weld quality. Measurement devices integrated in the clamping elements of the welding fixture can provide data to the welding robot for dynamic correction of the welding path. The paper describes a system of integrated measurement in clamping systems. Various approaches to measurement of critical dimension deviations are evaluated and optimal solutions have been selected. The obtainable accuracy is analyzed and verified through laboratory tests. A microcontroller-based control system has been developed to process and transmit the measurement data to the robot controller.

Keywords:
Robot; Welding; Measurement; Fixture

1 INTRODUCTION

All manufacturing operations that are performed automatically by some sort of machinery need a firm positioning of the work piece. This is one of the most fundamental truths of manufacturing. In many cases this is not a big problem either. But in modern highly automated processes where both the primary operation and the product or component has been automated challenges appear also in this area. The removal of human interference in the positioning and fixation of the work piece sometimes leads to challenges in securing a proper positioning to assure the wanted outcome of the manufacturing process.

2 HANDLING LARGE PART DIMENSION VARIATIONS

The problem of part tolerances outside the process window

Even products with wide tolerances can present big challenges to automation. One typical problem is that neither the final product nor the ingoing parts in an operation have particularly narrow tolerances. But the process in itself may require a much tighter tolerance to yield a good quality result. Typical examples in this category are grinding, polishing and welding. The case examined in this paper concerns welding.

The parts in many weldments can have quite wide tolerances if the weld is manually performed. Even if the parts have slight variation in relative positions from product to product the yield of the process is good since the operator will adjust the welding path according to observation in each case. It is not so in automatic welding. The robot or automatic welding machine will assume that the parts are identically placed in each operation. But it may well be that the parts are offset more than 1 mm from the ideal position even on relatively small parts. Such large offsets will create problems in the welding of thin walled components. Because of the thin material only small welding currents are allowed. The filling capacity is thus limited and the burn-in into the base material can be too low to assure proper welding strength.

Performance requirement in the case study

In this case study the challenge was to weld thin walled extruded box profiles to a base plate. The product is shown in figure 1. This is a simplified example of a component in automobile bumper system, the so called crash box. The wall thickness of the box typically lies in the region around 2mm, while the outer dimension tolerance of the box is in the order of ±0.9 mm in the worst case. The expected variation in outer dimension of the box is too large for a good quality automated welding process.

One possible solution to the problem is to use the well established method of edge detection by means of the welding gun as a touch probe [1,2]. But this method has limited accuracy, and it is time consuming since it has to be performed after the parts in the weldments has been locked in the fixture and before the actual welding starts. The time consumed for this pre-weld check was considered to be too high. It would severely reduce the productivity of the automated welding operation.

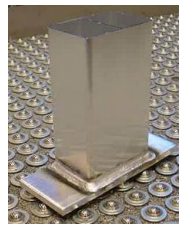


Figure 1: The study object, a simplified crash box

The alternative that was studied was to include measurement of the position of the extruded box as it is clamped in the fixture. By using this method the exact position of each box would be known at the moment it was clamped. This position information could then be used to adjust the welding path for the automatic welding operation.

For best quality welding the welding path should be corrected to within ±0.1mm of the ideal path. This is a conservative requirement to make sure that local variations along the weld path will not give risk of inferior local quality.

3 METHODS FOR MEASUREMENT COMBINED WITH CLAMPING

Evaluation of possible methods

The measurement of the position of the workpiece while clamping can be done either directly or indirectly through the clamping elements. Both possibilities have advantages and disadvantages as shown in table 1 [3,4].

Table 1: Comparison of measuring methods

Measuring method comparison	
Direct measurement on object	
Advantages	Disadvantages
• Most precise method	• Measurement device will interfere with clamping device or welding gun • Expensive measurement devices
Indirect measurement through clamping element	
Advantages	Disadvantages
• Measurement device away from clamp and weld area • Low costs measuring devices can be used • Mechanical protection of measurement device easy	• Less accurate due to coupling through clamp • Non-linearity in some of the applicable measurement methods

Table 2: Characteristics of clamp position measurement methods

Method	Merits
Rotational encoder on the clamp arm shaft	Low cost robust solution. Non linear measurement
Laser distance measurement of clamp arm position	Linear measurement Robust solution High cost
Pneumatic linear distance measurement of clamp arm position	Linear measurement Robust low cost solution Short measurement range
Inductive linear distance measurement of clamp arm position	Linear measurement Robust low cost solution Short measurement range
Touch probe linear distance measurement of clamp arm position	Linear measurement Less robust High cost

The evaluation of measuring methods placed strong emphasis on robustness, simplicity and reliability. Thus the qualitative advantages and disadvantages listed in table 1 lead to the clear conclusion that indirect measurement through the clamping elements was the better method.

For clamping both linear and rotational clamps were considered. Again the space requirements lead to the conclusion that rotational arm clamps would be the best solution. It then remains to decide which measurement system that would be most suitable for measurement of the clamp movement. Table 2 shows the candidates and their merits [3,4].

Again the requirement for robustness and low cost were dominating. In addition it was realized that only the rotational encoder could be fitted without any major mechanical interference problems with any other element of the system. The final design ended up with rotational clamps that included incremental rotary encodes for position measurement of the clamped surface.

4 THE COMPLETE INSTRUMENTED WELDING FIXTURE

Figure 2 shows the laboratory version of the instrumented welding jig. The work piece is clamped against two vertical fixed walls with three contact points on one wall and two on the other to ensure orthogonality and stable positioning. The variation in dimension of the workpiece can then be measured directly on the two surfaces pressed on by the clamps.

The clamps are pneumatically operated. The operation of the clamps and reading of the measured signals is controlled by a local microcontroller that communicates with a PC that acts as an overall process controller. This PC generates the path correction data for the welding robot controller.

Figure 2: Laboratory version of the complete welding jig with one work piece in position ready for clamping.

The precision of the selected clamping method

Figure 3 shows a sketch of the claming-measurement arm principle. In an ideal design the distance x in figure 3 should be 0 for the calibrating position. But for mechanical reasons the clamping arm had to be L-shaped, thus there will be a substantial x value in the measurement setup. In this design the clamp arm had the dimensions X=20 mm, Y=20 mm giving a total length L=101.98 mm.

This gives an amplification of the non-linear behaviour of this measuring principle. It is of interest to evaluate how large the error due to non-linearity will be. The measurement system is calibrated against a master block with dimensional tolerance better than ±0.01 mm. This assures a reference for the measurement system with an accuracy one order of magnitude smaller than the required correction accuracy.

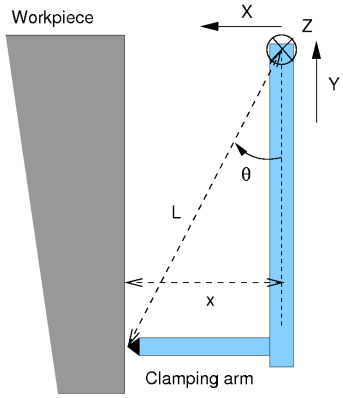


Figure 3: Sketch of the principle of the rotating arm clamp and measure device.

Let the angle θ_0 represent the reference angle corresponding to the reference position x_0 as is given in equation (1)

$$x_0 = L\sin\theta_0 \tag{1}$$

A small measurement distance Δx_s relative to x_0 can be expressed as:

$$\Delta x_s = L(\sin(\theta_0 + \Delta\theta_s) - \sin\theta_0) \tag{2}$$

For simplicity the measurement uses a linearization of Δx_s around x_0 as given in (3):

$$\Delta x_s = -\Delta\theta_s L\cos\theta_0 \tag{3}$$

The error due to this linearization is the difference between equations (2) and (3):

$$\Delta x_{err} = L\left[\sin(\theta_0 + \Delta\theta_s) - \sin\theta_0 + \Delta\theta_s\cos\theta_0\right] \tag{4}$$

Figure 4 shows the variation in Δx_{err} as function of Δx_s for different values of x_0. This error is one order of magnitude less than the resolution of the measuring system.

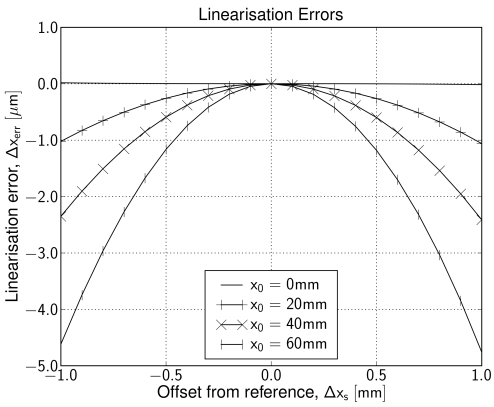


Figure 4: Theoretical measurement error Δx_{err} as function of measurement distance Δx_s for various x_0 values.

5 CONTROL SYSTEM FOR ROBOT PATH CORRECTIONS

The control PC is the central unit, through which a user initiates all action chains. The user interface developed has three modes: communication setup, calibration control, and production control. In communication setup mode, communication to the micro controller and the robot controller is specified and verified. In calibration mode, a semi-automatic calibration routine can be performed and stored. Production mode presents to the user a chain of check points for the whole operation cycle. The check points serve only safety purposes for the prototype. The necessary external signals for fully automated operation are that the welding is done and that a new crash box has been positioned in the fixture.

The robot controller is set up with a program where the welding path is taught from the nominal crash box. The robot controller has program code for receiving offsets over the RS232 interface from the control PC. Further, the robot controller listens over the RS232 interface for a command from the control PC for executing the offset corrected weld program.

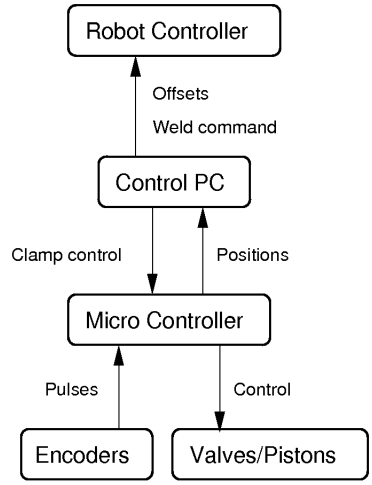


Figure 5: Block structure of the control system.

The micro controller traps pin change interrupts on one parallel port, where the 3 channels from each encoder are connected. Two of the lines from an encoder forms the quadrature pulse train, on basis of which the encoder position counter is updated. The third line for an encoder gives the index pulse, generated at a fixed position in the encoder. The index pulse interrupt from an encoder resets the encoder pulse counter to zero, thus eliminating possible drift by loss of pulses. The micro controller will reply with instantaneous encoder positions over the RS232 interface by request.

The pneumatic system of 3 double acting pistons is controlled by 3 solenoid controlled valves. One parallel port of the micro controller has 3 pins dedicated to control the gates of 3 MOSFETs, which controls the valve solenoids. The RS232 interface exposes direct control of the valves to the control PC.

6 LABORATORY VERIFICATION OF THE PRINCIPLE OF MEASUREMENT

The selected encoder for the clamp arm has a resolution of 5000 pulses/rev. With quadrature counting principle this gives 20000

counts/rev and the selected arm design gives a resolution of 32 µm/count. This should give sufficient accuracy.

A laboratory test was performed to verify that the expected measurement resolution 30 pulses/mm did not create problems for the measurement linearity. The test was performed by using the calibration block as base. Successively laboratory grade gauge blocks were stacked to create an offset from the calibration block, starting with a 1 mm gauge block as the offset 0 point. Figure 6 shows the measured result. It is in perfect agreement with the expected performance, giving 3 pulses per 0.1 mm offset change. The change in pulse steps is also very regular switching between 1 and 2 pulses per 0.05 mm. This regular pattern assures that no extra inaccuracy is introduced by the granularity in measurements near the reference point.

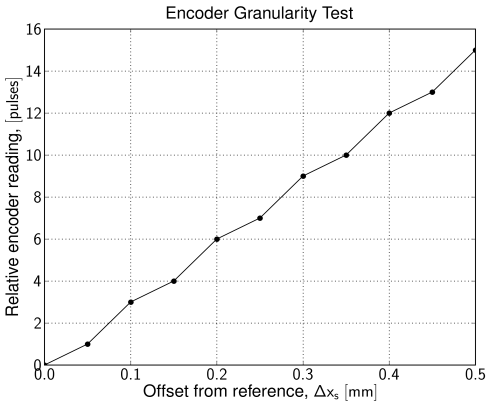


Figure 6: Granularity test for small linear offsets

7 LABORATORY WELDING TESTS

Real welding tests were performed to verify the performance of the correction system. Figure 7 shows a cross-section of a perfect weld performed with optimal path programming on a part with nominal dimensions. Figure 8 shows a weld on a part with 2 mm offset of the box wall. This weld has inferior quality because of too little melt-in into the box wall. Use of the path correction system results in the weld shown in figure 9 for a wall offset of 2mm. This is a weld of similar quality as the reference weld.

A series of similar welding tests were performed. The all gave similar results. The industrial partner in the project feels confident that the system performs as required to compensate for dimension tolerance on ingoing parts.

8 SUMMARY AND CONCLUSIONS

- A system has been built for making measurement of part dimension offset combined with the clamping action in a welding fixture.
- A theoretical model for the measurement error has been developed.
- A control system to operate both clamping and offset measurement has been developed.

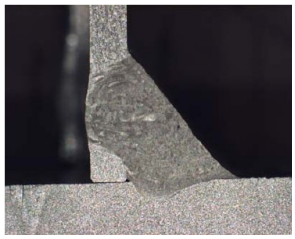


Figure 7: Reference weld of satisfying quality.

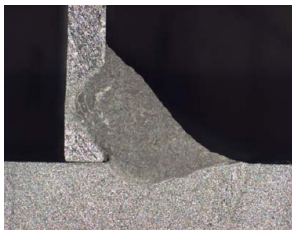


Figure 8: Weld test with 2 mm offset uncorrected, insufficient melt-in on box wall

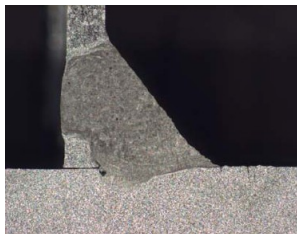


Figure 9: Weld test with 2mm offset corrected through feed forward correction from the clamping measurement system

The complete system has been tested in a laboratory setup. The laboratory tests verified that the system performed as expected. It enables the measurement of individual parts offset during the clamping cycle to generate offset command to the welding robot before the welding operation starts. The corrected weld satisfies the quality requirement for automatic welding of the investigated and similar parts.

9 ACKNOWLEDGMENTS

This research work was sponsored by Hydro Aluminium and the Norwegian Research council

10 REFERENCES

[1] Ågren, B.G., 1995, Sensor integration for robotic arc welding, Department of Production and Materials Engineering, Lund University.

[2] Cary, H.B., 1995, Arc Welding Automation, Marcel Dekker, NY.

[3] Bolton, W., 2003, Mechatronics – Electronic control systems in mechanical and electrical engineering, Pearson Education Ltd, Harlow, England.

[4] Cetinkunt, S., 2007, Mechatronics, John Wiley &Sons, Hoboken, NJ.

4.2 Holonic Manufacturing Paint Shop

Morten Lind, Olivier Roulet-Dubonnet, Per Åge Nyen, Lars Tore Gellein, Terje K. Lien, and Amund Skavhaug. Holonic Manufacturing Paint Shop. In Vladimír Marík, Thomas Strasser, and Alois Zoitl, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, pages 203–214. Springer Berlin/Heidelberg, September 2009. ISBN 978-3-642-03666-8. doi: 10.1007/978-3-642-03668-2_20

Declaration of co-authorship

The initial project conception for the laboratory demonstration is credited to Terje Lien, Per Åge Nyen, Lars Tore Gellein, and Pål Ystgaard. Per Åge Nyen made the initial process and simulation model for the uploading station; with the central operation of the laboratory demonstration system.

The conceptual analysis and design, with respect to holonic manufacturing and holonic control, of all elements, operations, and devices for the general laboratory system presented was made by Morten Lind, with involvement and contributions from Olivier Roulet-Dubonnet. For the same laboratory system, Olivier Roulet-Dubonnet was concentrating on the parallel activity of developing the holonic AGV system, briefly mentioned in the paper. There was a valuable and frequent exchange of ideas between the two activities; the same paradigmatic basis and the addressing of the same laboratory system made this natural.

Morten Lind was the sole writer of the paper with frequent reviewing and commenting by Olivier Roulet-Dubonnet in the writing process. In the final revision rounds, Amund Skavhaug gave a valuable and comprehensive contributions. The paper was review and commented by Terje Lien before submission. The graphics in Figure 1 is supplied by Per Åge Nyen from his simulation setup. All remaining graphics is designed and produced by Morten Lind.

Holonic Manufacturing Paint Shop

Morten Lind[1], Olivier Roulet-Dubonnet[1], Per Aage Nyen[2], Lars Tore Gellein[2], Terje Lien[1], and Amund Skavhaug[3]

[1] Department of Production and Quality Engineering,
The Norwegian University of Science and Technology, Norway,
morten.lind@ntnu.no
[2] SINTEF Raufoss Manufacturing, Norway
[3] Department of Engineering Cybernetics,
The Norwegian University of Science and Technology, Norway

Abstract. In pursuit of flexibility and agility within discrete manufacturing, the surrounding logistics and handling processes of a paint shop is under construction as a laboratory prototype application. Holonic Manufacturing seems to be a promising strategic paradigm and architecture to use for a system characterised by production logistics and control. This paper describes the physical devices to be used; the desired functionality; and the basic logic control designed. Additionally, the ideas for holonification based on the already designed logic control is presented.

1 Introduction

Distributed decisions and coordination of autonomous sections in manufacturing have been around as long as complex manufacturing. By complex manufacturing in our context, we mean dealing with multiple complex products and extensive sharing of manufacturing equipment across different simultaneous product variants, i.e. with frequent changeovers and reconfigurations on the shop floor. In the recent decades there has been focus on the flexible automation of these entities. The enabling technologies for this may be traced back to the birth of numerical control and computational intelligence in the 1950s.

Holonic Manufacturing is a paradigm for pervasive manufacturing automation, ranging from (and integrating with) the lowest level of real time shop floor control and all the way up to company or even corporate level. It covers most aspects of manufacturing, be it machine to machine cooperation or order to production department interaction.

The concept of a *Holonic Manufacturing System* (HMS) date back to the early 1990s when the *Intelligent Manufacturing Systems* (IMS) initiative set out a project with that name. The term *Holon* was coined by Arthur Koestler[1], some 40 years ago, for capturing the dualistic capabilities of autonomy and cooperativeness withing a single entity. The concept was found suitable to encompass the entities, physical as well as abstract, in manufacturing control and management.

There exist some architectures for Holonic Manufacturing Systems, such as PROSA[2] and ADACOR[3]. PROSA is strictly a reference architecture and

introduces the central concepts of basic holons: order, product, resource, and staff holons. High level scenarios illustrate the interactions of the different holon types. ADACOR is also an architecture, but with a different naming of the holon types. Notably the ADACOR supervisor holon differs from the PROSA staff holon, in that it formally coordinates the dynamics of holon aggregation and subordination. Leitão and Restivo applies the ADACOR architecture to a (partially simulated) machining and assembly workshop in several papers, see e.g. Leitão and Restivo[4, 5].

Two standards are highly relevant for Honlonic Manufacturing on very different levels: IEC61499 and FIPA; cf. Mařík et al.[6]. The IEC61499 standard regards a function block structuring of design and code for low level control oriented holons and their interactions. At the higher level, FIPA is a standard for ontology based agent communication.

The laboratory prototype described in this paper is a part of IntelliFeed, a cooperative project between the research institutions of the authors and industrial partners, supported by the Norwegian Research Council. Relevant and related projects also based at our research institutions are RAMP and CREAM; both part of the CRI NORMAN research program[7].

The particular laboratory prototype system we present here has its origin in the recognition that much manual labour is associated with the materials handling around paint shops in manufacturing industry. In itself, the full or partial automation of such systems will have a good potential for reducing trivial manual labour. But the potential is extended further if the automation is flexible and responsive, enabling the paint shop system to integrate with other parts of the entire manufacturing system.

This paper is organised as follows. In Sect. 2 the physical devices and their layout is described. Sect. 3, presents aspects of a holonification of elements and control. Discussion and future challenges is presented in Sect. 4.

2 Application Overview

In this section we describe the laboratory layout and the associated physical devices.

A real paint shop roughly consists of a painting system, a part upload station, a part download station, a painting carrier upload station, a painting carrier download station, and an overhead conveyor system. The painting carriers are hanging from the conveyor trolleys and are transported through all the stations and the painting system.

Our current goal is automation of the processes and materials handling at the part upload station. In the future, the other stations will undergo equivalent automation projects in a successive manner.

2.1 Laboratory Application Overview

A sketch of our initial laboratory system setup, currently used for simulation with QUEST, is shown in Fig. 1. The transport AGVs and a *Parts Arrangement*

Pre-print, HoloMAS2009

Station are not seen. The setup is planned for a Power-and-Free (PnF) overhead conveyor, though the conveyor type has not been decided for yet. As will become clear, a PnF conveyor is definitely to wish for when automating the uploading of parts to the painting carriers.

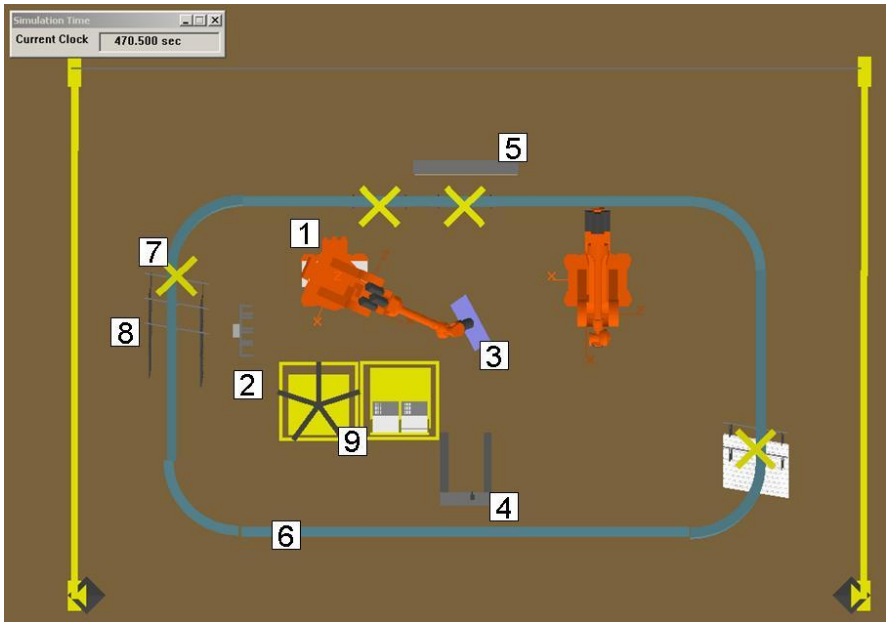


Fig. 1. Overview layout of a planned laboratory system. 1) *Upload Robot*, 2) *Upload Tool Rack*, 3) *Mounted Upload Tool*, 4) *Upload 3D Vision System*, 5) *Backlight Screen*, 6) *PnF Overhead Conveyor Track*, 7) *PnF Switch*, 8) *Painting Carriers*, and 9) *Local Part Storage Carriers*.

The robots we have available in the upload area are two Nachi SC15F, which are floor mounted within reach of each others and the conveyor track. Both are equipped with a 6D force sensor at the wrist. One of the Nachi robots have a modified controller, giving a direct $100 - 200\,Hz$ interaction with the servo controller using UDP over Ethernet. The other Nachi controller can easily be modified likewise.

In addition we have an ABB IRB 2400 with an S4C+ controller, also located in the laboratory but well outside the conveyor area. The ABB robot is connected to the rest of the system over an RS232 serial communication link. It has its own standard robot controller programmed in the complex RAPID control language. The program execution is hard real time, but interaction can only be through parameter modifications before real time execution.

Pre-print, HoloMAS2009

2.2 Part and Painting Carrier Handling

The part uploading is performed by a robot which has access to parts from a local part storage. The upload robot picks a part from one of the local storages and then attaches the part onto the presented painting carrier on the conveyor. Given the part and painting carrier type, as well as known painting carrier capacity state, the support system will be able to control the robot to attach the part at some free site on the painting carrier.

The painting carriers may have very different geometries and have a load capacity from one to several tens of parts. Frequently each painting carrier type may match different part types; this has, of course, a high impact on the sequencing of part batches. Some painting carrier types are adjustable and can host a whole family of parts types. The automatic handling of painting carriers is not a part of the initial laboratory system. Manual change of painting carriers will be performed, when need be.

The process of localising the attachment points on some of the simpler painting carriers, with the Scorpion 3D vision system from Tordivel, has already been implemented and verified. For stable identification and analysis of all painting carrier types, it may be necessary with more vision system or other sensory systems.

To ensure mechanical stability of the painting carrier under identification, analysis, and upload, a controlled clamping mechanism will be implemented underneath the PnF switch at the upload station.

2.3 Conveyor System

The conveyor system has not been decided yet. This decision is of cardinal importance for a lot of other hardware decisions, as well as the control logic.

We concentrate our efforts around the PnF conveyor, simply because of the severe implications which would arise by using the more common *Chained Trolley* conveyor[8]. The main difficulty with a chained trolley conveyor is that all trolleys travel at constant speed at all times. Thus either the parts must be uploaded onto the painting carriers in motion, or the painting carriers must be sidetracked to a small PnF conveyor loop with a buffer for reattachment to the main conveyor.

2.4 Painting Process Scheduling

For a given paint process setup, with parameters for rate of paint added and geometric configuration of parts, it is desirable to use the maximum speed of the conveyor while still meeting the paint quality requirements. Or, for given speed of the conveyor and geometric configuration of parts on the carriers, it is desirable to minimise the rate of paint added, while meeting the required paint quality, thus minimising the amount of wasted paint.

Upload and download capacity and equipment utilisation makes up a delicate trade-off. This depends on the pertinent part type, painting carrier type and painting process parameters. In a simple batch controlled system, where only

one part type and one painting carrier type is on-line at any time, there will often be under-utilisation in one or more of the upload station, the download station, and the painting system.

If mixing of parts within each painting carrier, or in the sequence of painting carriers, is allowed for, more freedom is given to the optimisation and the problems of under-utilisation can be remedied somewhat. The implication is a much more complicated logistics around uploading and downloading.

2.5 AGV System

Figure 2 shows a class diagram illustrating some important aspects of a multi-AGV system for material transport. Multiple AGVs are being built at our laboratory and the software system is in the design phase. The physical AGVs and AGV control system is based on earlier experience with a prototype AGV and vision based positioning system.

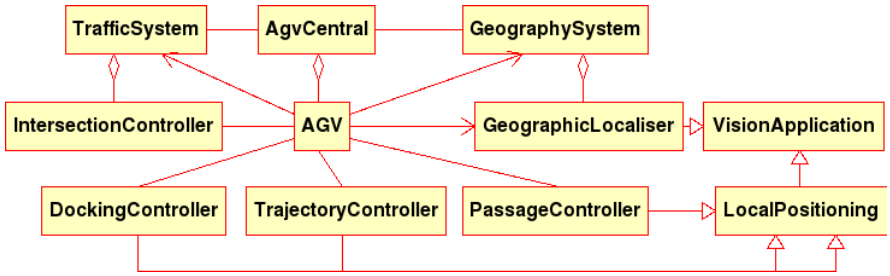


Fig. 2. A class diagram showing some important aspects and subsystems related to the AGV system.

The conceptual AGVs, being hosted entirely on the physical AGVs, are central components in the AGV system. We initially assume that all of the AGV localisation functionality will be based on vision applications. This is based on earlier successes with developing a vision based AGV localisation system.

The traffic system gathers information from and supplies information to the client AGVs about viable paths in the whole of the roaming area. It will be the the coordinator for pre-reservation of long term trajectories of the AGVs, whereas the AGVs themselves can react to short term (emergency) trajectory interferences. Specialised intersection controllers at known congested areas are regulatory rather than guiding in their relation with the approaching AGVs.

The geography system is the general global localisation system for the AGVs. It is demanded that the cameras of the geographic localiser servers cover the entire area where the AGVs may roam, and can serve the client AGVs with real time location information

Specialised conceptual controllers for local positioning and control may be defined to serve purposes relevant to situations or tasks, partly external to the

AGV system. Examples, relevant to our laboratory system are docking, trajectory, and passage controllers.

3 Holonification

Based on the initial plans, layouts, and experiments with uploading of parts, a logic control of a paint shop has been designed. The given design is more or less a traditional hierarchical control design.

In this section we sketch our ideas for holonification of the logic control system and the physical devices and layout in the previous sections. Ideally the designed explicit logic control will emerge out of the holonic control system currently under development.

We are not suggesting that we have a complete design for the holonic system, but this is a description of our initial thoughts and ideas on how a holonic architecture will be applied in the control and management system.

3.1 Resource Holons

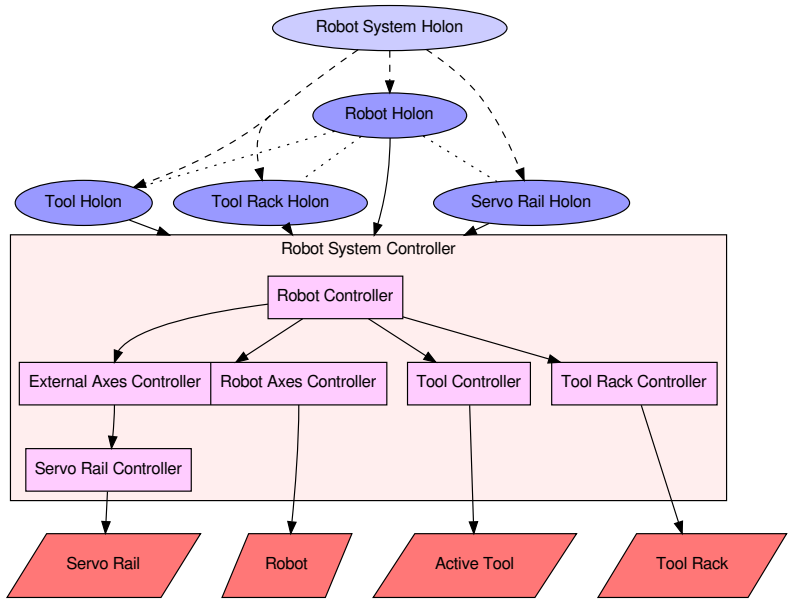
The resources, holons associated with physical devices, are related to the set of devices discussed in Sect. 2. This need not be a direct mapping, and indeed the devices described in Sect. 2 give too little detail to identify the entire set of resource holons.

The Robots plays a central role to the part handling processes. They are the devices with the hardest real time aspects which we try to holonify. Normally the real time aspects of the motion control is integrated with other real time tasks inside the commercial controller, and closed to the intervention in a broader system, cf. Fig. 3(a). This should not be so in a deeply holonified system, since it enables only a virtual high granularity of the holonified control; this is illustrated in Fig. 3(a), by the low level holons all addressing the integrated robot controller for accessing their logical control functionalities. The device controllers' functionalities are hierarchically organised inside the commercial controller.

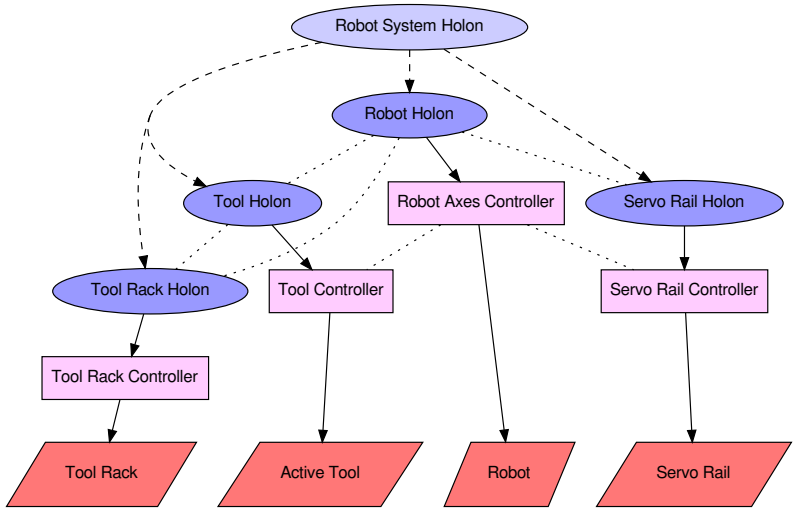
What is usually meant when referring to *a robot* in a manufacturing work station is the physical arm, the robot controller, and a whole range of various peripheral devices and their controllers, i.e. the whole *robot system*. These peripherals may consist of tools available for the robot, external (rail or gantry) axes for the robot, and even the whole range of sensors external to the robot arm. This is often well justified due to the *black hole* nature of commercial robot controller, in which coordination with external devices is possible only if these devices are integrated into, and controlled by the robot controller. The real time aspects of the entities integrated into the commercial controllers is a good reason for the tight integration, but is an example of both types of *lock-in* processes described by Mařík et al. [6].

We have made an effort in separating the motion control of the mechanical arm from the control of the peripheral devices. This will enable us to perceive

Pre-print, HoloMAS2009



(a) Centralised robot control system in vendor controller.



(b) Distributed holonic robot control system.

Fig. 3. Traditional integrated and holonic robot systems in automation. Holons are shown in blue colours and controllers and physical devices are shown in red colours.

the real robot, i.e. without tools, external axes, etc., as a holonic device, giving a higher true holonic granularity in the holonic system. This is illustrated in Fig. 3(b) where the holons access their physical controllers directly, and where the device controllers are allowed peer-to-peer interaction for hard real time performance.

The robots locations, motion capabilities, and work space envelopes are of high importance to the order and product holons. Even for a homogeneous set of robots with the same product capabilities, a slight difference in location relative to the process location, may give a major impact in process performance. This will be even more expressed in a system of heterogeneous robots. These relations may not be a disadvantage, but rather an advantage, especially if the batch layouts and the scheduler functionalities allow for mixing on-line part and painting carrier types.

We are planning to have two different robot systems in the initial laboratory system. One is the system around the central upload robot and another system around the less described arrangement robot. The arrangement station is where bin picking is taking place, picking single parts out of gross storage carriers and placing them structured to order on local storage carriers, possibly directly on top of AGVs.

The Parts Arrangement Station is the location where parts are taken from gross storage containers and transformed into a pickable arrangement in a local storage carrier. The whole station, once robotised with bin picking capabilities, will be in itself a complex holonic system, if the supplier of the bin picking software and hardware allows it. It will definitely be so, in case we endeavour to implement it in-house.

For the initial phases, however, we must envision the absence of robotisation of this station, thus leaving the bin picking a manual operation. This is by no means in contradiction with the holonic manufacturing thought. On the contrary, one of the forces of the HMS paradigm is that an operator need not be treated or represented differently from any other resource or staff holon. The physical interaction with an operator is slightly different, though the capabilities should not be inferior to that of a robotised processing entity.

The direct interaction of this station with the paint shop system is to the orders and schedulers. Product holons for parts play a role in identifying methods for parts, and possible handling process information. Product holons for the transport carriers play another role of specifying the arrangement of parts into the carrier and possibly some process information for installation of the parts.

The Robot Tools are mountable on the robot arm and customised to handle specific parts. The entire set of tools available to one or more robots, or other handling device, must cover the whole range of parts to be handled by the robots. In case of multiple tools, they must be organised in a tool rack, to be on-line changeable by the robot system. There will be a tool holon for each tool, but typically only the ones mounted on a robot at any given time will be active.

Fixtures, be it active or passive, may be considered as tools as well, and need not be at a fixed or active location all the time. In that sense, they are no different from what we normally refer to as tools; tools just just differ by being mountable on a robot end effector.

The tool set available to a robot system is of high importance to the product and order holons. A given order holon needs to be able to request from the robot system that it use a certain tool or tool capability, namely one that is compatible with one of the process alternatives in the pertinent product holon.

The Vision Systems planned for the initial laboratory system are comprised by the following four quite different types:

Painting Carrier Vision System: This is the 3D vision system already implemented with Scorpion 3D from Tordivel. It analyses for the points and directions of interest for a empty painting carrier presented to its cameras. This information is used by the upload robot system.

Part Localiser Vision System: Localises the parts for picking positions and orientations in the transport carriers (possibly the AGVs) in the local part storage area at the upload robot system. It is not to be considered a bin picking system, since the parts are at least semi-structured in the transport carriers.

Passage Vision System: The vision system used to guide the AGVs to cross intersection(s) of the transport route(s) with the overhead conveyor system. It is quite simply a matter of determining, in advance, when there will be passage where, and for how long.

Bin Picking Vision System: Used for identifying parts lying unstructured in gross storage carriers or containers.

This list of vision systems disregards the various dedicated and specialised AGV location vision systems, which are considered private to the AGV system.

The integration of the vision systems into holonic vision applications is a matter of letting orders interact with them in various ways, such as reserving for usage, configuring for detailed applications, and querying for or subscribing to information. But most vision system usage is expected to take place on a lower holonic level, integrated in a more numerical and detailed way with the direct consumers of the information, such as robot systems and AGVs.

The Conveyor System As mentioned earlier, we are not yet sure what type of overhead conveyor system we will install, hence we know little about the controller capabilities and properties. The overall conveyor controller interaction will, however, be controlled by a holon.

In case of a PnF conveyor it will be relevant to find out if the PnF stations can be directly controlled, and thus be integrated with a physical device holon. Otherwise a logical device holon must be set up to access the main conveyor controller. If tracking of the trolleys is possible, it may be relevant to model each trolley as a resource holon. However passive, such trolley holons will at least be

responsive to queries regarding arrival time, as well as associated painting carrier and order.

The AGV System The holonic AGV system is under development, but is developed as a generic AGV system rather than a custom made system for the laboratory paint shop.

It may well be the case that the AGV system exposes its *products*, i.e. the transport and part carrier capabilities and capacities. This is a necessary information to access for the orders and schedulers in the paint shop system. There may be various alternative procedures for creating transport and configuration orders in the AGV system from the paint shop system. One is to send transportation requests to one or more AGV centrals, receiving an offer after negotiations internal to the AGV system. Another, lower level alternative is to directly create a transport order in the paint shop system and then have the pertinent paint shop order holon interact with relevant individual AGVs and AGV system schedulers, to settle a suitable contract. Both alternatives should be offered, since they may have their strengths and weaknesses in different situations.

3.2 Order Holons

The orders in the system contain the static data relevant for the associated order holons. We think of the orders as a hierarchy where the highest level is a production plan, being a quite simple list of parts for production during the day. Typically, in manufacturing systems today, the production plan for the day has been determined over night, or some days in advance, by the company ERP system. In a batch oriented system such as a paint shop, the sequencing and scheduling of the production plan is done heuristically by the operators at the paint shop. All parts produced in the painting systems we know of, are produced for intermediate storage, so the parts produced are used no earlier than the following day.

An illustration of a possible division of orders for our laboratory case is shown in Fig. 4.

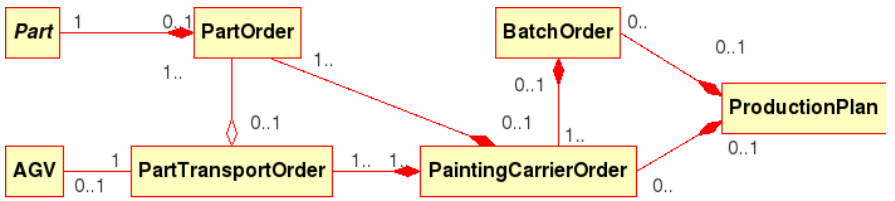


Fig. 4. Overview of aggregations and compositions among the order holon classes.

Pre-print, HoloMAS2009

3.3 Product Holons

The product holons for the upload station must contain a specification of how to attach a compatible set of parts onto a given painting carrier type. If similarity of parts and optimality of the painting process allow for it, certain painting carrier types will be able to host a range of part types. This matching is exemplified in Fig. 5.

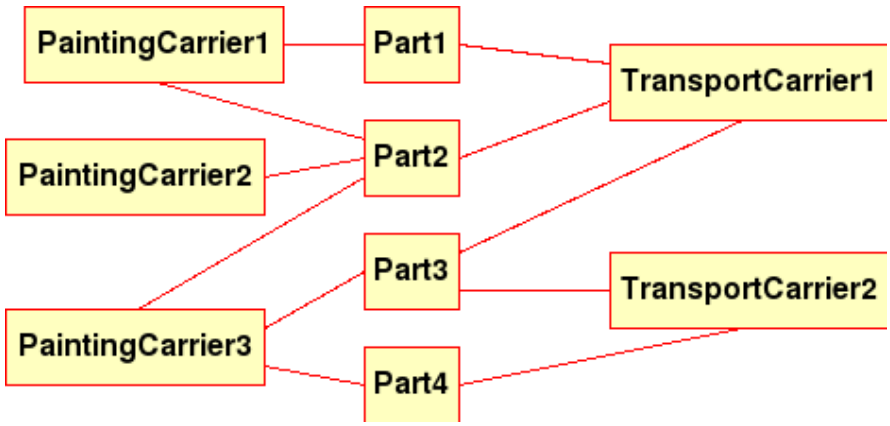


Fig. 5. Illustration of a match between painting and transport carrier types and part types in a simple example with four part types and three carrier types.

The identical problem of matching of carriers and parts arises for transportation purposes. Shown also in Fig. 5 is an example of a transport carrier type to part type match.

In the cases described, the product holons are very static and persistent structures. This will be the case with predefined mixing of part types on the same carrier type. But to be truly flexible, the mixing should be possible to decide for in an on-line manner, i.e. some scheduling functionality may have an on-line or real time interaction with the uploader system. In this case some volatile product holons will be configured ad hoc for a single painting or transport carrier, or configured for the sake of some number of carriers; e.g. for mixing and completing a set of batches.

3.4 Staff Holons

Of staff holons we most certainly will have schedulers. In fact, one of the first overall control mechanisms we should experiment with, may be to emulate the decision system of the responsible operator in the real paint shops. The responsible paint shop operator has a fundamental input the painting day plan from the company ERP system. He makes on-line scheduling decisions based on a variety of parameters and feelings, which we should make an effort in investigating.

Pre-print, HoloMAS2009

In addition to the vaguely interacting staff holons from PROSA[2] we embrace the more directly interacting supervisor holons from ADACOR[5]. We believe both aspects are good to have in separate entity types, whereas Leitão and Restivo[5] seems to suggest that the supervisor holon is more than a staff holon, and replaces it. The staff holons hold the responsibility for planning, long term scheduling, and global optimisation. And the supervisor holon, under normal system operation conditions, should have responsibility for and control of more local aspects of short term scheduling, optimisation, holon structures, and orchestration.

The distinguishing difference between staff and supervisor holons, in our interpretation, is that staff holons are available as a service and help to other holons, whereas decisions by supervisor holons have to be respected by other holons. Thus staff holons can be thought of as finding solutions to problems and supervisor holons to be executors and enforcers of such solutions. As per design of the supervisor holons by Leitão and Restivo[5], they lose their powers when disturbances arise, regaining them again when it is possible to restore local order. These views are not in contradiction with the PROSA and the ADACOR architectures.

4 Discussion and Future Work

While we will proceed with implementing the holonic control and management system in the near future, further development and implementation of hardware and device specific control software will take place in parallel.

One might say that we are still only in the analysis phase of developing the holonic manufacturing system for our laboratory prototype. We feel well into that phase, and have begun considering high level design issues. One pressing issue that we currently have much emphasis on is choosing middleware and platform.

Inspired by such works as Shin et al.[9], one viable path to follow is to start without a platform and use CORBA as middleware. This will enable us to progress fast by implementing our own platform using the Python Programming Language. Python is a language and platform with which we have good experience regarding both the management and soft real time domains of software. The direct implementation of the holons based on CORBA references invocations does not really support a fully flexible holonic platform. However, the platform we develop can be revised into a FIPA compliant one, adding complex communication abilities where and when it is needed.

Another path to follow might be more traditional, like the applications of ADACOR[5], where Leitão and Restivo use the FIPA compliant, Java based JADE agent platform. In order to apply our experience with Python we would use the FIPA compliant, Python based SPADE[10].

5 Concluding Remarks

At the current state of design of the laboratory paint shop prototype, we have gained such confidence in the HMS paradigm, that it will be used for the prototype, and recommended to the industrial partners in the project. The HMS paradigm has turned out to be a major initiator in bringing our minds from the offices towards the laboratory.

References

1. Koestler, A.: The Ghost in the Machine. London : Hutchinson (1967)
2. Brussel, H.V., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P.: Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry **37**(3) (1998) 255–274
3. Leitão, P.: An Agile and Adaptive Holonic Architecture for Manufacturing Control. PhD thesis, Department of Electrotechnical Engineering, Polytechnic Institute of Bragança (January 2004)
4. Leitão, P., Restivo, F.: ADACOR: A holonic architecture for agile and adaptive manufacturing control. Computers in Industry **57** (September 2005) 121–130
5. Leitão, P., Restivo, F.: Implementation of a Holonic Control System in a Flexible Manufacturing System. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews **38**(5) (September 2008) 699–709
6. Mařík, V., Fletcher, M., Pechouček, M.: Holons & Agents: Recent Developments and Mutual Impacts. In: Multi-Agent Systems and Applications II. Volume 2322 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (January 2002) 89–106
7. Lien, T.K., Welo, Nyen, Schütz, et al.: WP1: Technology and Trend Monitoring, White Papers. SINTEF Technology and Society (August 2007)
8. Kay, M.G.: Material Handling Equipment Taxonomy (1999)
9. Shin, J., Park, S., Ju, C., Cho, H.: CORBA-based integration framework for distributed shop floor control. Computers & Industrial Engineering **45**(3) (October 2003) 457–474
10. Gregori, M.E., Camará, J.P., Bada, G.A.: A Jabber-based Multi-Agent System Platform. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS06), New York, NY, USA, ACM (May 2006) 1282–1284

4.3 Development of a Holonic Free-Roaming AGV System for Part Manufacturing

Olivier Roulet-Dubonnet, Morten Lind, Lars Gellein, Per Nyen, Terje Lien, and Amund Skavhaug. Development of a Holonic Free-Roaming AGV System for Part Manufacturing. In Vladimír Marík, Thomas Strasser, and Alois Zoitl, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 5696 of *Lecture Notes in Computer Science*, pages 215–224. Springer Berlin/Heidelberg, September 2009. ISBN 978-3-642-03666-8. doi: 10.1007/978-3-642-03668-2_21

Declaration of co-authorship

The initial project conception for the laboratory demonstration is credited to Terje Lien, Per Åge Nyen, Lars Tore Gellein, and Pål Ystgaard. Per Åge Nyen made the initial process and simulation model for the uploading station; with the central operation of the laboratory demonstration system.

The principles and rough design of the AGV was conceived by Morten Lind. The prototype AGV was built and equipped with power and logic electronics by Stefano Pedemonte; an exchange student from Politecnico di Milano. The proof-of-concept systems for motion control and low-level network communication was developed and implemented by Morten Lind. The AGV localization principle based on ceiling-mounted cameras and AGV-mounted LEDs was analysed and specified by Morten Lind. A simple vision-based localization system based on these principles was implemented and integrated with the motion control system in close cooperation between Stefano Pedemonte and Morten Lind.

The paper describes the ensuing conceptual analysis and design for the distributed control of such AGVs, i.e. the AGV system, within a production systems. Most of the concepts were developed in close cooperation between Olivier Roulet-Dubonnet and Morten Lind. The detailing and description of the concepts, and the entire writing, as well as preparation and submission, of the paper is credited to Olivier Roulet-Dubonnet. Morten Lind made some contribution in reviewing the final version of the paper.

Development of a Holonic Free-Roaming AGV System for Part Manufacturing

Olivier Roulet-Dubonnet[1], Morten Lind[1], Lars Tore Gellein[3], Per Åge Nyen[3], Terje Lien[1], and Amund Skavhaug[2]

[1] Department of Production and Quality Engineering, The Norwegian University of Science and Technology
[2] Department of Engineering Cybernetics, The Norwegian University of Science and Technology
[3] Department of Production Engineering, SINTEF Technology and Society

Abstract. This paper presents an Automated Guided Vehicle (AGV) system under development and its industrial background as a support system for an automated paint department. The focus is on demonstrating how the holonic architecture can be used to implement a flexible AGV system. The system is composed of autonomous AGV holons who cooperate, directly or as groups, with other holons such as robot holons, vision-system holons and order holons to produce the real parts. The holonic architecture is described in detail and example use-cases presented.

1 Introduction

A niche for small and medium enterprises is to produce individually tailored products or services. If the production is done on a large scale, this is called mass customization[1, 2] and requires highly flexible automated production systems. The new generation of free-roaming Automated Guided Vehicles (AGV) can play an important role in this pursuit for flexible automation. AGVs are ground robots, usually used for transportation purposes on the manufacturing shop-floor. Taking fully advantage of the free-roaming AGVs require an integration into the manufacturing systems and a flexibility that the hierarchic, sequential systems currently implemented on most shop-floors do not offer.

Holonic manufacturing[3] is a promising architecture for the development of the new kind of flexible manufacturing systems needed. Holonic manufacturing is a highly distributed control paradigm that promises to handle frequent changes and disturbances successfully[4]. It combines features of both heterarchic and hierarchic organizational structures and is based on the concept of autonomous cooperating agents, called 'holons'. Holonic manufacturing is related to multi-agent systems (MAS), a paradigm derived from the distributed artificial intelligence field, but has some key differences[5]:

- Holonic manufacturing is a concept while MAS is a concept and a technology. It is possible to implement holonic manufacturing using MAS technology.

Pre-print, HoloMAS2009

- A holon can represent simultaneously a whole and a part of the whole: a holon can be composed of several lower level holons.
- Some holons per definition represent and contain physical devices while agents are normally software components.

AGV systems are, by nature, distributed and, as such, we see the holonic paradigm as a good candidate for the architecture of AGV systems. By AGV system we mean one or more physical AGVs and the control and communication software that enable them to receive orders, schedule and execute them.

A systematic review of published papers on material handling in manufacturing systems was performed, with a focus on holonic systems. There are many publications on holonic manufacturing systems but only a few concentrate on material handling. In one of the first papers on the subject, Liu[6] proposes a distributed holonic architecture with one staff holon, where all service requests are sent to and handled by the AGVs themselves. Srivastava[7] presents an approach for conflict-free shortest path, minimum time motion planning and deadlock avoidance for AGV systems. It also presents an architecture for AGV systems which is influenced by its focus on zone algorithms. The architecture is partially reused in our research project. Babiceanu, in his PhD thesis[8], proposes a holonic-based control system for automated material handling systems. His thesis focuses on scheduling and he shows that the results, obtained by running the holonic algorithms, are close to the optimal solution. Most publications on transport systems focus on AGVs running on fast tracks with fixed crossing nodes and empty pathways[6, 7, 9]. They do not investigate the case of free-roaming AGVs driving in a stochastic environment together with other vehicles, manufacturing artefacts and humans.

This paper presents an Automated Guided Vehicle (AGV) system under development and its industrial background. The focus is on demonstrating how the holonic architecture can be used to implement a flexible AGV system. The system is composed of autonomous AGV holons who cooperate, directly or as groups, with other holons such as robot holons, visions-system holons and order holons to produce the real parts. The holonic architecture is described and example use-cases presented.

The remainder of this paper is organized as follow: Section 2.1 presents the industrial test case for the AGV system. Section 3 describes the holonic architecture of the system. Section 4 considers some use-cases showing how the holonic architecture can support the required flexibility. Finally, section 5 presents discussion and conclusion.

2 Industrial Application and Physical Devices

2.1 Industrial Background of the AGV System

The industrial background of the AGV system is two medium sized enterprises producing consumer goods. Their manufacturing process is a typical example of

mass customization: they manufacture individually tailored products on a large scale.

This paper is written as part of a larger project which focuses on the automation of the paint department and its surrounding logistic processes. In an earlier work the current paint process was analyzed and a new automated paint solution was proposed. The new automated paint-shop department and its logical control has been thoroughly documented. The result of this work serves as a reference system to be used for the further development of the proposed holonic AGV system.

Figure 1 shows the proposed system simulated in QUEST, a commercial simulation program. The heavy yellow wires are the tracks of the overhead conveyor which transports the painting carriers past the upload cell, the download cell, and through the paint process. All the red boxes make up the devices for the paint process, such as washing, drying, painting, hardening. The four robots are grouped in an upload cell and a download cell. In the upload cell one robot grips parts from bin and places them on an AGV, the other grips part from the AGV and hangs them on the conveyor belt. The download cell has not been setup yet but follows the same principles.

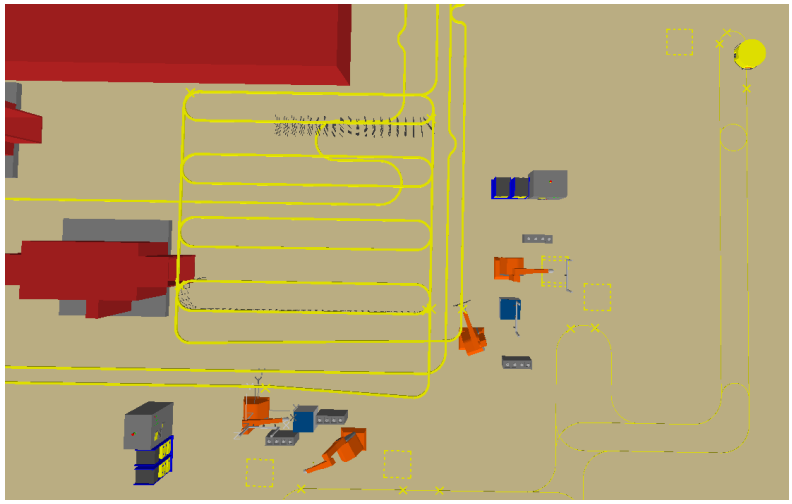


Fig. 1. The uploading(right side) and downloading(bottom side) cells of the paint department in QUEST

The role of the AGV system in the paint process is to perform the resupply of components for upload, and the transfer of downloaded components. The AGV system will also perform the transport tasks inside the cells, in-between the industrial robots, thus allowing for a better utilization of the shop-floor area e.g. by physically splitting the bin-picking operation from the uploading operation. For the industrial partners the AGV system is not viewed as a separate goal but

rather as a support service for the manufacturing cells we are working on and eventually the rest of the manufacturing process.

A concrete challenge in this transport application is that it will not be possible to reserve pathways in all areas where the AGVs will be driving: There are no static paths nor static nodes and the pathways cannot be guaranteed to be free of obstacles. This uncontrollable environment requires advanced obstacle avoidance capabilities. This fact, as well as the importance of flexibility in this project, has driven our focus on free-roaming AGVs.

2.2 The Laboratory Demonstrator

To demonstrate the feasibility of the proposed paint process, and to experiment with new technologies, a laboratory demonstrator was setup. The functionality contained within the upload cell together with the corresponding resupply of components constitutes the basis for the laboratory demonstrator.

The implemented AGV part of the laboratory demonstrator consists of a working AGV that communicates with a vision system used for accurate positioning within the laboratory. The system is in the process of being extended: two AGVs are being assembled and several more are planned. In addition, several vision systems are being set up. By design, the AGV fleet will be heterogeneous to be able to experiment with AGV-dependent optimization for the manufacturing system.

The first test case is to transport components within the uploading cell. A bin-picker robot picks parts from a container and organise them on an AGV. When the task is accomplished the AGV drives to an upload robot where components are uploaded to the power-and-free conveyor to get painted (cf. Fig.2).

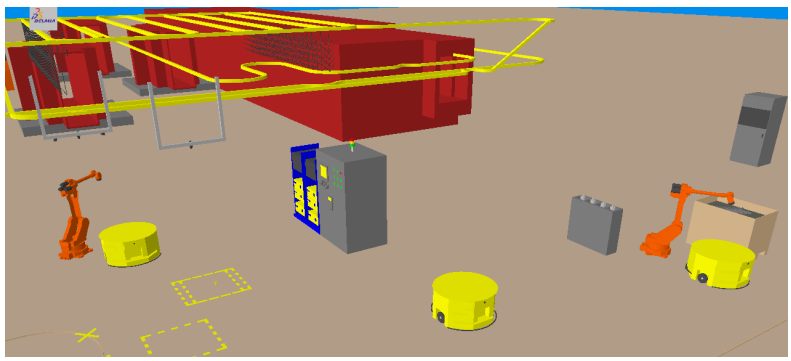


Fig. 2. The split upload cell using AGVs for transport between the two robots

Pre-print, HoloMAS2009

3 The Holonic AGV System for Part Manufacturing

AGV systems are, by nature, distributed and flexible. When they, in addition, are composed of free-roaming AGVs, with local navigation ability, they match physically the definition of autonomous holons in the holonic manufacturing concept. As such we see holonic manufacturing as a natural choice for the control architecture of AGV systems.

Compared to a more traditional architecture the holonic architecture offers off-line and on-line flexibility that makes it possible to implement the required adaptability of the system, thus exploiting the possibilities offered by free roaming AGVs. The off-line structural flexibility of the holonic architecture enables the implementation of features of heterarchic and hierarchic organization[4]. In addition, holonic manufacturing has a built-in concept of on-line aggregation of holons into a new holon, that can be used to represent the physical, temporary association of holons to a specific task (cf. Sec.4).

The AGV system has been designed based on the published literature on holonic manufacturing and material handling system. The system is an implementation of the PROSA architecture[4] and is in several aspects inspired by the work done by S.C. Srivastavaet al[7] and Babiceanu et al[10].

Scalability in holonic material handling systems is a central issue which has been little mentioned in publications. In the design process of our proposed AGV system we have taken scalability issues into account from the beginning. We have used a distributed architecture and have avoided communication bottlenecks and central servers where practical.

3.1 The Holons

The proposed AGV system is composed of the four base holons defined in the PROSA architecture.

- Resource holons represent the physical devices. In our cases resource holons are AGVs or groups of AGVs.
- Product holons represent the part to manufacture. In our case they are containers for information about the parts to transport.
- Order holons are generated for each transport request. They are the driving force of the holonic system. They compete or collaborate with each other to allocate AGVs and process their transport need.
- Staff holons are adviser holons. They do not execute actions directly but give advices to other holons. A typical example is the scheduling holon which generates an 'optimal' schedule and advice it to other holons.

A simplified overview of the architecture is shown in Fig.3. Resource holons are represented in rectangles, order holons in ovals and staff holons in hexagons. It shows independent AGV holons and AGV holons aggregated in an AGV-group. It also shows one path-planning holon inside an AGV holon, although this is not a requirement: path-planning holons can be part of AGV holons or shared.

Pre-print, HoloMAS2009

The main components of the AGV system have been split into autonomous entities: low-level AGV control, traffic, scheduling, path-planning, ordering, localisation. Not all components need to be implemented as holons. Entities can be implemented as separate processes, on servers or AGVs, but as soon as they can deliver useful information to other entities it makes sense to implement them as holons. Some entities, like ordering, are implemented using many order holons.

It is our experience that industrial partners often desire central algorithms for critical functions like scheduling. A possible implementation of such critical functions is to use a unique central holon or several holons using the same algorithm. The algorithm must have a deterministic result so that holons eventually will end up with the same conclusions. This requires a system resilience to temporary disagreements. Resilience can be implemented in a holonic system by using staff holons giving advices, and not orders to other holons. In the disagreement period, the obtained result might not be optimal, but we think that the added flexibility and scalability of the solution overweight this issue.

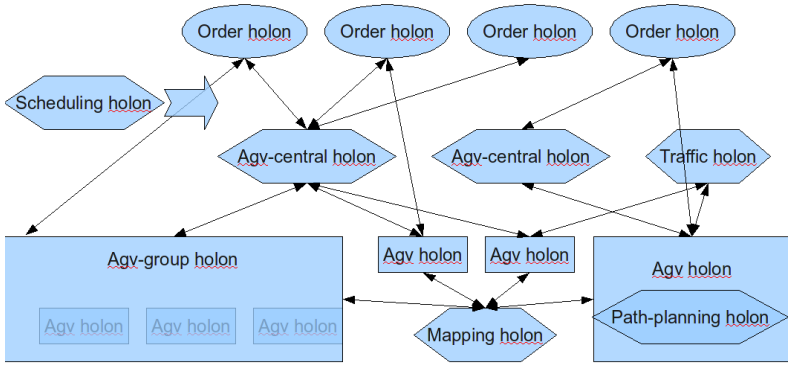


Fig. 3. Simplified overview of the AGV system control architecture

A more detailed description of the main holon types is presented in the following subsections.

The AGV Holon. The AGV holon is the resource holon controlling the physical AGV and as such it runs directly on the physical AGV. In our system the AGVs are required to be able to run holons and communicate through a wireless network.

An AGV holon has a queue of order holon tasks which are periodically rescheduled. To support AGV-groups and special situations within the manufacturing cell AGV holons can enter a slave-state, thus letting another AGV holon or holon from the manufacturing take over the high level physical control of the AGV.

Pre-print, HoloMAS2009

The Order Holon. The order holon is the specialized order holon for the AGV system. An order holon is created by a generic manufacturing order holon to solve a transport issue.

Order holons are the driving force of the holonic system. As long as its transport request is not executing, an order holon constantly queries the AGV-central holons for available AGVs and expected scheduling data. If it gets a better offer, or does not yet have an AGV assigned, it contacts the candidate AGV holon and request to be appended to its order queue.

The order holon also periodically contacts its assigned AGV to keep each-other aware of their current state. If an AGV holon does not get information from a order holon for a period of time, it removes it from its transport queue.

The AGV Central Holon. The AGV-central holon is a staff holon. Its name is inspired by an analogy to a 'taxi central'. The role of the AGV-central holons is to increase scalability by providing an interface to AGV holons. Each AGV uses one AGV-central holon chosen using, for example, geographic parameters.

AGV-central holons answer queries from order holons and advice them to an AGV. Order holons call several AGV-central holons and choose the best offer.

The Group Holon Group holons are created when several AGVs link themselves together to a special task. The group holon is an abstract holon type inherited by specialized group holons cf. Sect.4.1. In some cases it may appear as one AGV holon to other holons.

The Scheduling Holon. The scheduling holon is a staff holon providing scheduling data to other holons. These holons are a necessary complement to local scheduling information from the AGV holon, since AGV holons have a limited vision of the global manufacturing system. The scheduling holons give advices to order holons or AGV-central holon to influence the choice of AGVs by order holons.

By communicating with other holons it keeps an up-to-date view of the manufacturing system, generates an 'optimal' scheduling and tries to get it applied by trying to re-affect order holons to AGVs.

The Path Planning Holon. The path-planning holon is a staff holon. It is a holon computing an optimal transport path for an AGV. It needs a map of the shop-floor obtained through the mapping holon and optional information from the online-traffic-control holon. Since the AGVs are capable of local navigation, a path is, in our context, defined as a list of points to be loosely reached. The path-planning holon can either be part of an AGV holon or be shared between several AGV holons.

The Mapping Holon. This is a staff holon that collects mapping information from all running AGVs in a geographic area. Mapping is a crucial feature of

AGV systems for localization, navigation and path planning. Some of the AGVs are using Self Localization And Mapping algorithms, thus they depend on an up-to-date map to know their current position.

The idea is that all the AGVs participate in the on-line mapping of the shop-floor, thus reducing the need for manual updating of the current map. AGVs send periodically a report on their modification to their internal map, thus informing other AGVs of obstacles and changes on the shop-floor.

This system can used as the basis for the implementation of specialized scout AGVs that constantly investigate the shop-floor and maintain an up-to-date map. Unused AGVs could also be used to investigate unknown areas.

The Online Traffic Controller Holon. This is a staff holon whose role is to survey the AGV traffic in a geographical area and give advices to AGV holons to sort out conflicts and avoid congestions.

4 Holonic Use-Cases

This section details some use-cases as they have been defined for the AGV system. They focus on showing how the holonic architecture can be used to increase the flexibility of the AGV system.

4.1 AGV Holons as One AGV Holon

By this cryptic name we mean applying the holonic fractal concept to link together several AGVs to a special task. A group AGV can appear to the external holonic environment as a specialized AGV holon. The constituting AGV holons do not reply to requests as individuals but through the group holon. Examples of such applications are:

- Replace a palette conveyor with several AGVs: Several AGV holons enter a special 'pallet' state and one AGV holon start acting as a pallet conveyor supervisor. Required specialized holons are started. e.g. scheduling.
- Create a transport chain for an transport order with high priority.
- Transport large objects. The AGVs can link together their lower-level controllers thus allowing on AGV holon to control directly all physical AGVs. The AGVs can even be linked together mechanically.

4.2 AGV Holon as Part of a Manufacturing Cell Holon

A fundamental concept of our system is that AGVs are an integrated part of the manufacturing system. AGV holons can be assigned to a cell holon or, even, a robot holon for a given time.

- AGVs can be assigned to a robot system, thus allowing direct ordering from the robot to the AGV, effectively bypassing all scheduling and negotiations with other holons. It can be used, for example, to implement efficient cooperation between two robots out of each others reach: when a robot is finished placing parts on the AGV it communicates directly to the AGV to send it to next robot.
- In a similar idea AGVs can be used to increase the reach of an industrial robot. Thus allowing a smaller robot to temporary accomplish tasks usually requiring bigger robots.
- AGVs could also be used to push objects in a cell if required.
- AGVs can be affiliated to a robot on a track. An example taken from our paint conveyor: If instead of using a power-and-free conveyor for the hangers we would use a simple trolley conveyor and an industrial robot on rail. We could then follow the robot with the AGV while it is hanging parts on the conveyor, thus avoiding travel back and forth for the robot to pick parts. This solution would require a precise positioning system for the AGV or/and a vision system to give real-time part positions to the industrial robot.

4.3 The AGV as a Self Contained AGV System

In this concept a physical AGV represents the whole AGV System: the entire AGV system is started by starting one AGV. This feature is desired to simplify the administration of the system and to solve some specific scenarios e.g. transport to external storages areas outside the wireless network.

An AGV holon has a list of holons to connect, e.g. path-planning holon and AGV-central holons. Some of them are optional and others are required. If a required holon is not found then it is started either on the local AGV or on an available server.

A consequence of allowing holons to start other required holons, is that a mechanism is required to handle 'excessive' holons of one type. This is typical issue of distributed systems, holons of an excessive type need a consensus on who will close itself.

5 Discussion and Conclusion

In this paper we have presented the industrial background of an AGV system under development. The need for a flexible solution has been identified and a control architecture using the holonic manufacturing paradigm proposed. The paper has also presented, through several use-cases, how the holonic architecture can support the required flexibility of the AGV system.

In our proposed system there is clearly an emphases on the process viewpoint as defined in the reference PROSA architecture: It has been a conscious choice to focus on the AGV system as a support component of the manufacturing cell. As a consequence, the described use-cases are concerned with the interactions between the AGVs and the manufacturing cell and much less with high level

scheduling and traffic management. One might say that AGV system presented only partially defines a material handling solution but we see this system as working subsystem from which a larger system can be composed, an important characteristic of holonic manufacturing systems[11].

6 Future Work

Implementation of the proposed holonic AGV system will continue during the next years in cooperation with the industrial partners. Much work has yet to be done to evaluate the performance, scalability and flexibility of the proposed holonic system and to demonstrate the industrial feasibility of the proposed use-cases.

7 Acknowledgements

This paper presents research results obtained through work in the IntelliFeed and CRI Norman projects sponsored by The Research Council of Norway.

References

1. Zipkin, P.: The Limits of Mass Customization. Harvard Business Review **75** (1997) 91–101
2. Davis, S.: Future Perfect. Reading, Massachusetts/USA (1987)
3. Van Brussel, H.: Holonic Manufacturing Systems The Vision Matching the Problem. In: Proceedings of the 1 stEuropean Conference on Holonic Manufacturing Systems, Hannover, Germany, IFW-Hannover. (1994)
4. Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P.: Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry **37**(3) (1998) 255–274
5. Leitão, P.: Agent-based distributed manufacturing control: A state-of-the-art survey. Engineering Applications of Artificial Intelligence (2008)
6. Liu, S., Gruver, W., Kotak, D., Bardi, S.: Holonic manufacturing system for distributed control of automated guided vehicles. In: Systems, Man, and Cybernetics, 2000 IEEE International Conference on. Volume 3. (2000)
7. Srivastava, S., Choudhary, A., Kumar, S., Tiwari, M.: Development of an intelligent agent-based AGV controller for a flexible manufacturing system. The International Journal of Advanced Manufacturing Technology (2007) 1–18
8. Babiceanu, R.: Holonic-based control system for automated material handling systems. PhD thesis (2005)
9. Wallace, A.: Application of AI to AGV control-agent control of AGVs. International Journal of Production Research **39**(4) (2001) 709–726
10. Babiceanu, R., Chen, F., Sturges, R.: Framework for the control of automated material-handling systems using the holonic manufacturing approach. International Journal of Production Research **42**(17) (2004) 3551–3564
11. Valckenaers, P., Brussel, H.V.: Fundamental insights into holonic systems design. In: HoloMAS. (2005) 11–22

4.4 Open Real-Time Robot Controller Framework

Morten Lind, Johannes Schrimpf, and Thomas Ulleberg. Open Real-Time Robot Controller Framework. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 13–18, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4

Declaration of co-authorship

Experiment setups for and conduction on the three different robot controllers were a collaborative effort among Morten Lind, Johannes Schrimpf, Sebastian Dransfeld, and Thomas Ulleberg. Morten Lind implemented the final versions of all external control code, that was used in conducting the experiments. The experiments were developed and planned in close cooperation between Morten Lind and Johannes Schrimpf.

- Sebastian Dransfeld single-handedly performed all experiment setup and measurements for the KUKA RSI control.
- The Nachi controller setup was enabled by the efforts of Johannes Schrimpf, with help from Thomas Ulleberg, to intercept and modify the internal Nachi controller communication. The measurement experiments on the Nachi controller were carried out in collaboration between Morten Lind, Johannes Schrimpf, and Thomas Ulleberg.
- The gateway to the low-level control interface in the Universal Robots controller was designed and tested in collaboration among Johannes Schrimpf and Morten Lind, and was implemented by Johannes Schrimpf on the native controller platform. The experiments with the Universal Robots controller were conducted by Morten Lind and Johannes Schrimpf in collaboration.

The PyMoCo control framework was conceived and designed by Morten Lind, with a cooperative contribution from Johannes Schrimpf. PyMoCo was implemented single-handedly by Morten Lind, and tested extensively in close cooperation among Morten Lind and Johannes Schrimpf.

The paper was written, prepared, and submitted by Morten Lind. Johannes Schrimpf contributed review of the final version. All graphics is designed and produced by Morten Lind.

Open Real-Time Robot Controller Framework

Morten Lind[1,3], Johannes Schrimpf[2,3], Thomas Ulleberg[3]

[1]Norwegian University of Science and Technology, Department of Production and Quality Engineering, Trondheim, Norway
[2]Norwegian University of Science and Technology, Department of Engineering Cybernetics, Trondheim, Norway
[3]SINTEF Raufoss Manufacturing AS, Trondheim, Norway

Abstract
The challenge with advanced robot control in manufacturing is two-fold, regarding industrial robot controllers: 1) *General real-time control from external entities are not supported;* and only for special cases of application scenarios, limited real-time extensions to the controller can be purchased. 2) *The robot controller application-platforms are robot centric*; leaving an external application to battle with achieving the desired behaviour.
Based on free and open software resources, experiments have been performed with three industrial robot controllers, and measurements of response times and tracking delay from external control are presented. Also presented is the design of a motion control framework, demonstrating external integration of force feedback and visual servoing.

Keywords:
Manufacturing system; Real-time control; Robot motion control; Robot sensor-servoing

1 INTRODUCTION

The past couple of decades have seen an ever increasing demand for flexibility and adaptability in manufacturing automation. Regarding the near future of manufacturing in western countries, a quite probable scenario is that manufacturing of simple goods with no or little variation, will be almost non-existing. The manufacturing industry that will remain in this part of the world will have emphasis on a high degree of customization, almost to the level of having no product catalogue.

One of the drivers for this effect is the market demands, requesting customization and personalization, simply due to the possibility [1]. This is a stimulative and additive effect, changing existing manufacturing companies and shaping new ones. Another driver for this is the outsourcing or relocation of uncomplicated large-series production to low-cost countries. This latter effect is a subtractive and inhibitory effect in the sense that it removes manufacturing companies that do, or can, not change, and prevents establishment of new manufacturing companies that only manufacture simple goods.

These two effects are, of course, but two among a whole range of other effects, and can not encompass the plethora of aspects and types of manufacturing. However, in the general subject of manufacturing research, they are the predominant effects discussed regarding automation.

Automation is well in the process of taking over shop-floor level activities, like processing, handling, and transportation. Factory level activity, like orchestration, real-time (re-)scheduling, and online logistics management, is under development to be automated, hence closing the gap between *Enterprise Resource Planning* (ERP) systems and shop-floor control.

Motivation

Robot manipulators are used to meet the requirements of agility, reachability, flexibility, adaptability, dexterity, etc., in manufacturing systems. The most flexible kind of industrial manipulator is the (serially linked) articulate robot, and it mostly has 6 degrees of freedom (DOF). A challenge with such mechanisms is, that the mapping between the actuator and operational spaces are highly non-linear. This is why an advanced motion controller is always found associated with such a robot.

Historically, there has not been an overall application control at the factory or shop-floor levels, so robot-centric application controllers were implemented co-located with the motion controllers in the robot controllers. These have evolved to quite advanced platforms, but typically remain closed and proprietary, shielding off the underlying servo controller from the application programmer. Hence, the native application controllers are well suited for the use cases that were part of the platform developers' design criteria, but virtually excludes or hinders all other uses and application scenarios.

To render the robots more general and generic, the world of robotics has seen some projects aimed at developing open controllers, independent of the robot and controller manufacturers. Examples of such are the *Open Modular Controller*, developed by a team led by Jensen [2], and the *OROCOS* project [3]. Such projects provide an advanced application platform, which is completely open, and thus allowing any application scenario within the limits of mechanics, hardware, and real-time communication.

Facing the need for application flexibility and factory-wide automated control, the robot controllers are no longer adequate as application platform. Further, the open application-platform controllers may also be too complex, since they remain robot centric. I.e. they still assume that it is within the application controller of the robot, that the major part of the application is to be implemented.

In a distributed, intelligent system for automatic control at factory or shop-floor level, the application platform is in "the sky"; i.e. in the local network in the factory. Such a control system, e.g. a *Holonic Manufacturing System* [4], will benefit from soft real-time access to motion- or servo-control. Local application scenarios, like sensor-servo-based motion control, will need semi-hard real-time access to the servo-controller [5].

Related Work

Real-time external motion-control of native controllers for industrial manipulators is not a new phenomenon. In special applications, where the application platform in the native controller is inadequate, or where the real-time application control is already implemented on another computer platform, there hardly exists any viable alternative.

Cederberg et al. [6] mention $10Hz$ interaction frequency with an ABB IRB 2400/16 robot with an S4CPlus controller, through the native application controller. They use a combination of a RAPID program running in the controller and an external program using RAP to communicate from the external program. The tracking delay is not quantified, but judging from the programs presented, and by experience with the S4CPlus controller, it seems realistic to guess at no less than $500ms$. This is an example of real-time control through the application platform in the commercial controller, and may be classified as a gentle technique for circumventing the native application platform.

Wetterwald et al. [7] used the KUKA RSI with a KR60L30 HA robot for external motion (correction) control in a sewing application. This is a hybrid approach, since part of the application is implemented in the main controller, whereas the external control performs sensor-based real-time trajectory corrections. In their experiments, the robot motion could have been controlled freely over RSI, but the KUKA application platform was chosen for part of the entire application; somewhat due to historical reasons in the project.

Bigras et al. [8] implements a force-control loop around the operational space position-control over KUKA RSI with a KUKA KR210 robot. A central part of their work is impedance modelling of the robot joints and surroundings. This demonstrates a quite advanced control application made possibly by the real-time access to the KUKA controller.

Schnell et al. [9] used an advanced open controller, the *Open Modular Controller* [2], implementing servo-level control and providing a flexible platform for integration on top of a PC-platform. They used an ASEA IRB6/2 robot, and apart from the mechanical arm and servos, only the servo amplifiers were reused. At the lowest level, the controller itself addresses the servo amplifiers through a PMAC controller board. This is an example of a demanding effort for recycling old robots, completely modernizing their control system. An outdated application platform is replaced by a new, open, and advanced application platform.

A simple and most elegant external real-time control of an industrial robot is described by Dallefrate et al. [10]. They used the 7-DOF Mitsubishi PA-10 robot. The PA-10 controller supports direct access to velocity or torque control on the servo controller. Though elegant, it takes some effort to implement a trajectory controller to close a position control loop around either velocity or torque control. The servo controller of the PA-10 is accessible over ARCNET, with the possibility of achieving a control frequency up to $1kHz$. They report an impressively low jitter of less than $4\mu s$, in their specific Linux+RTAI environment.

Paper Outline

The remainder of this paper is in two parts. The first part, in Section 2, presents a simplified, conceptual model of how an industrial robot controller is organized. It is used as basis for the discussion of motion and servo controllers. Experiments with three different robot controllers are discussed and the performance-results are presented. The second part, in Section 3, gives a conceptual overview of the implemented framework for real-time, external motion-control. Finally, some concluding remarks and acknowledgements.

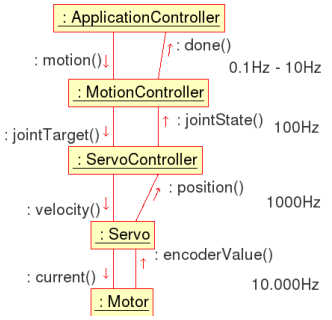


Figure 1: Simplified view of the communication within an industrial robot controller.

2 ROBOT CONTROLLERS

In this part, a simple conceptual model of a standard robot controller is presented. Afterwards, experiences with external connection to three different controllers is shortly described. Finally performance measurements on the three case-controllers are presented and discussed.

2.1 Controller Basics

Figure 1 illustrates a possible, if naive and simplified, communication diagram for the interesting components in a standard type of robot controller. Labels in the figure indicate the order of magnitude for frequency of interaction between the components.

At the top level the application controller handles the logic of the user application, and submits motion segment specifications to the motion controller. The motion controller will execute these by interpolation according to the motion type, and control the servo controller in executing the motion. The servo controller makes a refinement of the interpolated points from the motion controller. These finely spaced position targets are executed by, say, velocity commands to the servos, their synchronization and positions being monitored and controlled. The servos control the motor currents and read back the motor encoder values.

2.2 Robot Experiences

The results in this part of the paper is based on working experience with three different robot controllers, each of which supports external motion control. All communication is over ordinary, unmodified Ethernet, using UDP or TCP connections.

NACHI SC15F

By installation of an embedded single board computer (SBC), transparently intercepting the communication between the internal motion controller and the servo controller. The actual joint positions from the servo controller is sent out in a UDP packet over Ethernet, and is received by the external motion controller. This emission of joint positions are bound to the interpolation period between motion and servo controllers. The SBC further listens for UDP packets from the external motion controller, sending position commands to override the commands from the internal motion controller.

In the native NACHI controller, the internal communication frequency between motion and servo controllers can be configured freely within some range, but defaults to $100Hz$; which is understood to be recommended. The servo controller provides pure joint position control with joint position feedback.

KUKA KR60L30 HA with RSI

Experience with the KUKA RSI control interaction is mainly from a sewing application [7]. KUKA RSI supports external position control over TCP in real-time, either in joint or operational space, with position feedback. The version of RSI used is 2.1 with RSI-XML version 1.1.

The interpolation period of the RSI communication is $12ms$; i.e., a frequency of $83.33Hz$. The external control must be synchronized to the feedback from the RSI controller, with a $4ms$ time window to respond with a new desired position specification.

The possibility of real-time external motion-control in operational space may be a major advantage for companies that do not have the competence for developing or dare commissioning third part, free and open, motion control.

Universal Robots UR-6-85-5-A

The 6-DOF UR-6-85-5-A articulate manipulator from Universal Robots has an internal PC running a GNU/Linux OS for motion and application control. It is very open for access and deployment of software. The servo controller is directly accessible by compiling a "motion controller" program, using an API header file and linking with a library file; both supplied from Universal Robots. On the native controller computer, the high-level controller is then replaced by this new controller.

The controller in the presented work is about 100 lines of C-code, which simply implements an adaptor to the servo controller, exposing it over UDP sockets to an external motion controller. The servo controller sends, at $125Hz$, the actual position and velocity joint-vectors; i.e. a packet containing $(\mathbf{q}^a, \dot{\mathbf{q}}^a)$. In response, it requires the desired position, velocity, and acceleration joint-vectors for the next interpolation period; i.e. a packet containing $(\mathbf{q}^d, \dot{\mathbf{q}}^d, \ddot{\mathbf{q}}^d)$. Alternatively it is possible to control by pure joint velocity, and a future release of the controller software will give access to joint torque control as well.

2.3 Experiments and Performance

The expected performance of an external motion controller application will, naturally, depend heavily on the performance of the underlying servo controller. Specifically it is the response time and the tracking delay which are of interest in real-time sensor-servoing applications.

For the specific experiments presented in this paper, to be fair to KUKA and NACHI, it is imperative to mention here, that no tweaking or optimization of filtering in the servo-controllers was performed. It is possible to change the filtering, and possibly lower both response time and tracking delay.

Response Time and Tracking Delay

Response time is defined as the time from a change is made in the desired motion until an effect can be observed in the actual motion. The *tracking delay* is the amount of time that the actual motion is trailing the desired motion. These quantities are chosen for measurement mainly due to external observability, but also because they are of importance for designing motion control applications.

Robot	**Response** $[ms]$	**Tracking** $[ms]$
NACHI	45	120
KUKA	42	115
UR	12	9

Table 1: Summary of numerical results for response time and tracking delay for the different robots.

Both response time and tracking delay are observed from the external side, and hence they include network transport time. However, the latency in a standard switched local network will be of the order of $200\mu s$, which hardly contributes compared to the interpolation cycle period of around $10ms$.

These quantities are measured by customized small programs that do nothing but addressing the robot servo controller directly over the network. While executing some desired motion, the corresponding times of sending and receiving the positions are logged together with the positions.

Response time is found by commanding the servo controller with a step function, the step being set as high as the pertinent servo controller accepts. Tracking delay is measured on a ramp or a sine motion as the time the actual position of a joint is trailing the desired position.

Measurements

Some selected measurements are displayed by plots of desired and actual joint position vs. time. The desired positions sent to the robot are shown as green crosses connected by green line segments, and the actual positions reported from the robot are shown as blue points connected by blue line segments.

Figure 2 shows plots of sine responses from the robot controllers. The motions shown are generated with 10° amplitude and at a cyclic frequency of $1Hz$. All measurements are moving only the base shoulder joint (joint 0), with the upper arm vertical and the forearm horizontal. The plots show the first 1.5 periods of the motion. The experiments continued for several periods with the same behaviour as observed in the plots.

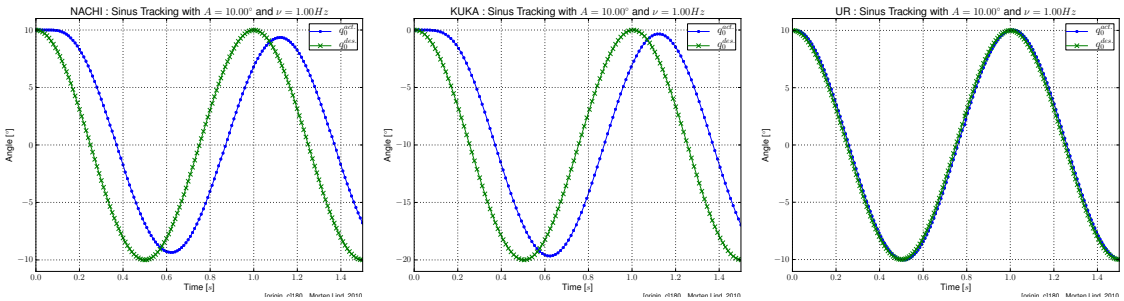
Figure 3 shows plots of step responses from the experiment controllers. For each robot is seen an individual size of the step, which has been experimentally maximized. The maximization is done to ensure as fast and strong a response as possible. Since the KUKA and NACHI servo controller performs filtering of the motion, they accept a much larger step than the unfiltered servo control in the Universal Robots robot. The time axis in the plots have been zeroed to the time where the step is sent.

Measurement series for the sine responses were made on all robots by specialized programs. The step responses are calculated from one single experiment, since it has no parameters. The tracking delay was inspected over a series of experiments where both amplitude and frequency was varied.

Results

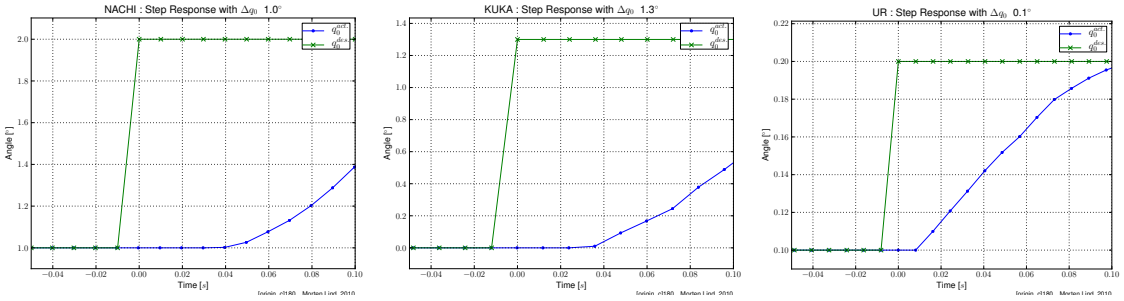
By analysis of Figures 2 and 3 some estimates of the sine- and step-responses for the different robot controllers can be extracted.

Measurement of tracking delay is performed at the steepest position of the desired trajectory as the horizontal shift to the actual trajectory on the sine response curves. The response time is measured as the time passed from the step is sent and until a half interpolation cycle before the first significantly changed interpolation point.



(a) NACHI (b) KUKA with RSI. (c) Universal Robot.

Figure 2: Sine wave tracking by the three experiment robots.



(a) NACHI (b) KUKA with RSI. (c) Universal Robot.

Figure 3: Step responses from the three experiment robots.

The numerical results from the analysis of the plots are summarized in Table 1.

The KUKA and NACHI robots have similar characteristics with just beyond $40ms$ response time and around $120ms$ tracking delay. This is easily understood if it is assumed that their servo controllers have a motion buffer for estimating trajectory parameters, and filter the control to the desired trajectories for the servos.

From experiments with the Universal Robots robot, both response time and tracking delay is measured to around $10ms$, which is around one interpolation period. This observation is consistent with the absence of trajectory parameter estimator and filtering.

3 CONTROLLER FRAMEWORK: PYMOCO

The "space" of motion control is huge, and it may interact with many systems of very different nature. Therefore it is preferable to keep motion control on the most agile platform, providing a vast amount of libraries for computing and communication. This leaves the ultimate flexibility to the application designer regarding implementation method or paradigm, programming language, and platform. This is in strong opposition to the application platform design within contemporary robot controllers.

This part presents a simple framework for motion control that have been developed, tested, and used in applications; however, still to be considered experimental. Since it is entirely implemented in the Python Programming Language, it has been named *PyMoCo*.

An important advantage of having the entire code base in Python is that it should require almost no effort in porting it among any Operation System platform that supports the Python interpreter; e.g. OS X, any Windows OS, or any GNU/Linux distribution. Debian and Ubuntu distributions of GNU/Linux was used for developing, testing, and applying it.

The Natural Level of Separation

It is in the motion control layer that advanced control scenarios with respect to external orchestration or real-time sensor-integration will be relatively easy. One level lower, in the servo controller, things get control theoretically quite involved, and will anyways be rather robot specific regarding dynamics, mechanics, and electronics. Further, below the servo controller level, the control frequency will be very high and jitter tolerance low.

In this light, interfacing to the servo controller from an external motion controller should be considered a natural choice. System and application developers will thus be empowered by the possibility of implementing suitable motion control, while the robot manufacturer takes care of the very robot specific and complex control issues.

Servo Controller Interfaces

For the KUKA and NACHI robots, the trajectory data for the interface is of the same nature: Joint positions are sent to the controller and joint positions are received. The Universal Robots controller requires additional trajectory data: joint-velocities and -accelerations. The data returned from the Universal Robots servo controller is also extended with actual joint velocities.

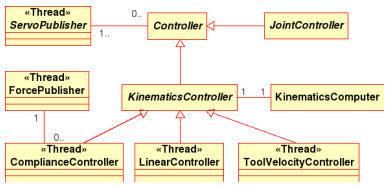


Figure 4: Class diagram of the central classes in the PyMoCo design.

To control the Universal Robots robot with the same interface as the KUKA and NACHI robots, a P(ID)-controller at the interface layer will be implemented; possibly at the cost of some response time and tracking delay. This will enable use of the same motion controllers across all of the robots. However, some motion controllers may in fact take advantage of the explicit control of accelerations and velocities.

3.1 Framework Design

A class diagram of some core classes and some specific controllers in the framework is presented in Figure 4. The central classes are explained in this section.

The controller framework provides two essential classes: The ServoPublisher and the Controller. They are the functional proxies to communicate with the servo controller. Similar to the ServoPublisher class, publisher classes for sensor inputs can be implemented; ForcePublisher is one such sensor publisher. The Controller class, besides from having some fundamental functionality itself, is the base class for the hierarchy of different motion controller classes.

The ServoPublisher *Class*

The *ServoPublisher* supports the publisher-subscriber pattern, calling a special method on each subscriber when there are new state data from the servo controller. The *ServoPublisher* is an abstract base class, and is to be specialize into proxies for the different controllers; e.g. *NACHIServoPublisher*, etc.

The Controller *Class*

The *Controller* base class is an abstract class for mediating the desired motion from an explicit controller to the servo controller. It will need specific implementations for implementing motion control strategies.

Notable functionalities, which reside in the base *Controller* are the ability to address data in correct format to the servo controller; consistency checks on the data; scaling of data into known limits of the servo controller, if so configured; and providing a basic handler of notifications from the *ServoPublisher*.

In specialized base-controllers, i.e. those that differentiate among the different servo controllers, special consistency checks or control aspects can be implemented. An example is the implementation of a basic P(ID)-controller for the Universal Robots controller, *URPositionController*, where position control can be implemented over the raw servo controller interface.

The KinematicsController *Class*

A *KinematicsController* is to be distinguished from a joint-based controller, in that it relates operational space to the joint space of the robot. The controller classes that specialize the abstract class *JointController*, is only relating to the joint space of the robot.

A fundamental component of a *KinematicsController* is a *KinematicsComputer* class. The *KinematicsComputer* is the class that provides the fundamental computation elements for the actual robot, mathematically relating joint and operational spaces. This leaves the *KinematicsController* specializations to focus on motion strategies and application of sensor inputs.

The *KinematicsController* is an abstract entity which must be specialized to implement some motion strategy. Example motion strategies are *CompliancyController*, *ToolLinearController*, *ToolVelocityController*, etc.

The SensorPublisher *Class*

To accommodate and distribute asynchronous sensor input, the implementer must provide specializations of the abstract class *SensorPublisher*. Some external sensors may support polling and some may submit asynchronous publications of their data. Both of these can be handled and cross-transformed in a specialized *SensorPublisher*.

As an example, consider a force sensor system which broadcasts force data over the network, but does not support polling. A *ForcePublisher* can be implemented to collect force data, and support polling internally to the controllers in the PyMoCo framework. This is an important decoupling mechanism.

3.2 Examples: Compliance Control System and Visual Servoing

A complicated control scenario, which serve as a good proof-of-concept, is the implementation of a 6D force compliance control. The NACHI robot has been equipped with a 6D force sensor at the tool flange, giving full force and torque data in its reference system. The force sensor is connected to a LabView application on a PC running Mandriva GNU/Linux. Sensor data are broadcast over UDP as fast as they are read off the sensor.

The purpose of a force compliance controller is to achieve the motion that “follow” the force and torque applied to the robot tool. The core communications in this PyMoCo application is best illustrated by the sequence diagram, shown in Figure 5.

It is important to note here, that there are two independent sequences in Figure 5, and that they trigger asynchronously. One sequence is triggered by the *ServoPublisher* publishing new state data from the servo controller, and the other is the one triggered by the *ForcePublisher*, publishing new force data. In general the sensor publishing event does not have a fixed timely pattern and may be cyclic, sporadic, or episodic. The coupling between the servo and sensor data is performed in the the *ComplianceController*.

Using the NACHI robot with $120ms$ tracking delay and $45ms$ response time, cf. Table 1, the input from the force sensor needs some filtering to match the delays in the servo controller. This was done by a simple exponential moving average with a suitable smoothing constant. The resulting control is adequate for manipulating the robot by hand, or generally in slow-varying force applications. This result would be similar with the KUKA robot.

Another case of use of the PyMoCo framework is described in [11]. In that application, the PyMoCo framework is used as a component in an application, which handles multiple sensor inputs and processing in its own framework. The PyMoCo framework is configured to provide a *ToolVelocityController* to the application.

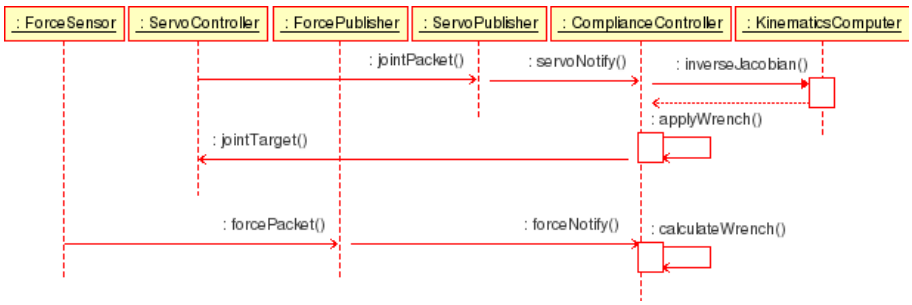


Figure 5: Sequence diagram illustrating the two, asynchronous, cyclic interactions in a single-sensor servoing controller.

4 CONCLUSIONS

We have described how access is gained to servo-level control from an external PC for three different industrial 6-axis manipulators. The three methods of access are very different, but may be considered as spanning the possibilities of external servo level control.

The experiments, based on sine wave tracking, show some considerable response times and tracking delays for the purely position-controlled robots, and minimal response time and tracking delay for the robot commanded by acceleration, velocity, and position.

The framework for external motion control, PyMoCo, has been sketched at the software design level. The framework provides some basic types of controllers, which can be customized or extended, and provides utilities and connectivity for further implementation of general or specialized motion controllers and sensor publishers.

Future work will concentrate on more advanced motion controllers and sensor integration in the presented framework. Parallel to this, we will be planning activities toward integration of new robot controllers as well as optimize the filter settings of the KUKA and NACHI controllers; cf. Section 2.3.

5 ACKNOWLEDGEMENTS

Thanks to professor Terje Lien, Norwegian University of Science and Technology, for good support and discussions.

We owe thanks to NACHI Robotic Systems Inc. for allowing a guided insight into their controller protocol. Without their willingness and help, the interaction with the NACHI robot would not have been possible in our setting.

Esben Hallundbæk Østergaard, Universal Robots, gave good help and guidance in interfacing to their servo controller.

Sebastian Dransfeld, SINTEF Raufoss Manufacturing AS, spent some hours in the laboratory on our request. We are thankful to him for performing the response measurements on the KUKA robot controller.

This work has been financed mainly by the IntelliFeed project, and, through the Norwegian University of Science and Technology, the RAMP project under the SFI Norman research programme. Both the IntelliFeed project and the SFI Norman programme are funded by The Research Council of Norway.

6 REFERENCES

[1] Carpanzano, E., Jovane, F., 2007, Advanced Automation Solutions for Future Adaptive Factories, CIRP Annals - Manufacturing Technology, 56/1:435–438.

[2] Jensen, S.M., 1998, Open Modular Controller, Proceedings of the 29th International Symposium on Robotics.

[3] Bruyninckx, H., 2001, Open Robot Control Software: the OROCOS project, Proceedings of the 2001 IEEE International Conference on Robotics and Automation, volume 3, 2523–2528.

[4] Vrba, P., Marík, V., 2005, From Holonic Control to Virtual Enterprises: The Multi-Agent Approach, Zurawski, R. (editor), The Industrial Information Technology Handbook, CRC Press.

[5] Blomdell, A., Bolmsjö, G., Brogårdh, T., Cederberg, P., Isaksson, M., Johansson, R., Haage, M., Nilsson, K., Olsson, M., Olsson, T., Robertsson, A., Wang, J., 2005, Extending an Industrial Robot Controller: Implementation and Applications of a Fast Open Sensor Interface, IEEE Robotics Automation Magazine, 12/3:85–94.

[6] Cederberg, P., Olsson, M., Bolmsjö, G., 2002, Remote control of a standard ABB robot system in real time using the Robot Application Protocol (RAP), Proceedings of the 33rd International Symposium on Robotics.

[7] Wetterwald, L.E., Dransfeld, S., Raabe, H., Ulleberg, T., Lind, M., 2008, Flexible Robotic Sewing with Real Time Adaptive Control, ElMaraghy, H.A. (editor), Proceedings of the 2rd CIRP Conference on Assembly Technologies and Systems, 552–561.

[8] Bigras, P., Lambert, M., Perron, C., 2007, New Formulation for an Industrial Robot Force Controller: Real-time implementation on a KUKA Robot, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 2794–2799.

[9] Schnell, J., Andersen, S., Langer, G., Sørensen, C., 1999, Development of a robot holon using an open modular controller, Proceedings of the IEEE International Conference on Control Applications, 2:1642–1647.

[10] Dallefrate, D., Colombo, D., Tosatti, L.M., 2005, Development of robot controllers based on PC hardware and open source software, Seventh Real-Time Linux Workshop.

[11] Schrimpf, J., Lind, M., Ulleberg, T., Zhang, C., Mathisen, G., 2010, Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control, Lien, T.K. (editor), Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems, Tapir Academic Press, NO-7005, Trondheim, Norway, 19–23.

4.5 Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control

Johannes Schrimpf, Morten Lind, Thomas Ulleberg, Chen Zhang, and Geir Mathisen. Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 19–23, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4

Declaration of co-authorship

The design and implementation of application, application control, vision analysis, and hardware is credited to Johannes Schrimpf, with Morten Lind and Thomas Ulleberg contributing in occasionally discussions. Thomas Ulleberg participated actively in all laboratory sessions during the development. Morten Lind contributed directly during integration of the PyMoCo framework for the robot motion control under the application, and participated actively in related experiment sessions in the laboratory. Chen Zhang contributed with discussions during development of the presented work.

Johannes Schrimpf single-handedly wrote the paper. Geir Mathisen contributed with discussions during the initial preparations for writing the paper. Chen Zhang contributed by discussing, commenting, and revising the paper during the entire writing process. Thomas Ulleberg and Morten Lind contributed by reviewing the final version of the paper. Johannes Schrimpf handled the preparation and submission of the manuscript.

Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control

Johannes Schrimpf[1,3], Morten Lind[2,3], Thomas Ulleberg[3], Chen Zhang[4], Geir Mathisen[1,5]

[1]Norwegian University of Science and Technology, Dept. of Engineering Cybernetics, Trondheim, Norway
[2]Norwegian University of Science and Technology, Dept. of Production and Quality Engineering, Trondheim, Norway
[3]SINTEF Raufoss Manufacturing AS, Trondheim, Norway
[4]Darmstadt University of Technology, Institute of Automatic Control, Darmstadt, Germany
[5]SINTEF ICT, Trondheim, Norway

AbstractIn the future, industrial robot systems have to be more flexible and autonomous to serve the needs of increasing product variation and shorter time-to-market. In the past, off-line programming of robot systems was often sufficient to meet the demands. But now, considering more complex automation tasks like handling of non rigid materials, the need for robot systems to interact in real-time with their surroundings is getting more and more important.
This paper describes real-time sensor servoing concepts aimed at industrial applications such as welding or sewing. To evaluate the methods, an adequate test platform is developed. A real-time, line-following algorithm based on a line-of-sight concept is presented, as well as a tool orientation control algorithm based on surface normal detection. The robot system consists of a 6-axis industrial manipulator and a vision system including a camera with four laser pointers used in distance measurements. The goal is to show new concepts for applications where the tool to surface-normal orientation has to be controlled in real-time according to a specified trajectory or control scheme, and where the path cannot be pre-programmed in the robot program. In the test case the tool is orientated perpendicularly to the workpiece surface and the path is given by a marking. Preliminary experiments verify that the algorithms work properly on the test platform.

Keywords:
robotics, manufacturing, real-time control, sensor servoing, visual servoing, line-of-sight, tool orientation control, laser triangulation, eye-in-hand

1 INTRODUCTION

Nowadays, increasing product variations, shorter time-to-marked, and more complex automation tasks like handling of non rigid materials, lead to new challenges in manufacturing. Industrial robot systems have to be more flexible and autonomous than ever. While in the past off-line programmed robot system were sufficient to meet the demands, today real-time interaction with the surroundings and thus sensor integration and real-time control are often desirable.

These demands have led us to implement a servo-level interface and an external motion controller for a NACHI SC15F 6-axis industrial manipulator [1]. The next natural step was to build, demonstrate, and evaluate a sensor interface, demonstrating the possibilities of real-time sensor servoing within manufacturing. It was decided to build a system which integrates different control algorithms and sensors to show the possibilities of real-time robot control. The focus is on cases where the tool has to follow a given path and has to be orientated perpendicularly to the workpiece, as for example in welding or sewing applications [2]. Common systems use time-consuming offline programming or, less time consuming, path planning based on CAD models [3]. To make the system more flexible for changes and independent of CAD files, the focus is on a system which does not need to be preprogrammed, but uses online path-planning in real-time.

The challenge can be divided into two smaller parts: the path-following and the tool orientation detection and control. Path-following is a common scenario in many automatization areas, not only robotics, for example in navigation of ships and vehicles. A common method is the line-of-sight algorithm [4, 5].

As solution for the surface normal detection, there were proposed different methods, including force sensors or visual sensors. Marques et al. use a triangulation based surface orientation and distance sensor which allows contactless sensing of the surface orientation [6]. A similar sensor is used by Caccia to detect the surface normals and distances for the navigation of underwater vehicles [7].

Other systems use force sensors to measure the surface orientation. One robot system which combines a fixed camera for position detection and force sensors for the surface orientation detection is developed by Hosoda et al. [8].

Zhang et al. proposed an automatic robot program generation method based on a combination of an eye-in-hand vision system to follow a marked path on the workpiece and a force sensor to measure the tool orientation [9]. In the resulting system, the robot has to move on a defined pattern, for example a zigzag path, to detect the local geometry. This is suitable for path generation in advance, but not for smooth path-following in real-time.

In this paper a visual line-of-sight tracking method is combined with the advantages of contactless surface orientation and distance measurement based on laser triangulation. This gives us a system which allows smooth line tracking with the tool orientated normal to the surface in real-time. Effort was concentrated on the practical implementation of the test platform which will be used as starting point for further evaluations of the real-time interface.

As test case, a scenario is defined where the tool has to follow an optical marked path on the workpiece, while orientated per-

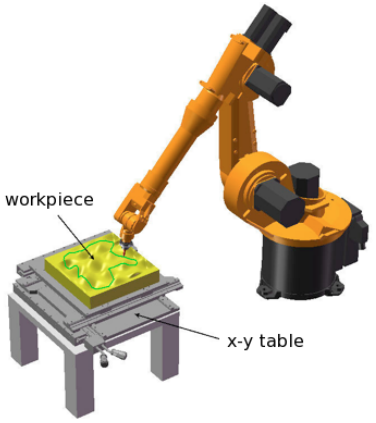


Figure 1: The test platform consists of a NACHI SC15F with a real time sensor interface and a hilly shaped workpiece.

pendicularly to the workpiece surface. The first tests verify that the algorithms work properly on the test platform.

2 SYSTEM DESCRIPTION

The test platform consists of a table with a 3-D structured surface and a NACHI SC15F 6-axis industrial manipulator with a tool, which is to be aligned perpendicularly to the workpiece surface and has to follow an optically marked line on the workpiece, as illustrated in Figure 1. The tool is a placeholder for a real tool, which can be used in applications such as welding and sewing [2].

The tool center point was defined to be at a given distance from the tool flange; in our experiments $10cm$. The tool coordinate systems origin is in the tool center point and the z-axis stands perpendicularly to the tool flange.

The desired movement of the tool is given by the combination of a marked path on the workpiece as well as the surface normal vectors at the desired path. A vector was defined, expressing the deviation of the actual tool center point from the desired tool center point in the tool coordinate system, both in position and orientation:

$$\mathbf{e} = \begin{bmatrix} e_x & e_y & e_z & e_{\theta,x} & e_{\theta,y} & e_{\theta,z} \end{bmatrix}^T , \qquad (1)$$

where e_x and e_y denote the distance between the desired position and the projection of the tool center point in the x-y plane. e_z is the deviation along the z-axis, and $e_{\theta,x}$, $e_{\theta,y}$ and $e_{\theta,z}$ the deviations in rotation around the x-, y- and z-axis, respectively. In general, correction of rotation around the z-axis is possible, but due to rotation symmetry it will not be considered in our demonstrator; hence, e_{rz} will be set to 0.

The system can be seen as a closed loop control system consisting of the vision system, divided in the hardware part and the software part; a robot controller (PyMoCo) written in Python [1]; the real-time interface to the robot controller; and the robot system. The structure of this system is depicted in Figure 2.

2.1 Tool Distance and Orientation Detection

Triangulation-based Distance Sensor Principle

To detect the tool distance and orientation with reference to the workpiece surface, a system was defined, which is based on

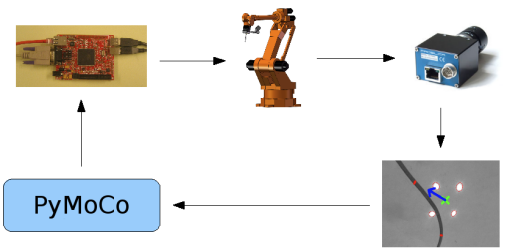


Figure 2: The control structure - from upper left: real-time interface, robot, camera, vision system, PyMoCo (Python motion controller).

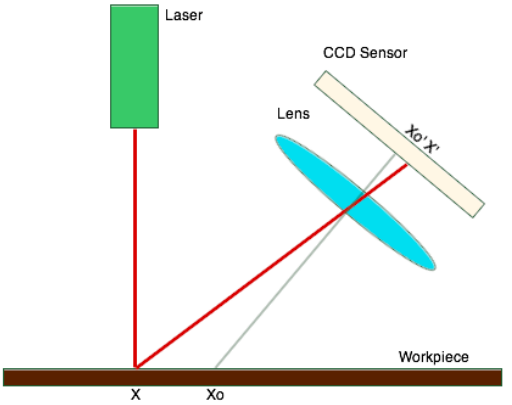


Figure 3: A triangulation-based distance sensor with a CCD-array. When the sensor is moved in vertical direction, the laser point is projected to another position on the CCD array.

several triangulation based distance measurements. The sensor system is closely related to the Opto 3D sensor described in [6] and a sensor described in [7].

Triangulation-based distance sensors consist of a light source, mostly a laser beam, which acts as a pointer, and a detector which is mounted at some distance from the laser source, as shown in Figure 3. Ideally, the laser source for triangulation has a high accuracy to illuminate a small spot over a large distance. Depending on the distance to be measured, the desired accuracy and the light conditions of the environment, infrared sensors can be used as light source instead of the laser.

The light emitted by the light source is reflected by the object's surface and returns to the detector. A lens focuses the reflected light onto a light-sensitive component, which can detect the position of the light point on the projection, e.g. CCD arrays, special photo diodes, or cameras. Thereby, the angle between the laser beam and the returning light can be measured, and hence the distance can be calculated.

Implementation of the Distance Measurement

In our setup, four laser pointers are used. Even if theoretically three laser pointers are sufficient, one extra laser is attached for symmetric reasons in the algorithm and to increase the accuracy by eliminating linear dependencies in the surface normal detection.

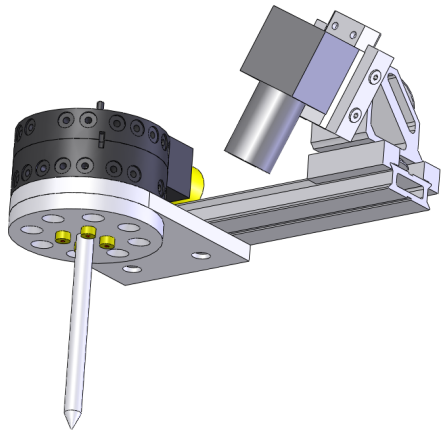


Figure 4: The sensor system attached to a tool changing system. The stick is a placeholder for a real tool.

The sensor hardware consists of four low cost laser pointers and a Prosilica GC1350 Gigabit ethernet camera, mounted on the tool as shown in Figure 4.

The distance d_{zi} between the sensor i and the surface can be derived by the formula

$$d_{zi} = f_i(u) \quad \text{with} \quad i \in \{1,2,3,4\} \ , \tag{2}$$

where u is the x-pixel-value of the center of the laser dot on the image and f_i is a nonlinear function, which is extracted from calibration measurements, as shown in Figure 5.

The resolution of the height measurement depends on the distance between the workpiece and the camera. The dependency is shown in Figure 6. In the working height of $10cm$, the resolution is around $4.3\frac{\text{pixel}}{\text{mm}}$.

The distance between the tool and the surface was defined to be the average of the separate distance measurements:

$$d_z = \frac{1}{4}\sum_{i=1}^{4} d_{zi} \ , \tag{3}$$

where d_z is the distance between the tool and the workpiece and d_{zi} are the separate distance measurements.

By taking the working height of $10cm$ into account, the height error e_z can be derived.

$$e_z = 10\text{cm} - d_z \ , \tag{4}$$

which is used as input to the tool height controller.

Implementation of the Tool Orientation Measurement

To determin the tool orientation in reference to the surface, the surface normal vector is calculated. The calculation of the surface normal vector of the workpiece at the tool center point is based on four distance measurements around the tool center point. On the basis of the measured distance and the physical setup it is possible to derive the coordinates of the four laser dots $\mathbf{p_1}$ to $\mathbf{p_4}$ on the surface in tool coordinates:

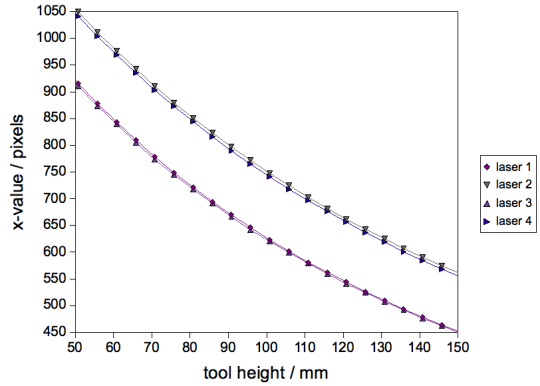


Figure 5: In the vision system the x-values of the laser point projection on the picture are converted to heights. This figure shows the dependency between the x-pixel and the height measurement for each laser.

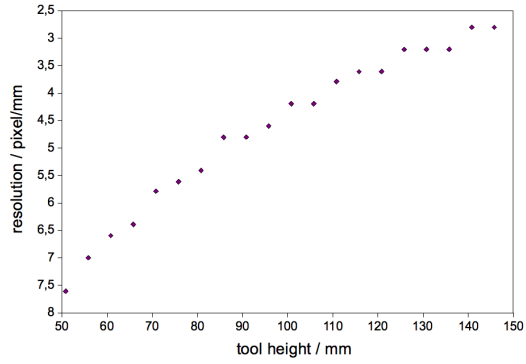


Figure 6: The height measurement resolution is dependent on the actual distance between the camera and the workpiece.

$$\mathbf{p_i} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T \quad \text{with} \quad z_i = d_{zi} \ , \tag{5}$$

where x_i and y_i are given by the coordinates of the lasers on the tool x-y-plane.

To derive the surface normal vector, it was assumed that the local area around the tool center point is planar. Now, two vectors are described by the positions of the laser points referenced in tool coordinates, preferably vectors with a right angle in-between. By calculating the cross product of two vectors, the resulting normal vector $\mathbf{n}$ can be calculated:

$$\mathbf{n} = \begin{bmatrix} n_x & n_y & n_z \end{bmatrix}^T = (\mathbf{p_3} - \mathbf{p_1}) \times (\mathbf{p_4} - \mathbf{p_2}) \ . \tag{6}$$

By comparing this normal vector with the tool's z-axis, the deviation from the desired position can be described, for example by a vector which acts as rotation axis and by the angle of rotation around this axis. In our case, the axis-vector lies in-plane and the rotation angle equals the angle between the normal vector of the plane and the z-axis.

Now the error values can be derived:

$$\begin{bmatrix} e_{\theta,x} & e_{\theta,y} & e_{\theta,z} \end{bmatrix}^T = \begin{bmatrix} n_x & n_y & 0 \end{bmatrix}^T \ . \tag{7}$$

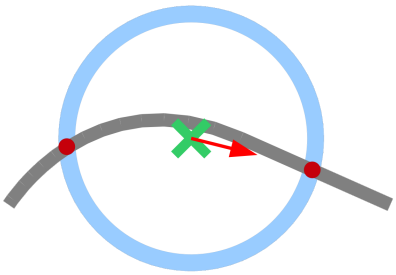


Figure 7: In the line-of-sight algorithm a circle is drawn around the tool center point, and the intersection points between the circle and the marked path are taken as possible way-points. Our algorithm choses the way-point with the smallest angle deviation from the actual movement direction.

$e_{\theta,z}$ is 0 in this case, since rotations around the tool's z-axis are not taken into account.

These values can now be passed on to a tool orientation controller which translates the deviation from the desired alignment to rotation commands for the robot controller.

Technically, the tool orientation algorithm is programmed in C++ using the OpenCV library [10], which is a cross-platform computer-vision library focusing on real-time image processing.

2.2 Line-of-Sight Path Following

As line tracking algorithm, an autopilot algorithm described in [4] is used. The method is called *Line-of-Sight guidance* (LOS). In the LOS algorithm the direction of motion Ψ_{LOS} of a vessel is defined by the coordinates of the next way-point:

$$\Psi_{LOS} = atan2\left(\frac{y_k - y}{x_k - x}\right), \tag{8}$$

where (x_k, y_k) are the coordinates of the way-point and (x, y) are the coordinates of the vessel. If the vessel enters a *circle-of-acceptance* around the actual way-point, the next way-point is chosen.

In our tracking algorithm, a sliding way-point is used, which lies on the line to follow in a constant distance. A circle is assumed around the projection of the tool center point, and the intersection points between this circle and the marked line are considered as possible way-points, see Figure 7. Our algorithm choses the way-point with the smallest angle deviation from the actual movement direction.

The main parameter in this method is the radius, which has large influence on both accuracy and robustness of the system. While a larger radius makes the system more robust, the accuracy is decreased. On the other hand, low values for the radius give more accuracy, but disturbances can make the method unstable. In general, this method is very robust against disturbances, when the radius is chosen large enough, but it is always a compromise between accuracy and stability [5].

As with the tool orientation algorithm, the tracking algorithm is implemented with OpenCV.

The output of the line-of-sight algorithm is the desired x_d and y_d value in tool coordinate system. These values correspond with the e_x and e_y values:

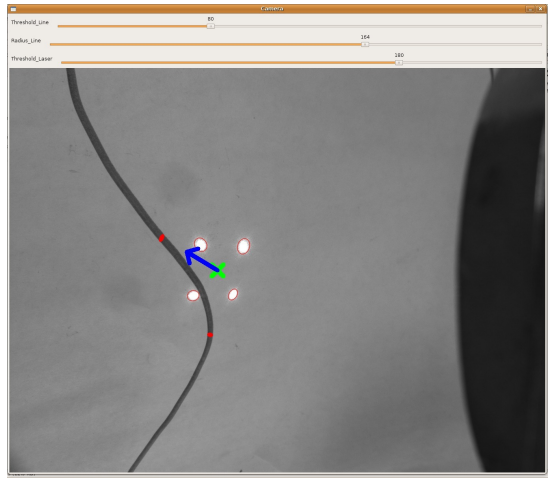


Figure 8: The graphical user interface allows to set the parameters for the line following and the tool orientation detection algorithms.

$$\begin{bmatrix} e_x & e_y \end{bmatrix}^T = \begin{bmatrix} x_d & y_d \end{bmatrix}^T \tag{9}$$

The stability is also highly dependent on the desired movement speed of the robot. For fast movements the radius has to be increased to ensure the stability of the tracker.

2.3 Tool Velocity Controller

The robot movement planning is done by using an open tool velocity controller which is connected to a servo-level controller interface of the robot.

The input to the tool velocity controller is a 6-element movement vector, in principle a twist, which includes both a linear movement vector and an angular velocity vector.

$$\xi = \begin{bmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T \tag{10}$$

The first 3 elements build a linear movement vector for the robot tool, while the remaining 3 elements describe angular velocities around the tool center point. The movement vector is derived by applying a P-Controller to the **e**-vector defined in formula 1:

$$\xi = \mathbf{diag}(\mathbf{K}) \cdot \mathbf{e} \tag{11}$$

with

$$\mathbf{K} = \begin{bmatrix} K_x & K_y & K_z & K_{\theta,x} & K_{\theta,y} & K_{\theta,z} \end{bmatrix}^T. \tag{12}$$

2.4 GUI

A graphical user interface (GUI) was developed to monitor and adjust the control program. It is shown in Figure 8. In the GUI it is possible to change the value of the radius of the line-of-sight algorithms. It can be adjusted according to the disturbances applied to the test platform. Two additional sliders allow adjusting the thresholds for line-detection and laser-detection for using the system under different light conditions.

In the figure, the dots on the line are possible way-points derived by the line-of-sight algorithm. The cross corresponds to the projection of the tool center point of the workpiece and the arrow points in the movement direction towards the chosen way-point. One can clearly see the laser points and their deviation from the square, which represents a slight non-perpendicular orientation.

3 EXPERIMENTS

To verify the mentioned control methods, a test case was defined where the robot tool had to follow a marked line with the tool orientation perpendicularly orientated to the workpiece surface.

A hilly shaped workpiece with a marked path was mounted on an freely movable table. The algorithms were tested under static conditions and under random disturbances simulated by shifting the table position during robot path following.

In the initial experiments it could be verified that the algorithms work properly in the test setup. Further experiments will be done to evaluate the influence of the LOS radius and the movement speed on the accuracy and stability of the line following.

Furthermore, delays in the robot system were discovered which are assumed to be motion buffering in the servo controller [1].

4 CONCLUSION

A test platform was built to demonstrate real-time sensor-servoing applications. A test case was defined where the tool has to follow a marked path on the workpiece and remain aligned perpendicularly to the workpiece surface. A contactless sensor system was implemented using a camera and four laser beams to measure the distance between the tool and the workpiece. The surface normal is calculated based on the distance measurements. The camera is also used to identify the path on the workpiece. A way-point system was implemented, based on a line-of-sight guidance algorithm. The vision system was implemented using C++ and OpenCV.

A controller was implemented in Python to convert the measurement results into motion commands, which are passed to a low level servo controller interface.

It was observed that the sensor-servoing system works properly in the initial test scenarios, and can be used for further experiments. It serves as a starting point for evaluation of different control strategies and servo-controller constraints in industrial applications [1].

5 ACKNOWLEDGEMENTS

The authors wish to thank the SFI Norman programme for financial support and supplying equipment to the RAMP project, in which this work was a part.

Also thanks to Terje Mugaas, SINTEF ICT, Department of Applied Cybernetics, for paving the way for the development of the real-time interface to the robot controller.

6 REFERENCES

[1] Lind, M., Schrimpf, J., Ulleberg, T., 2010, Open External Real-time Servo-level Robot Control Framework, Proceeding 3rd Conference on Assembly Technologies and systems (CATS 2010).

[2] Wetterwald, L.E., Dransfeld, S., Raabe, H., Ulleberg, T., 2008, Flexible Robotic Sewing with Real Time Adaptive Control, Proceeding 2nd Conference on Assembly Technologies and systems (CATS 2008).

[3] Mitsi, S., Bouzakis, K.D., Mansour, G., Sagris, D., Maliaris, G., 2005, Off-line programming of an industrial robot for manufacturing, The International Journal of Advanced Manufacturing Technology, 26/3:262–267.

[4] Fossen, T.I., 2002, Marine Control Systems - Guidance, Navigation and Control of Ships, Rigs, and Underwater Vehicles, Marine Cybernetics, Trondheim.

[5] Børhaug, E., 2008, Nonlinear Control and Synchronization of Mechanical Systems, Ph.D. thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

[6] Marques, L., Nunes, U., de Almeida, A., 1998, A new 3D optical triangulation sensor for robotics, Advanced Motion Control, 1998. AMC '98-Coimbra., 1998 5th International Workshop on, 512–517.

[7] Caccia, M., 2006, Laser-Triangulation Optical-Correlation Sensor for ROV Slow Motion Estimation, Oceanic Engineering, IEEE Journal of, 31/3:711 –727.

[8] Hosoda, K., Igarashi, K., Asada, M., 1998, Adaptive hybrid control for visual and force servoing in an unknown environment, Robotics & Automation Magazine, IEEE, 5/4:39 – 43.

[9] Zhang, H., Chen, H., Xi, N., 2006, Automated robot programming based on sensor fusion, Industrial Robot: An International Journal, 33/6:451 – 459.

[10] OpenCV, http://opencv.willowgarage.com/wiki/.

4.6 Development of a Low-Cost Prototype AGV

Olivier Roulet-Dubonnet, Morten Lind, and Terje Kristoffer Lien. Development of a Low-Cost Prototype AGV. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 25–29, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4

Declaration of co-authorship

The principles and rough design of the original prototype AGV was conceived by Morten Lind. The prototype AGV was built and equipped with power and logic electronics by Stefano Pedemonte; an exchange student from Politecnico di Milano. The proof-of-concept systems for motion control and low-level network communication was developed and implemented by Morten Lind. The AGV localization principle based on ceiling-mounted cameras and AGV-mounted LEDs was conceived and analysed by Morten Lind. A simple vision-based localization system based on these principles was implemented and integrated with the motion control system in close cooperation between Stefano Pedemonte and Morten Lind.

Cooperatively Olivier Roulet-Dubonnet and Morten Lind worked on analysing the fundamentals of an AGV system for coordinated and cooperative control of a group of AGVs, and further on the analysis of available communication middleware. Based on the cooperatively developed conceptual analysis and design, Olivier Roulet-Dubonnet implemented the agent-level middleware, later dubbed IceHMS, in its entirety. During development of the agent-middleware, Morten Lind continuously made minor design-level contributions and occasional case-based code corrections.

The writing, preparation, and submission of the paper is solely credited to Olivier Roulet-Dubonnet. Morten Lind contributed continuously in the writing process with minor reviews and produced the graphics in Figure 2; based on layout from Olivier Roulet-Dubonnet. Terje Lien contributed with reviewing the final version of the paper.

Development of a Low-Cost Prototype AGV

Olivier Roulet-Dubonnet[1], Morten Lind[1], Terje K. Lien[1]

[1]Department of Production and Quality Engineering, The Norwegian University of Science and Technology, Norway

Abstract
This article presents a low cost AGV System, developed and built at a laboratory at NTNU / SINTEF. The system is intended as an automated transport solution to and from prototype assembly cells. This article also presents a discussion of selected hardware and software components available for the development of low-cost flexible automated transport solutions. The AGVs have been built from available off-the-shelf hardware components and are controlled by open-source software. The localization system is based on odometry and ceiling-mounted cameras, which are communicating with the AGVs in a distributed holonic framework.

Keywords:
Manufacturing, Robotics, Assembly Technology, Automation, Holonic Manufacturing Systems, AGV

1 INTRODUCTION

Manufacturing automation in enterprises with large total production volume, small series and frequent changes in the product spectra, is challenging. The manufacturing system needs to handle high part variety, frequent changes and maintain efficiency. The new generation of free-ranging, small and fast Automated Guided Vehicles (AGVs) is a promising tool in this regard. AGVs can help reduce buffers and batch sizes by spatially decoupling the temporary storages from the manufacturing line. Compared to manual transport, AGVs can reduce the transport cost per unit and increase the transport reliability thus making it practical to increase the total transport and as a consequence improve the internal manufacturing logistic. Compared to other automated transport solutions, like pallet conveyors, AGVs offer unequalled flexibility. However, there are few free-ranging AGV systems commercially available and, to our indication, their current price level makes the return on investment too long for small to medium sized enterprises, which have often limited resources. To experiment with free-roaming AGVs as resources in automated manufacturing control, we have built an AGV System composed of off-the-shelf components and open-source software. The AGV System has been developed and built at a laboratory at NTNU / SINTEF. The system is intended as an automated transport solution to and from prototype assembly cells.

Different approaches are possible when developing a large mechatronic system. A particular point of dissension is the relative value of purchase cost against design, assembly, and maintenance cost. In manufacturing it is generally the rule to emphasize proven, reliable solutions. However this creates a less dynamic environment and industrial solutions tend to be more affected by the vendor lock-in issue than the consumer marked. The vendor lock-in issue is well described by P. Valckenaers et al.[1, sec3.2,p4] and can be summarized as followed: Vendor lock-in is the process of being locked to a specific product due to the reliance of our processes and equipments to the product specificity(in opposition to quality). Lock-in is a consequence of earlier choice and the problem is increased by the use of proprietary standards. Examples of vendor lock-in can be found everywhere, from CNC controllers to sensor systems and operating systems on main stream computers. When developing a system for research in flexible manufacturing we consider it important to value proven solutions, but avoid obvious vendor lock-in.

In addition to describing the current AGV System this article also presents a discussion of selected hardware and software components currently available for the development of a low-cost flexible automated transport solution.

2 HARDWARE

A prototype AGV System with three AGVs has been implemented at our laboratory facility. In this section we present the hardware used and the thoughts behind the chosen solution.

2.1 Geometry

The AGVs have been designed to carry pallets the size of a quarter of a euro pallet: 600mm x 400mm. This size offers the advantage of being roughly as large as a person, thus the robot can enter corridors and doors; It is also large enough to carry standard pallets from our partners pallets conveyors.

The mechanical frames of the first two AGVs are inherited from demounted appliances. This is not a long term solution and a frame made from bolted aluminium profiles is being designed for the third version.

The AGV use a differential drive propulsion system: Two independent wheels can rotate independently to move the robot. The advantages of differential drive, compared to ackerman steering, are the simplicity and the ability to turn on place. The propulsive wheels are situated at the back of the AGV and two casters are mounted at the front. The first AGV prototype can be seen in Fig.1. The use of casters is not problems free, they are unstable at higher speed and they make the orientation regulation at low speed difficult when they are not oriented in the currently chosen direction. However they are low-cost and readily available in many size and quality.

The batteries are sealed lead batteries on the first AGV and lithium ion batteries on the next models. The batteries are currently changed and recharged by hand.

2.2 Motors And Wheels

A central element of the robot design is the wheel-motor assembly. To simplify the design, thus assembly and maintenance, we looked for integrated wheel motor components. Wheel motors, also called hub motors are motor integrated into a hub. Many wheel motor products are available, unfortunately

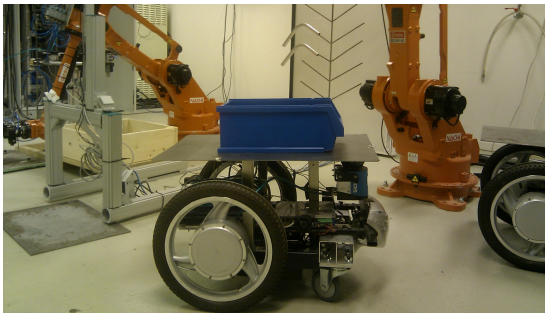


Figure 1: The first prototype AGV

they are designed for wheel chairs, scooters or bicycles and, as a consequence, do not have encoders. Encoders are necessary for precise velocity and position regulation, they are also used to compute odometry data. Adding encoders to a wheel motor system is often cumbersome as it does not have any rotational axes available. Using integrated encoders would also further decreases the number of parts. A possible solution is to use brushless hub motors and use the hall effect sensor for speed and position regulation.

The implemented solution is currently different on all three prototypes:

- The original AGV uses wheel-motors and custom rotating optical encoders which has been fitted inside the hubs by hand. The solution has been reliable the last three years but requires a lot of work for each new AGV to build.
- The second AGV uses the same wheel-motor and an external optical sensor which is a prototype developed by philips. The sensor is based on the technology used for optical computer mouse but with an increases precision from ten to three percent and a larger focus zone. The sensor is mounted on the frame of the AGV and uses the cover on the wheel as reflective surface. The sensor communicates with a micro-controller using SPI and is used, as rotating encoder for speed and position regulation ; the difference being that the result cannot be more than 3 percents accurate. This is though to be acceptable in our application.
- The third system uses a standard gear-encoder-motor assembly from a major electric motor manufacturer. This solves the encoder problem but brings in several additional parts.

2.3 Electronic

The first two prototypes are based on a standard design with a power board and wheels encoders connected to a microcontroller for real-time control. The micro-controller itself is connected to the main board through usb. The solution is flexible, allows for fancy solutions like optical extern encoders but requires much manual work to setup. The third AGV, that is not assembled yet, uses an industrial grade motor controller with input for encoders and integrated speed regulation. The motor controller is connected to the main board using serial. The solution is more expensive to buy and restricted to standard rotating optical encoders, but saves man hours to setup and rely on industrial components designed to be reliable.

Another approach, which has been considered but not applied yet, is to use a microcontroller or IO board to control the power board but to do the motor regulation in the main board. This is especially interesting when the real-time constraints can be achieved without a specialized real-time operating system. In our experience, common motor regulation frequency used in robotics, up to 100Hz, are easy to achieve with general barebones Linux systems with and without low latency scheduling. The main advantage of this solution is to keep flexibility but move microcontroller code in the, more development friendly, main board.

3 SOFTWARE

Software is a very complex and critical part of an AGV Solution and generally of autonomous robots. A possible solution to reduce development cost is to use code from open-source robotics projects available. With open-source we mean free-software[2], thus with a licence compatible with research and commercial use.

It is possible to classify the software needed for an AGV in two interconnected categories: The AGV controller and the AGV System. The AGV System is in charge of the high level AGVs managements. It handles the communication with the ERP and MES systems and all inter-AGVs issues like scheduling, and traffic regulation. The term AGV controller is loosely defined but, in our scope, we define it as being the software in charge of the local management of an AGV including motor regulation, sensors, localization and local navigation.

3.1 AGV System

The AGV System application is often partly customized or even implemented in-house to fit the shop floor requirements and other existing production systems. In this project we have chosen to develop a system based on the Holonic Manufacturing System concept (HMS). The HMS concept is well defined in P. Valckenaers et al[1].

HMS is related to Multi-Agent Systems, but they have some key differences since HMS is designed with manufacturing in mind. Holons, per definition, represent and contain physical devices while agents are normally software components. A holon can be composed of other holons to represent assembly cells and devices composed of other devices. In addition Multi-Agent Systems often implement heterarchic structures with emerging behaviour while HMS explicitly support hierarchic and hybrid structure.

To implement the system we reviewed available holonic and multi-agent tools. However the mainstream Multi-Agent systems [3, 4] are all java based platforms. A distributed manufacturing control system is an collection of heterogeneous hardware and software components; implementing a wide range of control entities. To integrate those components we need a common communication framework, but we consider a common software platform as a severe limitation when it dictates the programming language and impose constraints on the hardware devices.

As a consequence we implemented the AGV system using a custom framework based on *The Internet Communication Engine* (Ice^{TM}) [5] and is inspired by the work of Vallejo et.al[6]. Ice is a modern object-oriented communication middleware with support for many programming languages. Ice offers many features like persistence, replication, location service, load-balancing and publish/subscribe.

The current holonic framework is composed of two parts:

1. The Ice setup which define the network protocols and the Ice registry server to handle a Multi-Agent System.
2. Helper classes written in python to interact with Ice.

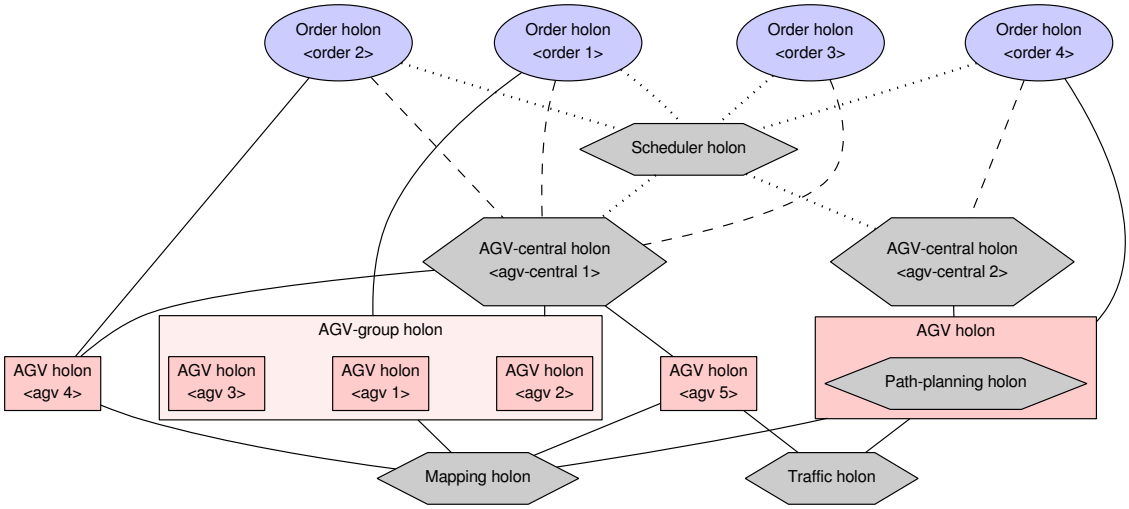


Figure 2: An overview of the AGV System, showing selected components

This makes it very easy to implement holons using Python. The clients written in other programming languages, so far C++ and java, need to use directly the Ice interface and follow the Multi-Agent System conventions. This has not been a limitation yet, but when need arises we will develop appropriate helper classes in other languages.

The AGV System has been developed based on the published literature on holonic manufacturing and material handling system. The system is currently a classic implementation of the PROSA architecture and is in several respects inspired by the work done by Srivastava et al.[7] and Barbiceanu[8]. Figure 2 shows the main holons of the system.

Following the PROSA architecture the order holons are the driving force of the holonic system. The order holons monitor the AGV System through the AGV-central holons and attempt to (re-)assign itself to the AGV with the most advantageous schedule, cf. figure2.

3.2 AGV Controller

There exist many open-source projects for mobile robots implementing some forms of AGV controller. The majority of those projects have been created with research or personal interest as motivation, something which clearly influenced their design and the range of implemented features.

The Player Stage project[9], seems to be the most used open-source software for mobile robots. It offers drivers for many research robots and devices and a set of servers that implement features like local navigation and map-based 2D localization. The project also developed a 2D and a 3D simulation programs which can be used through player. The Player client library is available in several languages, however, the servers, components that expose functionalities to other Player servers and clients, must currently be written in C++. CARMEN[10] is another project which describes itself as *a modular robot control software*. It is written in C and seems less active than the others.

In our context, the most interesting open-source project is The Robot Operating System (ROS)[11]. ROS targets more complicated, android likes, autonomous mobile robots with mechanical arms. It is, therefore, a more complete solution designed for robots with several computing units; Compared to the other available framework, some key features we appreciate are: Support for several communication modes(publish/subscribe and remote call), multi-language and a large set of tools to ease development. Special care has been taken from the developer to separate general robotic code from the ROS specific framework. The idea is to encourage the development of ROS agnostic libraries that can be used in other robotic frameworks. ROS is a research project but the developers are aiming to develop a production robot, and it is therefore a very good fit for manufacturing research. ROS is also the project that seems to be developed at highest rate.

In the current prototype controller we use the framework developed for the higher level distributed AGV System and assembly cell controller. The main advantage of the solution is the possibility to directly expose internal objects to the rest of the manufacturing system. Since the chosen framework is also a thin communication layer and is influenced by earlier experience with ROS it is relatively easy to port code from one platform to another.

4 LOCALIZATION ISSUE

Localization is a major challenge for free-roaming AGV Systems, it is therefore discussed in a separate section in this article. Many solutions available on the market are based on fixed markers on the floor or walls which are read by laser or magnetic sensors. Those solutions require opening the floor or fixing markers on the walls every few meters and are, consequently, of limited flexibility. In this article we review two solutions: Map based localization using laser range finders and a simpler system using ceiling mounted cameras.

4.1 Map Based Localization

Map based localization[12] is a promising solution that has the potential to solve the localization problem in a very flexible way. However, generating an internal representation of the spatial layout of the local environment to position itself is a very complex task which is an active research area.

There exists solution for 2D map localization[12], but 2D map localization gets easily confused by thin objects like chairs and

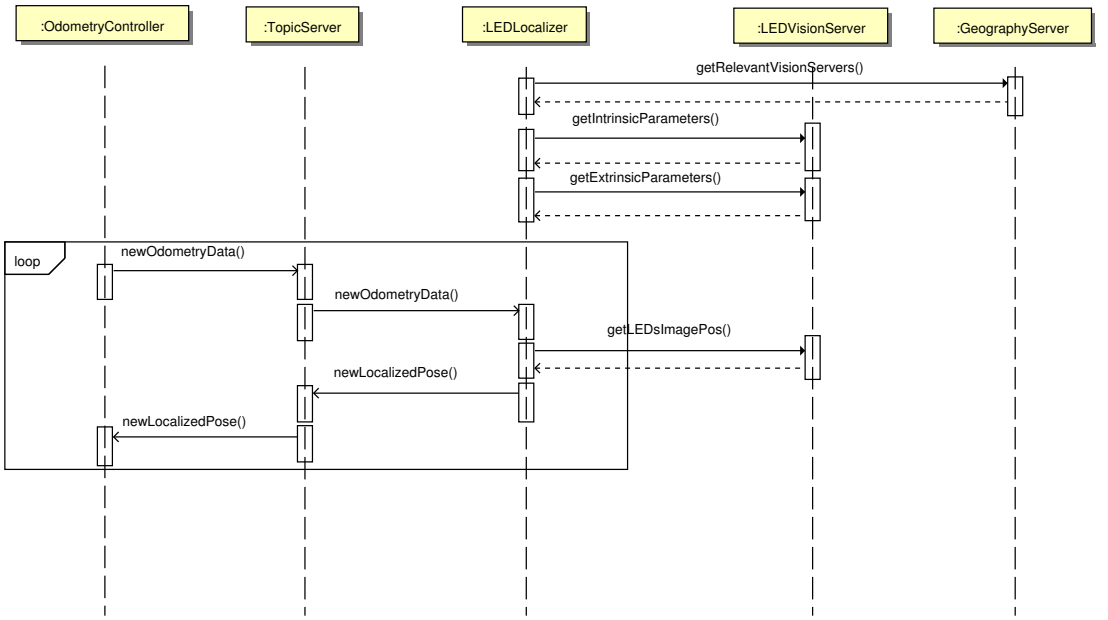


Figure 3: A localizer holon using vision holons and odometry data for localization.

does not take into account objects under or over the vision level. Localization based on 3D data is still a research area but we are approaching a solution.

Commercial complete solutions have started to appear on the market. An available AGV System from RMT Robotics is ADAM[13]. However price is still an issue for those systems. Even the sensors are still relatively expensive although their prices are getting rapidly lower, and sensors for map based localization are appearing on automatic mobile vacuum cleaner for home use[14].

We are experimenting with the navigation stack from ROS and have laser range finders installed on two AGVs, but the AGVs in the lab currently use a localization solution based on cameras mounted on the ceiling. The camera based system is especially interesting for precise local positioning in a loading or offloading station and within an assembly cell.

4.2 Localization Using Ceiling Mounted Camera

A potential low cost localization solution is to use roof mounted cameras which localize LEDs mounted on the AGVs. There exist many variations of camera localization systems using either cameras on the mobile robot looking for patterns on the ceiling or ceiling mounted cameras recognizing patterns on the AGV; they are regularly used in robotic competition.

The proposed system uses no pattern but four undifferentiated LEDs on each side of the AGVs and cameras attached on the ceiling. Each camera surveys a zone on the floor and stream a list of localized LEDs to the AGVs requesting the information. Using known camera characteristics, the AGVs uses the LEDs positions on the image to compute the world coordinates of the LEDs. Having prior knowledge of its own position, orientation and movement, the AGV can then deduce which LEDs are its own, cf. Fig.3.

The process of transforming LED camera image coordinates to LED world coordinates is called homography and has a unique solution when the Z coordinates of the LEDs in the world coordinate system is known. The computation can be done either by the cameras or the AGVs themselves. The process requires the cameras to be calibrated, thus to know the characteristic, position and orientation of the cameras.

The camera calibration process is manual, thus it is an issue and require further research. A potential solution is to use overlapping zones and the AGVs to automatically calibrate the cameras: If the cameras keep track of their calibration state, an AGV receiving different position information from two cameras can send correction information to the camera with the lowest calibration state. In theory this solution would only require to calibrate one camera at installation, but it is subject to quality degradation with the distance from the last calibrated camera. Although it is to be noted that, in our application, global positioning precision is of lesser importance that local positioning precision: If the local docking is within tolerance, it does not matter if the AGV system has the global position of the docking area wrong.

Another solution would be to implement automated calibration with markers on the floor but this solution suffers from the same problems as floor or walls magnetic markers.

5 DISCUSSION AND CONCLUSION

This article has outlined a low cost AGV System, developed and built at a laboratory at NTNU / SINTEF and intended as an automated transport solution to and from prototype assembly cells. It has also presented a discussion of the selected hardware and software, and their alternatives.

Many aspects of the system have not been considered yet, like battery replacement and maintenance. The safety standard for industrial AGVs defined in **BS EN 1525**[15] has also not been implemented; Consequently, the proposed system is currently targeted at unmanned areas. However, the AGVs are an interesting platform for manufacturing executing system prototyping and to experiment with new mechatronic solutions.

6 REFERENCES

[1] Valckenaers, P., Brussel, H.V., 2005, Fundamental Insights into Holonic Systems Design, HoloMAS, Springer, volume 3593, 11–22. 1, 3.1

[2] 2010, The Free Software Foundation, http://www.fsf.org. 3

[3] Bellifemine, F., Poggi, A., Rimassa, R.G., 1999, JADE - A FIPA-compliant agent framework, Proceedings of the Practical Applications of Intelligent Agents, The Practical Application Company Ltd. 3.1

[4] Busetta, P., Ronnquist, R., Hodgson, A., Lucas, A., 1999, Jack intelligent agents-components for intelligent agents in Java, AgentLink News Letter, 2:2–5. 3.1

[5] Henning, M., 2004, A new approach to object-oriented middleware, IEEE Internet Computing, 8, 1:66–75. 3.1

[6] Vallejo, D., Albusac, J., Mateos, J., Glez-Morcillo, C., Jimenez, L., 2009, A modern approach to multiagent development, The Journal of Systems & Software. 3.1

[7] Srivastava, S., Choudhary, A., Kumar, S., Tiwari, M., 2007, Development of an intelligent agent-based AGV controller for a flexible manufacturing system, The International Journal of Advanced Manufacturing Technology, 1–18. 3.1

[8] Babiceanu, R.F., 2005, Holonic-based control system for automated material handling systems, Ph.D. thesis, Industrial and Systems Engineering, Virginia Tech. 3.1

[9] Gerkey, B., Vaughan, R., Howard, A., 2003, The player/stage project: Tools for multi-robot and distributed sensor systems, Proceedings of the 11th international conference on advanced robotics, Citeseer, 317–323. 3.2

[10] Montemerlo, M., Roy, N., Thrun, S., 2003, Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit, Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Citeseer, 2436–2441. 3.2

[11] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A., 2009, ROS: an open-source Robot Operating System, Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA), IEEE. 3.2

[12] Meyer, J., Filliat, D., 2003, Map-based navigation in mobile robots: II. A review of map-learning and path-planning strategies, Cognitive Systems Research, 4, 4:283–317. 4.1

[13] Robotics, R., ADAM. The intelligent AGV., http://www.adam-i-agv.com/, accessed: 2010-03-02. 4.1

[14] Konolige, K., Augenbraun, J., Donaldson, N., Fiebig, C., Shah, P., 2008, A Low-Cost Laser Distance Sensor, Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena, California. 4.1

[15] 1998, BS EN 1525, Safety of industrial vehicles – Automated guided vehicles (AGV) and their systems. 5

4.7 Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control

Morten Lind and Olivier Roulet-Dubonnet. Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control. In Terje Kristoffer Lien, editor, *CIRP Conference on Assembly Technologies and Systems*, pages 67–72, NO-7005, Trondheim, Norway, June 2010. Tapir Academic Press. ISBN 978-82-519-2616-4

Declaration of co-authorship

Investigating and experimenting with Blender and the Blender Game Engine, as well as conceiving the idea of using it for real-time emulation of production devices with regard to control, is entirely the work of Morten Lind. Morten Lind modelled, developed, implemented, and experimented with the laboratory demonstration system from the IntelliFeed project.

The underlying agent-middleware, which was based on ZeroC Ice, was conceptually analysed and designed cooperatively by Olivier Roulet-Dubonnet and Morten Lind. Olivier Roulet-Dubonnet carried the conceptual design into an implementation, which was a development that ran in parallel to the development and implementation of the real-time emulation system. There was a strong interplay and collaboration between these two parallel developments, and Olivier Roulet-Dubonnet and Morten Lind closely inspected and commented on each others progress and problems.

Morten Lind was the sole writer of the paper and designer and producer of all the graphics. Olivier Roulet-Dubonnet contributed with valuable, continuous review during the writing process. Submission and preparation of the manuscript was undertaken by Morten Lind.

Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control

Morten Lind[1], Olivier Roulet-Dubonnet[1]

[1]Norwegian University of Science and Technology, Department of Production and Quality Engineering, Norway

Abstract
This paper describes a work in progress towards a framework for emulation of shop-floor devices, providing a basis for simulation of real-time manufacturing-control. Though the emulated reality is centralized, the surrounding manufacturing control system entities are distributed over ordinary Ethernet. The implemented principles of the device controllers, device interfaces, and control system agents is in accordance with the Holonic Manufacturing System paradigm.
The work is based on open and free software tools and systems; thus the resulting implementation can be made easily available. The main software tools used are Blender, ZeroC Ice, the Python programming language, and the Debian GNU/Linux operating system.
The implementation and principle has been used to make realistic simulations of a holonic AGV system and for shop-floor simulation of robotized upload to conveyor carriers for industrial paint-shop control.

Keywords:
Agent-based simulation; Shop-floor control; Holonic manufacturing systems; Distributed control system; Manufacturing system; Real-time control

1 INTRODUCTION

Recent paradigms for distributed shop-floor and factory control in manufacturing, notably the multi-agent and holonic paradigms [1, 2], allow for increased autonomy in orchestrating and controlling physical manufacturing entities. Control systems following these paradigms exhibit emerging behaviour, whereas more traditional, centralized control-paradigms induce systems with designed behaviour.

Background

To design, analyze, optimize, and verify manufacturing systems following traditional control paradigms, *Discrete Event-based Simulation* (DES) have a long history of application; for a comprehensive survey see e.g. [3]. DES efficiently utilize deterministic models of the simulated systems. Autonomous agents or holons in a *Distributed Control System* (DCS) can not, generally, in their very essence, provide such prerequisites. Hence, the emergent behaviour of agent-based or holonic control-systems must be realistically simulated; i.e. by agent-based simulation, using the actual agent code, and in real-time [2, 4]. For an original reference to *Holonic Manufacturing System*s and their motivation, confer [1].

Event-based simulations are useful in verifying, on a macroscopic level, that buffer capacities are adequate; that productivity, throughput, activity, efficiency, etc., are as expected; to search for critical operations and transport routes; etc. The event based simulation takes as a prerequisite that quite accurate and deterministic models of all processes in the entire system exists. This is at the crux of event based simulation, and allows time to skip forward between any two consecutive events, trusting the model that such two events are really consecutive. In a real DCS, with possibility for autonomy in every entity, such a deterministic and true model can not be constructed. Hence, to simulate an agent-based control system, it is necessary to deploy the real control entities in a real-time simulation system.

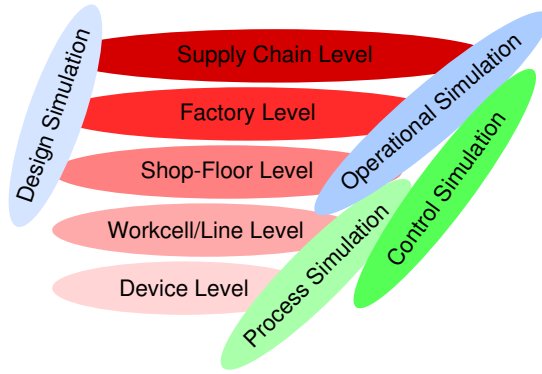


Figure 1: Levels of manufacturing and associated regions of simulation types.

Figure 1 illustrates a rough view on simulation categories associated with abstraction levels in manufacturing. Design, operational, and process simulations are a rough categorization of traditional simulation approaches.

Design simulations may not be timely or control related at all, but might focus on access and geometric layout of the shop-floor and factory levels. At the other end of the spectrum, process simulation may not be relating to the shop-floor or factory, or even device level, but could be a physics simulation of the melting in a welding process. Operational simulation would typically be based on results from design and process simulations and simulate, in a timely manner, how long-term operation of the factory performs. Its results would be in terms of statistics over variables like throughput, efficiency, equipment utilization, down-time, productivity, etc.

What is proposed is *control simulation* that operates at a mesoscopic level. It must capture the real-time control part of the

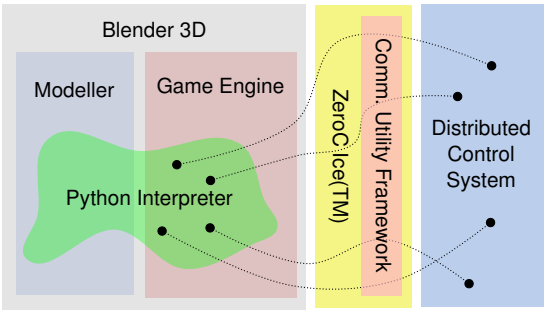


Figure 2: A structural overview of the basic architecture used for emulation/simulation.

device-related process simulation, and must extend upwards into the levels covered by operational simulation. This will be inefficient compared to DES, since it will be a real-time simulation at the control agent and process level. However, it may be the only way to accurately design, analyze, optimize, and verify a manufacturing system based on agent or holonic control. To address the inefficiency, depending on the depth of control detail, it may be possible to have a coarse or adaptive time-step.

Motivation

Rather than using an existing, general agent simulation platform, a custom-built platform was selected. This is motivated from the desire to simulate the behaviour of the actual agent system, which can then remain oblivious to whether it executes in the emulated or real world.

A simulation modelling tool, like MAST [5, 4], could have been used, if it was not tied to the Java programming language. MAST was thus relegated in favour of freedom of implementation language among the distributed entities. This freedom can be obtained by choosing a flexible communication middleware.

The framework and principles being developed for control simulation is not to be compared to a "simulation development environment" or "simulation studio". In its current design, it is rather to be considered a by-product of the process of developing a holonic or agent-based manufacturing control system.

The emulation framework presented in this paper is based on the Python programming language and the Blender animation studio. At the foundation of the system is the Ice communication middleware. These tools are all free and open software, as well as cross platform, and both Windows and GNU/Linux is used in development and experiments.

Paper Outline

The remainder of this paper is in two parts. First, Section 2 gives an introduction to the tools used and the architecture of the emulation framework. The second part, in Section 3, presents illustrative aspects from the simulation of an automated material handling case.

2 DEVELOPMENT TOOLS AND ARCHITECTURE

A structural overview of the development system architecture, underlying the implementing of an emulation and simulation system, is shown in Figure 2.

It is important to note the distinction between the emulation system and the simulating control system. The emulation system is entirely within Blender; specifically modelled with the modeller, driven by the *Blender Game-Engine* (BGE) and controlled mainly from the embedded Python interpreter. The simulating control system is the actually implemented DCS. It connects to the emulator on the local area network, and is separated from the emulator by a utility-framework implemented with the *ZeroC Ice* communication middleware.

The individual components of this structure are briefly introduced in the following sections.

2.1 Blender

Blender comes with a game engine, which is perfect for visualizing the motions and handling the geometrically related information in real-time. The best resource for documentation at all levels and parts of Blender is the Blender Wiki [6]. Following is a short introduction to the modeller and the BGE, with emphasis on the latter.

Modeller

The Blender modeller is a 3D modelling environment supporting mesh modelling. It does not support solid modelling, and as such is not a mechanical modelling environment. However, it is originally an animation studio, and the modelling features are indeed sufficient for modelling and visualizing a setup in a manufacturing environment.

An important aspect of simulation and modelling is the concept of a closed mesh surface, with which all solids are modelled. A closed mesh surface has, unless manually obscured, a clear distinction of inside and outside, by use of the surface normals on the individual faces. This enables a consistent use of closed surfaces as solids in the simulation.

As a very important feature, greatly enhancing usability and efficiency, the modeller supports library loading, such that objects or groups of objects from another Blender-model can be loaded. When loading an object or group from an external model, materials and properties from the pertinent objects are loaded with it. Upon loading, it is selected whether to load a copy or make a link to the external resource. The linking is preferable due to model reuse, where, for instance a model file for each type of robot can be created and maintained separate from a emulation model-file. When the robot model is updated in the model file, the instances of the linked robot is updated in the simulation model. Thus a set of library files for all reusable parts can be developed, maintained, and used across emulation models.

The modeller itself supports an API for the embedded Python interpreter, enabling Python scripts to control almost any part of the modeller functionality and interaction with all the modelled entities. The API to the modeller is alive even during BGE execution, enabling access to important information and resources not exposed by the BGE Python API.

Game and Physics Engines

The strength of the BGE is that almost any physical-mechanical process can be modelled and simulated realistically. For some kinds of processes, like free body dynamics with collisions, this can be very challenging. In some cases this can be severely levered by activating the physics engine. The physics engine can take care of such cases as friction, collisions, and general body dynamics. The physics engine even support advanced features as soft bodies, like textile or rubber, and particle systems, like smoke or steam. However, some deficiencies in the current version limit its use.

An example of said deficiencies in the physics engine regards the lack of support for static friction (or *stiction*). Thus a body resting on top of an accelerating body, will always move, no matter how slight the acceleration. Such situations need to be dealt with explicitly, by strategies for freezing dynamic bodies when they have been determined to have come to rest.

The BGE is set up in the modeller, by adding logic elements and properties to geometric entities. To every geometric entity any number of *sensors*, *controllers*, and *actuators* can be added; these are referred to as *logic bricks*. Any sensor can activate any controller and any controller can activate any actuator; this goes for the entire set of sensors, controllers, and actuators among all geometric entities. The cardinality of the links between sensors and controllers and between controllers and actuators is completely free.

The sensors are the triggers of events to the controllers, and when triggering they contain and supply valuable information to the triggered controller code. The concrete sensor types available comprise a large set. The sensors may be roughly classified into the categories of geometry, mechanics, time, and user interaction. The trigger semantics for any sensor can be set among such types as pulse mode, tap mode, positive, negative, and some more advanced modes.

The controllers are more limited in type. Basically the controller types are classified into Boolean operators on sensor input and Python controllers. The Boolean controllers work by doing a Boolean operation on the sensor triggers logic states, and on positive result triggers the associated actuators. Boolean operators and actuators are not used in the current experimental system. Rather, the more general Python module-type controllers are used, which call a Python function in a module, or a method on a Python object.

The Python controllers, rather than trigger associated actuators, operate directly on the Blender APIs of the modeller and the BGE. For most cases, use of actuators are unnecessary. It is, however, important to know that some special operations can not be done on the APIs, and have to be performed by actuators. One such special operation is that of adding an object to the game scene. For this, an *AddObjectActuator* must be allocated at configuration-time in the modeller; it can, however, be re-configured at execution time.

A facilitating property of the embedded Python interpreter in Blender is that it is unrestricted. Thus, everything possible with and accessible to an ordinary Python interpreter on the OS platform is also accessible for the Python code in the embedded interpreter. Important examples of technologies are the NumPy numerical extensions to Python, the use of threads and sockets, and connection to the communication middleware. Availability of threads and sockets enables the code in the interpreter to set up active agents or holons inside the embedded interpreter.

2.2 ZeroC Ice

The emulated devices are exposed to the DCS over Ethernet using the *Internet Communication Engine* (Ice) [7]. Ice is an modern, object-oriented communication middleware, developed by ZeroC, Inc., providing a full set of features to support development of distributed application.

The utility framework is used directly in the BGE. This would not have been possible using an existing Multi-Agent System platform. The scalability of the simulation is also facilitated by the documented low overhead of Ice [8].

Ice is also multi-language, and thus the emulated devices can be accessed using any of the programming languages supported by Ice. Compared to raw sockets, Ice offers high-level managed facilities such as location services, the publish/subscribe pattern, and synchronous and asynchronous method invocations.

The exposed interfaces of all registered objects, notably the emulated and real devices, are defined in *Slice* interface language. Slice is a purely declarative language to describe object interfaces exposed through Ice. The devices have a complex inheritance hierarchy which is defined in Slice files. The same Slice files are used by the real and emulated devices. Thus, exposing the same interface over the network , the emulated devices are not structurally discernible from the real devices.

2.3 Application Architecture

The development system architecture was illustrated in Figure 2 and described in the preceding sections. This section will give, in abstract terms, a short overview of some principles for application-setup in the BGE.

Devices and Sensors

An emulated device in the BGE is a composition of blender geometries making up for the material part, and properties, internal states, and rules for motion in the device controller in the embedded Python interpreter. A device can have an interface exposing it to the DCS. It can alternatively be purely emulator-internal, in case of a device that is entirely uncontrolled or implicitly controlled by other device controllers.

Emulated sensor devices may comprise the whole range of physical sensors used throughout robotics and automation. They may work by explicit relations among the geometries in the model, or by implicit relations deducted from emulated device controllers working with the model.

Initialization

A special BGE sensor will always be set up to trigger initialization of all emulated devices and sensors in the embedded Python interpreter. This BGE sensor will trigger only in the first time-frame of the execution session, and its associated BGE controllers will simply invoke functions in all the active modules for creating control devices necessary for the emulation.

The most flexible setup is achieved by having complex and adaptive initialization code in all control modules, which, upon inspection of the geometries and their properties in the emulation scene, can construct the corresponding Python controllers.

Time-Step Update

Some controllers will be bound to synchronizing their control update to the time steps in the BGE. For this, as a principle, there will always be a BGE “Always”-sensor configured, which will call a dedicated Python controller that can send a “timeUpdate” event to all device-controllers in need.

An example is an emulated robot servo controller, which will need, in each time frame of the emulation, to update the position of the links of the robot. In contrast to this, a controller for a binary finger-gripper, i.e. a gripper that has only two states of the finger set, is never activated from inside the BGE, but only upon a control message from the external DCS.

A sensor or controlled device may need sub-time-step updates. Such can not be generated from the BGE sensors, but the freedom in the embedded Python interpreter allows for creation of a thread that sends a message at any frequency that may be desired.

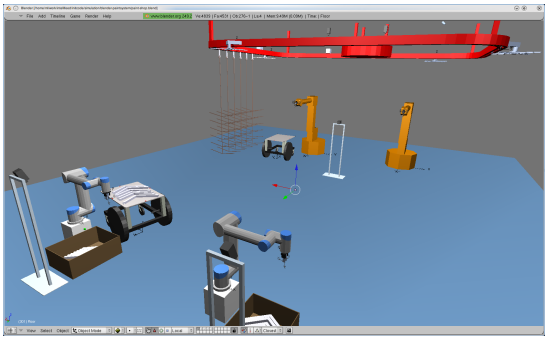


Figure 3: A perspective overview of the case setup from the Blender modeller.

Event Update

The event-based updates come from configured BGE sensors. Of importance to modelling real sensors used in manufacturing automation are the geometrically related BGE sensors for triggering on collisions or proximity. Other BGE sensor types, relating to time, mechanics, user interaction, and properties and states of BGE objects, may also contribute to the implementation of CPU-efficient or simpler controllers.

Distributed Control System

The DCS, on the right in Figure 2, is the target for the simulation. The BGE emulator is merely providing a consistent world for the DCS to execute with.

The external control system may be executing in a single OS process; especially so in simple cases. In fact, the only constraints on the implementation and deployment architecture of the external control system, are that the timeliness demanded by the application is adequate and that Ice communication is supported.

3 SIMULATION CASE SYSTEM

The simulation case, which has been the general target for developing the emulation framework, is a laboratory case in the IntelliFeed project; a joint project at The Norwegian University of Science and Technology, Department of Production and Quality Technology and at SINTEF Raufoss Manufacturing AS. An overview of the laboratory setup, as modelled in Blender, is seen in Figure 3.

The case setup in the laboratory and modelled in the framework is a shop-floor section supplying workpieces to an industrial painting system. The red rail-system in the upper part of Figure 3 is a PnF-conveyor loop, on which painting system carriers, or rather, PnF-trolleys, are circulating. Filled carriers are to be transferred to a chained trolley conveyor of the painting system. This main conveyor of the painting system is not part of the laboratory setup.

The PnF feature of the conveyor is a major facilitator, if not a necessity, when automating the upload of parts for painting to the painting system carriers. PnF-stops have been placed at the upload robots and at various buffers around the PnF-track. A PnF-stop is a mechanical entity, of which the blocking state, with respect to the trolleys, can be pneumatically controlled. Thus, a painting system carrier can be stopped and fixed during upload, and the trolleys can be stacked at, and released from, buffer points before upload and transfer, to ensure correct process timing.

The two orange NACHI SC15F manipulators in the background are handling workpieces for painting. The workpieces are picked from the supplying AGVs and hung onto the painting system carriers. The painting system carriers, seen as brown skeletal objects hanging down from the left end of the PnF-conveyor, are all unique due to mechanical entanglement in their handling, so a 3D stereo-vision system is used for identifying every hanging point. The 3D vision system used is described by Ystgaard in [9].

The AGVs, of which two are seen in the scene, transport workpieces to order from the two picking cells seen in the foreground and on the left. The picking cells are not yet set up in the laboratory, but are under design. They are based on a lightweight manipulator from Universal Robots.

This section illustrates, without going into too much detail, some representative interaction sequences for different categories in the software system. UML sequence-diagrams are used to illustrate examples of the interactions internal to the emulation system; interactions between the DCS and the emulation system; and, for completeness, interactions at the logic level in the DCS.

3.1 Emulation-Internal

A lot of code is spent on maintaining the emulated reality at the physical level. This functionality is not exposed over Ice, since it will not be accessible to the DCS in the real application.

As a simple example the following scenario is considered: A train of two PnF-trolleys are blocked by a PnF-stop and then subsequently one of the trolleys are released from the PnF-stop. Though the scenario is simple, a lot of communication and activation takes place to realize it in a realistic manner.

The scenario is presented as the sequence of messages exchanged between the BGE, the PnF-stop, and the two PnF-trolleys. The control of the trolleys is purely internal to the emulation, since they simply follow the conveyor chain; unless stopped against an active stop, or against another trolley. The only externally controllable entity in the scenario is the PnF-stop. The external activation of the PnF-stop agent may suggest that the presented scenario is not exclusively emulation-internal. However, the role played by the external activation is minor, and the emphasis is on the communication internal to the emulation.

Figure 4 shows the sequence diagram for the scenario. The situation assumed at the beginning of the scenario is that two PnF-trolleys are approaching the PnF-stop. The scenario begins with an external activation for closing the PnF-stop. The PnF-stop agent accomplishes this in the BGE by moving its geometric stopping object into the path of the trolleys. As the first trolley arrives at the stop, the stopping geometry triggers the collision sensor of the trolley, signalling that it touched a blocking stop. Later, the second trolley will collide with the first, triggering a sensor message from the BGE to the second trolley that it is colliding with another trolley.

The collision sensor messages from the BGE actually triggers whenever there is a *change* in the collision state for the pertinent object. So, it is up to the PnF-trolley agent to infer if a new collision occurred, or if an existing contact disappeared. The first case will lead to the trolley stopping and the latter may lead to the trolley resuming motion again.

As the PnF-stop agent now receives an external request to release one single trolley, it will retract the stopping-geometry, which is blocking the way for the first trolley. As it is supposed to release only one trolley, it will start listening for changes in its

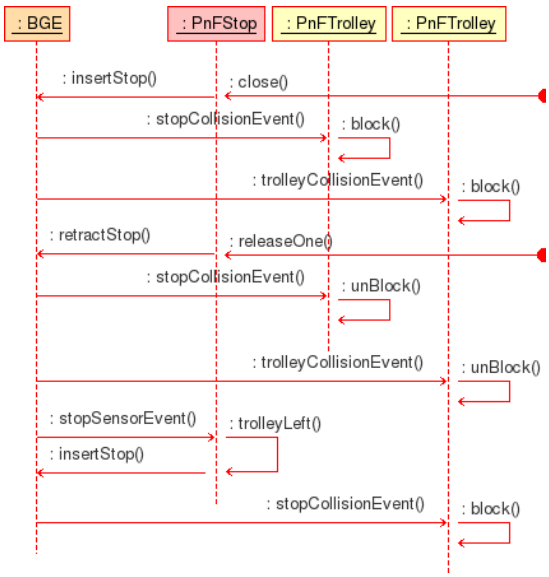


Figure 4: The mechanism for stopping a train of independent PnF-trolleys by a PnF-stop, and then release one of the trolleys.

associated proximity sensor, to determine when the first trolley has left the blocking region.

Meanwhile, the stop collision sensor for the first trolley triggers, signalling that there are no longer contact with the stop-geometry, and the trolley will start moving again. A short while later, the second trolley will have its trolley collision sensor trigger, since it is no longer in contact with the first trolley. Thus both trolleys are now moving again.

When the first trolley has moved entirely past the proximity sensor of the PnF-stop, the sensor triggers the PnF-stop agent. The PnF-stop agent acts upon this by once again inserting its stop-geometry into the track of the trolleys. Hence, as the second trolley reaches the stop, it will be blocked again.

This scenario is applicable at every buffer point on the PnF-conveyor where there is a PnF-stop. Inside the emulator as well as in reality. It can be extended almost trivially backward and forward in time. The number of trolleys will, of course be different, and in some cases it is up to the external control logic to ensure that there is never more than one trolley at a PnF-stop.

3.2 DCS-Emulator Interaction

As an illustration of the interaction between the DCS and the emulated devices, an example of an iteration of the cyclic sequence of one of the picking cells is presented. The picking cells are the two cells shown in the lower and left part of Figure 3.

Either picking cell is composed of the small, lightweight 6-DOF manipulator from Universal Robots, a box of accessible workpieces to pick from, a workpiece vision system to locate pickable workpieces from the box, and a docking area for AGVs.

In the sequence diagram in Figure 5 the picking cell agent is seen to the left. It is deployed in the DCS and connects to, and controls, all the shown emulated devices. The only other resource used from the DCS is the carrier layout agent, holding information about where the picked parts should be placed.

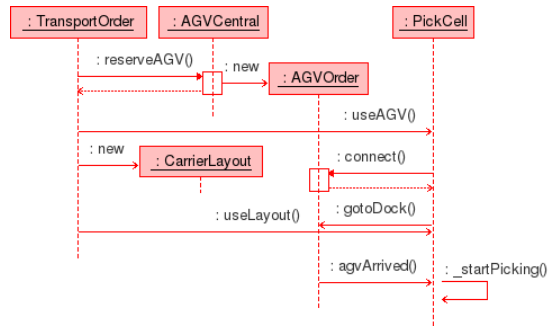


Figure 6: The sequence for initiating a transport of parts from a picking cell.

The picking cell agent gets the poses for the visible workpiece from the workpiece vision system. A workpiece is selected, the selection strategy not shown, and the motion controller for the robot is commanded to the grasp pose. Naturally, the motion controller interacts heavily with the BGE for performing the motion with the robot arm, but for clarity in the sequence diagram, this has been left out. Meanwhile, from the carrier layout the next pose for putting the workpiece in the transport carrier is retrieved. As the motion controller reports the completion of the motion, the tool controller is commanded to grasp. In case of a finger gripper, this results in moving the geometries for the fingers of the gripper in the BGE. Inspection of messages from BGE collision sensors, associated with the finger geometries, serves to assert correct grasp of the workpiece. The motion controller is then commanded to go to the placing pose, and then the tool controller is commanded to release the grasp.

It is interesting to consider the “graspOK” message sent back to the picking cell agent from the tool controller. In an ordinary setup for picking, there are no sensory system associated with the fingers on the gripper. In such a case, the message can only carry information that the gripper did the motion of the fingers. This does not mean that the workpiece was actually grasped. However, if some sensory system measures that there is contact with something between the fingers, or a vision system can observe the fingers, the message can actually carry information that some object is grasped. Either way, the emulated “finger sensors” will be present, if only to serve the internal bookkeeping of the BGE, about which workpiece or object to actually move with the gripper. Whether or not the information of the emulated sensors can be exposed to the simulation system, depends upon whether the modelled, real system has such sensors.

3.3 DCS-Internal

For completeness, but unrelated to the BGE emulation, the sequence in Figure 6 illustrates the initiating role of a holonic transport order. The purpose of the transport order is to supply workpieces from the picking cells to the upload robots at the conveyor system. The sequence takes place among agents exclusively deployed in the DCS.

The transport order requests an AGV for the transport task from the AGV central, which in turn creates an AGV order associated with a physical AGV. The transport order then sets up the picking cell with a suitable carrier layout for the set of workpieces to deliver. When the AGV order reports arrival of the AGV at the specified docking pose at the picking cell, the picking cell can start the cycle over the operation that was illustrated in Figure 5.

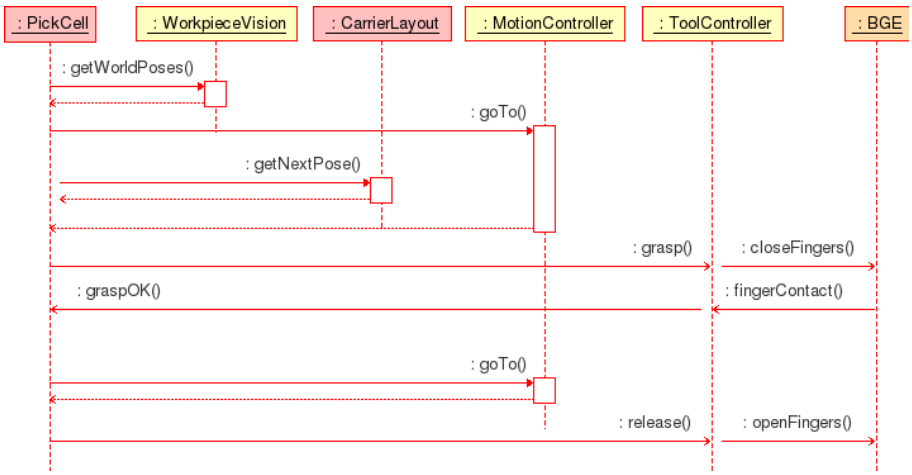


Figure 5: A pick sequence involving a fair amount of interactions.

4 CONCLUSION AND FUTURE WORK

We have described a work in progress towards a framework for real-time emulation of shop-floor devices. Some preliminary simulation activity has begun, using a DCS in connection with the emulation.

The AGV-system described in [10] has been fully emulated with the preliminary framework described in this paper, and has helped in developing the holonic level control system for the AGV-system.

The emulation for the laboratory application, used as case in Section 3, has been modelled almost completely. Some design decisions regarding workpiece handling still remain. The development of the holonic control system for the laboratory case can proceed when these final decisions have been made.

When the described simulation case is completed, work will proceed in extending and refining the simulated system. It will be incrementally expanded to model and emulate more of the factory surrounding the painting system. Depending on optimizations and techniques, it shall be interesting to see to what extent the simulation principle will be useful.

Several other manufacturing prototype systems at our department also have distributed real-time control aspects relevant for emulation and simulation similar to the painting system.

For scalability reasons, it will be interesting to attempt to distribute the emulation model among several nodes, each running a Blender Game Engine. One challenge with this is how to maintain a consistent emulation of the manufacturing system in real-time across the game engines.

5 ACKNOWLEDGEMENTS

Thanks to our supervisor, professor Terje Lien, Norwegian University of Science and Technology, Department of Production and Quality Engineering, for supporting this work.

Thanks to Per Aage Nyen, SINTEF Raufoss Manufacturing, and Johannes Schrimpf, Norwegian University of Science and Technology, Department of Engineering Cybernetics, for important comments and discussions.

Thanks to professor Amund Skavhaug, Norwegian University of Science and Technology, Department of Engineering Cybernetics, for interesting discussions regarding distributed and real-time control.

This work has been financed through the IntelliFeed project, funded by The Research Council of Norway.

6 REFERENCES

[1] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P., 1998, Reference architecture for holonic manufacturing systems: PROSA, Computers in Industry, 37/3:255–274.

[2] Vrba, P., Marík, V., 2005, From Holonic Control to Virtual Enterprises: The Multi-Agent Approach, Zurawski, R. (editor), The Industrial Information Technology Handbook, CRC Press.

[3] Smith, J.S., 2003, Survey on the Use of Simulation for Manufacturing System Design and Operation, Journal of Manufacturing Systems, 22/2:157–171.

[4] Vrba, P., Marík, V., 2005, Simulation in Agent-based Manufacturing Control Systems, Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, IEEE, volume 2, 1718–1723.

[5] Marík, V., Vrba, P., Fletcher, M., 2005, Agent-Based Simulation: MAST Case Study, Emerging Solutions for Future Manufacturing Systems, 159:61–72.

[6] Blender Wiki, http://wiki.blender.org/.

[7] Henning, M., 2004, A New Approach To Object-Oriented Middleware, IEEE Internet Computing, 8/1:66–75.

[8] Henning, M., 2009, Choosing Middleware: Why Performance and Scalability do (and do not) Matter, Online.

[9] Ystgaard, P., 2010, Accuracy in binocular robot-vision system, Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems.

[10] Roulet-Dubonnet, O., Lind, M., Lien, T.K., 2010, Development of a Low-Cost Prototype AGV, Lien, T.K. (editor), Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems, Tapir Academic Press, NO-7005, Trondheim, Norway, 25–29.

4.8 Holonic shop-floor application for handling, feeding and transportation of workpieces

Morten Lind and Olivier Roulet-Dubonnet. Holonic shop-floor application for handling, feeding, and transportation of workpieces. *International Journal of Production Research*, 49:1441–1454, 2011. ISSN 0020-7543. doi: 10.1080/00207543.2010.519115

Declaration of co-authorship

This paper is an invited journal paper based on two papers from the HoloMAS2009 conference proceedings. Sections 1 and 2 are integrated introductory sections from the two original papers and may be considered as a shared contribution from Olivier Roulet-Dubonnet and Morten Lind. Section 3 may be considered as a pure contribution from Morten Lind, and Section 4 as a pure contribution from Olivier Roulet-Dubonnet.

While Morten Lind was the responsible author and handled the integration of the texts, the writing credits largely follow the contribution credits as stated above. The submission and review process with the journal and editors was handled by Morten Lind. Morten Lind created the graphics in Figures 1 through 4, and the diagram in Figure 5 was produced by Morten Lind, based on layout from Olivier Roulet-Dubonnet.

International Journal of Production Research
Vol. 49, No. 5, 1 March 2011

RESEARCH ARTICLE

Holonic Shop-Floor Application for Handling, Feeding, and Transportation of Workpieces

Morten Lind[a*] and Olivier Roulet-Dubonnet[a]

[a]*Department of Production and Quality Engineering*
NTNU Valgrinda 7491, Trondheim, Norway
(*Received 00 Month 200x; final version received 00 Month 200x*)

In pursuit of flexible and agile automation within the domain of discrete manufacturing, a paint-shop with its surrounding logistics and handling processes is under construction as a laboratory prototype application.

Holonic Manufacturing is argued to be a promising strategic paradigm and architecture to use for a system characterized by flexible production logistics and control. This paper describes the physical devices to be used; the desired functionality; and the basic logic control design. Additionally, the ideas for holonification, based on the already designed logic control, are presented.

The paper also outlines a holonic Automated Guided Vehicle (AGV) system developed for the automated paint-shop. The AGV system is composed of autonomous AGV holons that cooperate, individually or in groups, with other holons, such as robot holons and vision-system holons.

Keywords: Holonic Manufacturing Systems; AGV system; Part handling; Robot systems; Shop-floor control

*Corresponding author. Email: morten.lind@ntnu.no

ISSN: 1478-6451 print/ISSN 1478-646X online

1. Introduction

A niche for small and medium enterprises is to supply individually tailored products or services. If the production is done on a large scale, this is called mass customization (Zipkin 1997, Davis 1987) and requires highly flexible automated production systems. To deal with the complexity arising from the desired flexibility, a natural strategy is the distribution of the control system. This way, functionalities gets separated and isolated to their natural sites. Distributed decision-making and cooperation of autonomous sections in manufacturing have been around as long as complex manufacturing. "Complex manufacturing" in the present context can be described as *"dealing with multiple complex products and extensive sharing of manufacturing equipment across different simultaneous product variants"*, i.e. involving frequent changeovers and reconfigurations on the shop floor. In the recent decades there has been focus on the flexible automation under such circumstances. The enabling technologies for this may be traced back to the birth of numerical control and computational intelligence in the 1950s.

Holonic Manufacturing is a paradigm for pervasive distributed manufacturing automation, ranging from the lowest level of real time shop-floor control and all the way up to company or even corporate level. It covers most aspects of manufacturing, be it machine to machine cooperation or order to production department interaction. The concept of a *Holonic Manufacturing System* (HMS) date back to the early 1990s when the *Intelligent Manufacturing Systems* (IMS) initiative set out a project with that name. The term *Holon* was coined by Koestler (1967), some 40 years ago, for capturing the dualistic properties of autonomy and cooperativeness within a single entity. Van Brussel (1994) discusses various approaches to autonomous, distributed control and argues that HMS is suitable for manufacturing control and management.

There exist some general architectures for Holonic Manufacturing Systems, such as PROSA (Van Brussel *et al.* 1998) and ADACOR (Leitão 2004). PROSA is strictly a reference architecture and introduces the central concepts of basic holons: order, product, resource, and staff holons. High-level scenarios illustrate the interactions of the different holon types. ADACOR is also an architecture, but with a different naming of the holon types. Notably the ADACOR supervisor holon differs from the PROSA staff holon, in that it formally coordinates the dynamics of holon aggregation and subordination. Leitão and Restivo applies the ADACOR architecture to a (partially simulated) machining and assembly workshop in several papers; see e.g. Leitão and Restivo (2005, 2008).

In an early paper on holonic AGV control, Liu *et al.* (2000) proposes a distributed holonic architecture, where all service requests are handled by the AGVs themselves. Srivastava *et al.* (2007) presents an approach for conflict-free shortest path, minimum-time motion-planning and deadlock-avoidance for AGV systems. He presents an architecture for AGV systems which is influenced by it's focus on zone algorithms. The architecture is partially reused in the presented research project. In his PhD thesis, Babiceanu (2005) proposes a holonic-based control system for automated material handling systems. The thesis focuses on scheduling and he shows that the results, obtained by running the holonic algorithms, are close to the optimal solution.

The outline of the remainder of this article is as follows. Section 2 presents the industrial background of the laboratory prototype with its current manual operation. It then describes the physical devices to be used; the desired functionality; and the basic logic control design of the automated solution. Section 3 presents the ideas for holonification, based on the already designed logic control, and Section 4 presents a more detailed description of the AGV system.

2. Application Overview

This section first describes a typical industrial paint-shop operation and the challenges with automating a paint-shop. It proceeds to describe the prototype laboratory system for upload, and the associated devices and subsystems in an operation-oriented sense.

2.1. *Manually Operated Paint-Shop*

A contemporary industrial paint-shop roughly consists of a very long chained trolley conveyor leading painting carriers through the various processing sites of the painting system. The processing steps, taken care of by separated processing plants along the conveyor, are typically of the following kind:

(1) Cleaning
(2) Drying
(3) Painting
(4) Drying
(5) Hardening

At certain stretches along the conveyor, workpieces and painting carriers are handled. The up- and download of workpieces to and from painting carriers is a constant and continuously ongoing process, since each workpiece is to take only one pass through the painting system. The up- and downloading of the painting carriers only takes place when the workpiece type changes, or for occasional maintenance. The "matching"-complexity related to production logistics and planning is described by Williams and Sadakane (1997).

The up- and download processes are typically manual in contemporary paint-shops in SMEs. This is mainly due to a combination of the complex handling, localization of sites on the carriers, and mechanical stability and geometrical precision of the carriers.

2.2. *Automating a Paint-Shop*

The current development goal is the automation of the processes and materials handling at, and surrounding, the workpiece upload process. In the future, the other handling processes may undergo equivalent automation projects in a successive manner.

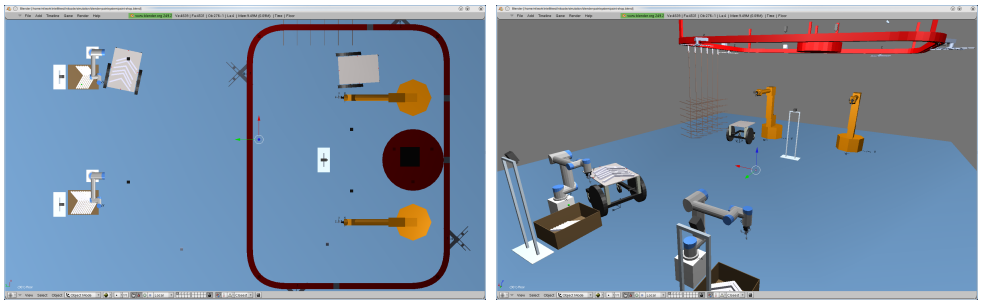
Three challenges for the automation of the immediate process exist:

(1) Painting carriers have low mechanical stability and high geometrical tolerance; much higher than the path-tolerance for mounting workpieces onto the carriers.
(2) The motion sequences involved with the attachment of workpieces onto the carriers are quite complicated; they can contain combined rotation and translation or peg-in-hole types of motion.
(3) Chained trolleys are driven with continuous motion, and the attached carriers are dangling and jumping. The sensory system and the motion-control have to be quite advanced to meet these conditions.

2.3. *Laboratory Application Overview*

A sketch of the planned laboratory system setup is seen in Figure 1.

Referring to Figure 1(a), the following is an overview description of the devices in the



(a) Top view of the laboratory setup. (b) Perspective view of the laboratory setup.

Figure 1. Model of the laboratory setup.

laboratory setup:

- An overhead Power-and-Free (PnF) conveyor is shown as a dark red track in the right side. The "rusty" skeletal structures are painting carriers hanging from trolleys on the conveyor. There are four controllable stops with trolley presence sensors on the track, illustrated by small grey cubes on top of the track. Details of the conveyor may be observed in Figure 2.
- The two large, orange NACHI SC15-F robots are placed enclosed by the PnF conveyor track, in adequate reach of the PnF-stop dedicated for uploading.
- Two Universal Robots UR-6-85-5-A robots, seen on the left, are dedicated to supplying workpieces.
- Two AGVs for transporting workpieces between workpiece supply and upload areas are seen in the setup.
- A 3D stereo vision system is used for localizing upload sites on the painting carriers. The associated cameras are mounted on a ground support near the centre of the PnF-track.
- Several vision systems with cameras around the setup are to recognize workpieces in various (semi-)structured arrangements. The cameras are shown as small black-grey boxes hanging from the (invisible) ceiling or on ground support at the workpiece supply cells.
- Specialized vision systems for global localization of AGVs are associated with small ceiling mounted cameras.

2.4. *Workpiece Upload*

Central to workpiece uploading is a robot which has access to workpieces from an AGV and to a painting carrier at the PnF-stop dedicated for upload. The upload robot picks a workpiece from the AGV and attaches it onto the presented painting carrier. Given the workpiece and painting carrier type, as well as known painting carrier load state, the system will be able to control the robot to attach the workpiece at some free site on the painting carrier.

The painting carriers may have very different geometries and have a load capacity from one to several tens of workpieces. Frequently each painting carrier type may match different workpiece types. Some painting carrier types are adjustable and can host a whole family of workpiece types.

The initial prototype will be using only one workpiece type, and only compatible

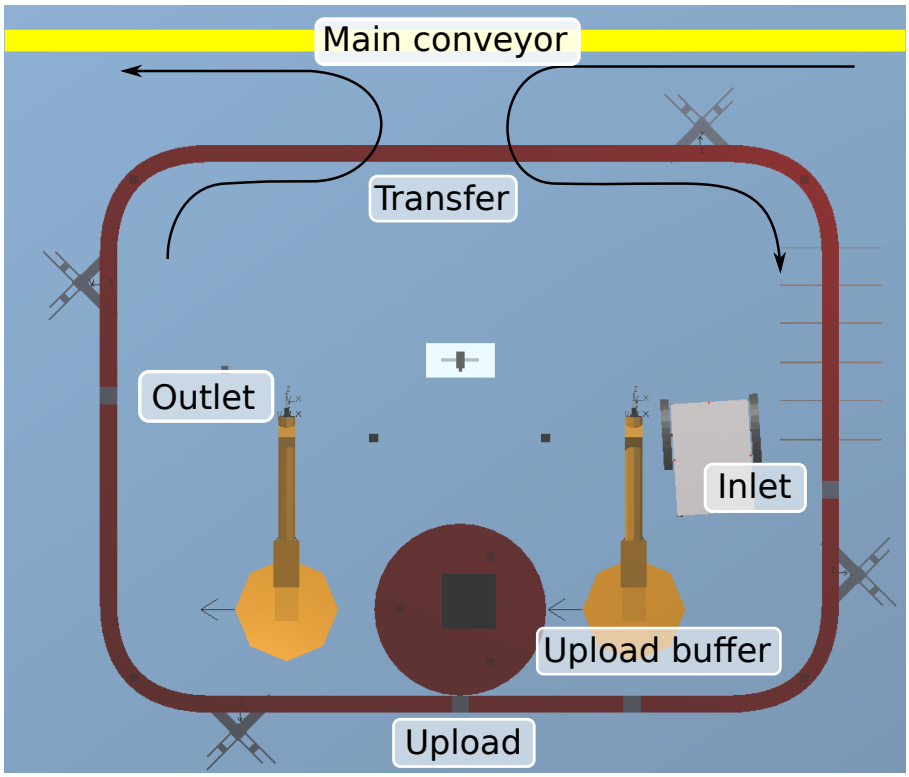


Figure 2. A detailed overview of the PnF conveyor for upload.

painting carrier types. There is thus currently no need for changing painting carriers or changing gripper tools on the robots.

Challenges (1) and (2), described in Section 2.2, have been addressed satisfactorily for two workpiece types with associated (different) painting carrier types.

2.5. *PnF Conveyor System*

As mentioned earlier, it is necessary to address the problems posed by the typical use of chained trolley conveyors in industrial paint-shops; challenge (3) in Section 2.2. To this end, the solution is to place a PnF conveyor in conjunction with the main painting system conveyor, and to transfer carriers as they arrive into buffering for upload. See Figure 2.

The PnF conveyor set up in the laboratory has four pneumatically controlled stops, each with a capacitive sensor for trolley presence. It also has four mechanical rotator units, that rotates the attachment point of the trolley by 90° on passage. The effect of the rotators is to ensure that the carriers face the correct direction at all times.

An embedded computer interfaces with the valves controlling the PnF-stops and the presence sensors at the stops. It exposes basic functionality over Ethernet for external control of the stops and reading of the sensors.

The PnF-stops, symbolized by small grey boxes on top of the PnF-track, are labelled according to their function in Figure 2. The “Inlet” and “Outlet” stops serve as buffering from and to the transfer at the junction with the main conveyor. The “Upload”-stop is

where the PnF-trolley is stopped for upload to the mounted painting carrier. The speed of the PnF-carrier is quite low, in the order of $10cm/s$, so to speed up the exchange of carrier at the "Upload"-stop, an "Upload buffer"-stop is placed upstream close by.

Overall, the conveyor elements must be controlled to ensure that carriers are supplied to the upload robot, and such that the transfer area receives carriers at the correct pace.

2.6. *Workpiece Supply*

Workpiece supply cells are where workpieces are picked by a robot from gross storage boxes and put into arrangement on top of an AGV. The AGV may be equipped with a workpiece-compatible transport carrier, which could be offloaded at the delivery point.

In case the robot is to pick workpieces from gross storage boxes, a semi-structure arrangement may be generally anticipated, and some sensory system for localization must be deployed.

The robot must have the workpiece storage and the docked AGV within reach. The robots used are small, light-weight industrial robots. It may happen that the area on top of the AGV, available for workpiece arrangement, is too large for the reach of the robot. In that case, the AGV must be able to re-dock with a different pose on request, enabling access to unreachable areas.

2.7. *AGV System*

The role of the AGV system is to undertake the resupply of components for upload. The AGV system thus enables the geographical separation of the bin-picking operation from the uploading operation.

A concrete challenge for the transport system is that it will not be possible to reserve pathways in all areas with AGV operation. There are no static paths nor static nodes and the pathways cannot be guaranteed to be free of obstacles. This uncontrollable, production environment requires advanced obstacle-recognition and -avoidance capabilities. This fact, as well as the importance of flexibility in this project, has driven the focus towards free-roaming AGVs.

Multiple AGVs are being built at the laboratory and the control system is in the experimentation phase. The physical AGVs and control system is based on earlier experience with a prototype AGV and vision based positioning system.

Section 4 is dedicated to a description of the developed AGV system.

3. Holonification

Initial work on the laboratory setup experimented with a hierarchic, distributed control system for managing the upload process. This section concentrates on the conceptual holonification of the devices and resources presented in the preceding sections. Ideally, the explicit logic control behaviour of the hierarchic control system will emerge from the holonic control system currently under development.

3.1. *Resource Holons*

The resource holons are related to the set of devices discussed in Section 2. This section presents some aspects of the resource holons in relation to devices.

3.1.1. Robots

For ordinary industrial robots, the real-time aspects of the motion control is integrated with other real-time tasks inside the controller, limiting the interaction from an external system. This design prevents a "deeply holonified" system; see Figure 3(a). Owing to communication and computing equipment standards some decades ago, the integrated controller design made good sense.

One problem with the integrated robot controller architecture generally arises with sensor-driven motion control. The designers of the integrated controllers used some scenarios for some sensors to affect motion control. The robot controller platform hence forces the application designers to limit the use cases of the robot system.

Some modern or experimental robot controllers support external motion control. This is the case for the robots used in the presented work, and it enables the deeper, and more flexible holonic-level integration presented in Figure 3(b).

Two different robot systems are planned in the initial laboratory system. One type is the system around the central upload robot and another around the workpiece supply robot.

3.1.2. Workpiece Supply System

The workpiece supply cells transform workpieces from batch containers into a pickable arrangement in a transport carrier. One of the major challenges is the workpiece localization in the containers. In most cases, a semi-structured arrangement of workpieces can be anticipated, for which ordinary 2D-vision can be used for localization.

The direct interaction of this system with the paint-shop system is through the orders and schedulers. Product holons for workpieces play a role in identification, localization, and graping. Product holons for the transport carriers play another role of specifying the arrangement of workpieces into the carrier and possibly some process information for insertion.

3.1.3. Tools

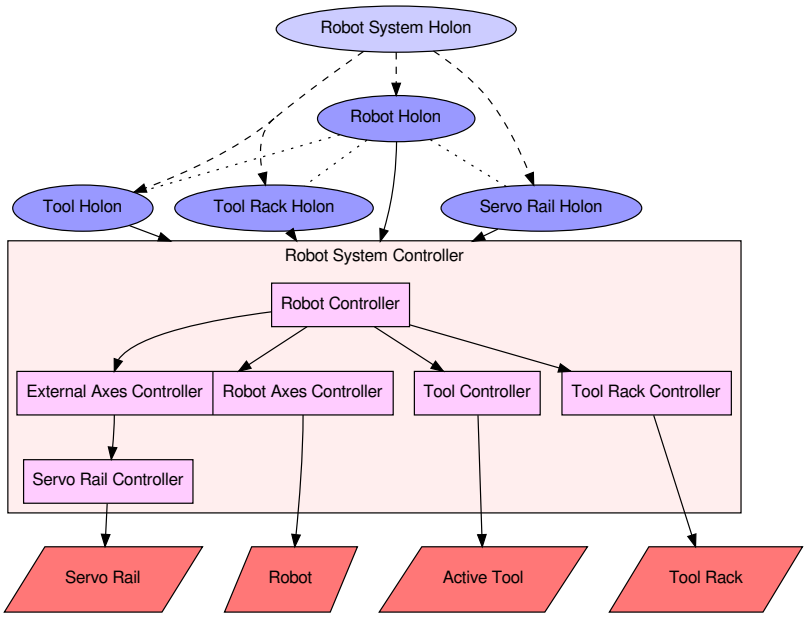
The tools in the presented system are grippers, mountable on a robot end effector, for handling workpieces. The entire set of tools available to one or more robots must cover the whole range of workpieces to be handled by the robots.

In case of multiple tools for a single robot, or shared among several robots, the tools may be organized in a tool rack to be on-line changeable by the robot system. There will be a tool holon for each tool, but typically only the ones mounted on a robot at any given time will be active. In case of a multi-function tool, each functionality may be modelled as a separate resource holon.

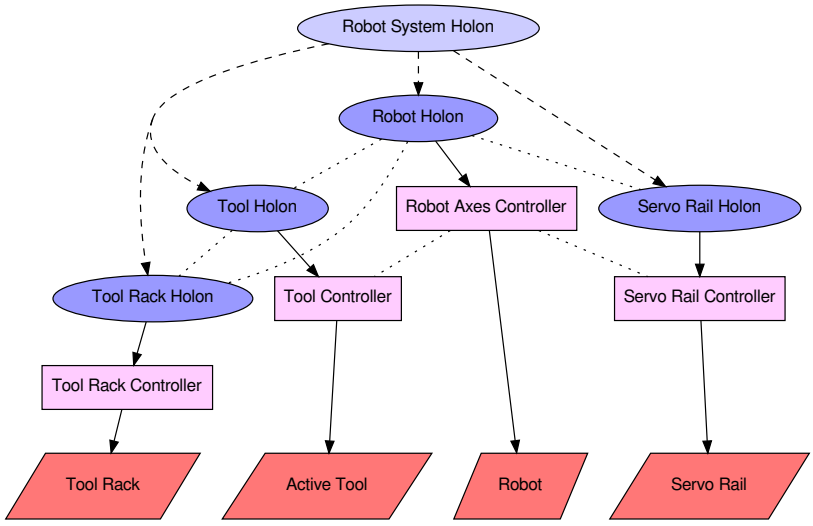
3.1.4. Vision Systems

The different types of vision systems are :

(1) *Painting Carrier Analyzer:* This is a commercial 3D stereo vision system, which has been configured and tested. It analyzes for empty sites on a painting carrier presented to its cameras. This information is used by the upload robot system.
(2) *Workpiece Localizer:* Localizes the workpieces for picking, either semi-structured in batch boxes or in structured layout.



(a) Centralized robot control system in vendor controller.



(b) Distributed holonic robot control system.

Figure 3. Traditional integrated and holonic robot systems in automation. Legend: ovals: holons, rectangles: controllers, and parallelograms: physical devices.

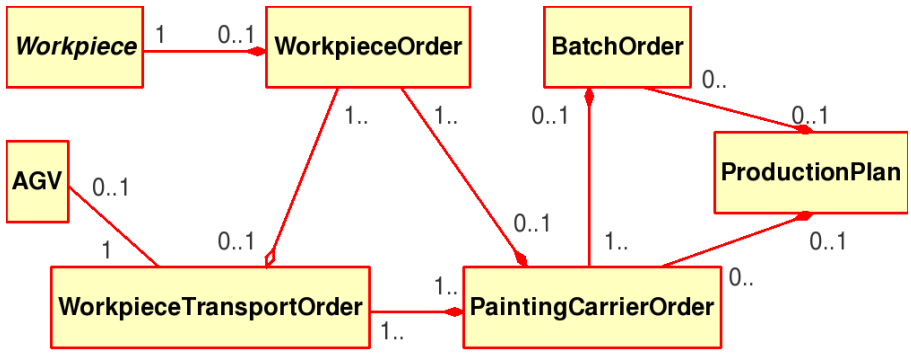


Figure 4. Overview of aggregations and compositions among the order holon classes.

(3) *AGV Localizer:* Vision systems specialized in localizing LEDs on AGVs, giving sufficient information to the subscribed AGVs to compute their position and orientation.

The integration of the vision systems into holonic vision applications is a matter of letting orders interact with them in various ways, such as reserving, configuring, querying, and subscribing.

3.1.5. PnF-Conveyor

The PnF-conveyor was described to some detail in Section 2.5. The objects for holonification are the stops and their associated presence sensors.

The stops and sensors are modelled as individual holons. Since there is only one integrated controller for all the stops and sensors, the system will resemble the integrated robot system in Figure 3(a). However, since there are no hidden real-time interaction among stops and sensors, the integrated control for the PnF-conveyor will not impact the holonic control system use cases.

3.2. *Order Holons*

The production-orders in the factory contain the data relevant for the associated top-level holonic orders. Orders are modelled as a hierarchy where the highest level is a production day-plan. In contemporary manufacturing systems in SMEs, the production day-plan has been determined over night, or even some days in advance, by the ERP system.

In a typical paint-shop in an SME, the sequencing and scheduling of the production plan is done heuristically by the operators at the paint-shop. They perform the implicit break-down of the day-plan into batch orders, painting carrier orders, stock reservation orders, transport orders, as symbolized in Figure 4. Such funcitonality and logic is what must be implemented in, and executed by the order holons.

As a general rule in HMS, it is the order holons that take all higher level initiatives in the control system.

3.3. *Product Holons*

End-products are naturally represented in the product holon system. They are the items exposed for ordering, from customers, retail stores, importers, or the company sales de-

partment. Elements that will be represented in a end-product holon would comprise usual manufacturing information such as order number, bill of materials, assembly sequence, machining process information, painting process information, etc.

Products that has constituent parts, indicated by bill of materials and assembly sequences, have reference to the pertinent product holons for the constituent parts.

Product holons may also represent something more abstract. The following are examples of information that may be modelled as "abstract" product holons, relevant to the paint-shop case:

- Arrangement of workpieces in transport or on painting carriers.
- Process information regarding grasping and attachment process motion.
- Vision analysis configuration and result specification.
- Transport related specifications like routes, deadlines, docking poses and tolerances.

3.4. *Staff and Supervisor Holons*

Staff holons are candidates for implementation of complex, global, and long-running computations. The basic holons will not be suitable for realizing such functionality, since they must remain focused on their core, short-term objectives.

Planning, sequencing, scheduling, prioritizing, and global monitoring of tasks may be left as an emerging functionality from the cooperating order and resource holons. However, this may be severely far from optimal, and, in complex, cases even unsolvable. Staff holons may be deployed to perform such global computational tasks, and provide results to basic holons that request it.

Where the staff holons from PROSA are lacking any kind of controlling authority, the supervisor holons from ADACOR was designed to meet the practical need for coordination. Supervisor holons may be used to create or destroy elementary holons, and to control aggregation of holons. They can further be used to enforce results from staff holons, such that order and resource holons will be coordinated to obey certain sequencing or prioritization schemes.

4. Holonic AGV System

The AGV system has been developed based on the published literature on holonic manufacturing and material handling system. The system is an implementation of the PROSA architecture and is in several respects inspired by the work done by Srivastava *et al.* (2007) and Babiceanu *et al.* (2004).

4.1. *Overview*

The holonic AGV control architecture is exemplified by an object diagram in Figure 5. Resource holons are represented by rectangles, order holons by ovals and staff holons by hexagons. It shows independent AGV holons and AGV holons aggregated in an AGV-group. It also shows a path-planning holon inside an AGV holon, although this is not a requirement; path-planning holons can be part of AGV holons or shared.

A requirement to the AGV system is to be integrable into the manufacturing system. In addition to allowing the manufacturing system to create transport orders, integrability is currently implemented by allowing elements from the holonic manufacturing system

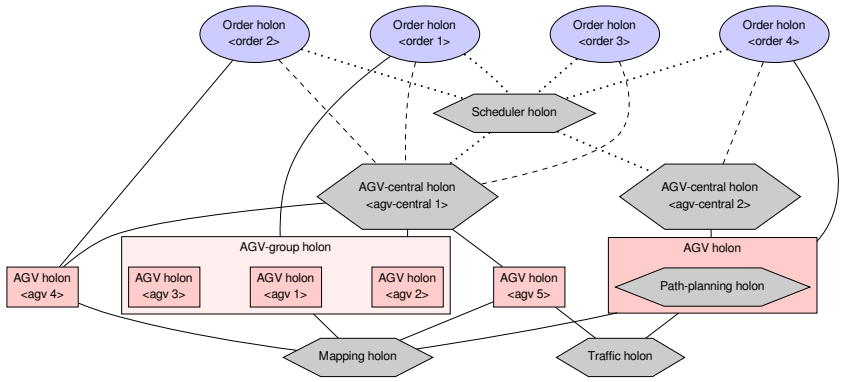


Figure 5. Example object diagram for the holonic AGV system.

to reserve an AGV for a period of time. Thus enabling real-time cooperation between an AGV and other devices of the production system. Realistic application examples are:

- Efficient geometric separation of cooperating robots, like the functioning of a turntable.
- Following a robot on a servo-track.
- Precise control of AGV positioning.

4.2. *Holons*

A short description of selected holons from the AGV system is presented in the following subsections.

4.2.1. AGV Holon

The AGV holon is the resource holon administrating the physical AGV. It is assumed to run on the physical AGV; although this might depend on the available hardware in the pertinent case.

The internals of an AGV holon can be categorized into the following:

- An aggregate of internal holons when the system runs on an open AGV controller. The internal holons may be exposed to the rest of the control system but it may not be the case if the controller is implemented using, for example, the Robot Operating System Quigley *et al.* (2009) where each ROS-node can be seen as a holon.
- A wrapper around a commercial AGV controller. In this case AGV holons might even be entities that are limited to monitor and send orders to a completely closed commercial AGV system, resulting in an AGV holon with low flexibility.
- An AGV-group composed of AGV holons in slave state. Cf. Section 4.2.6 regarding AGV groups.

Since different AGV types result in different features, the AGV holon needs to expose its capabilities to the rest of the system.

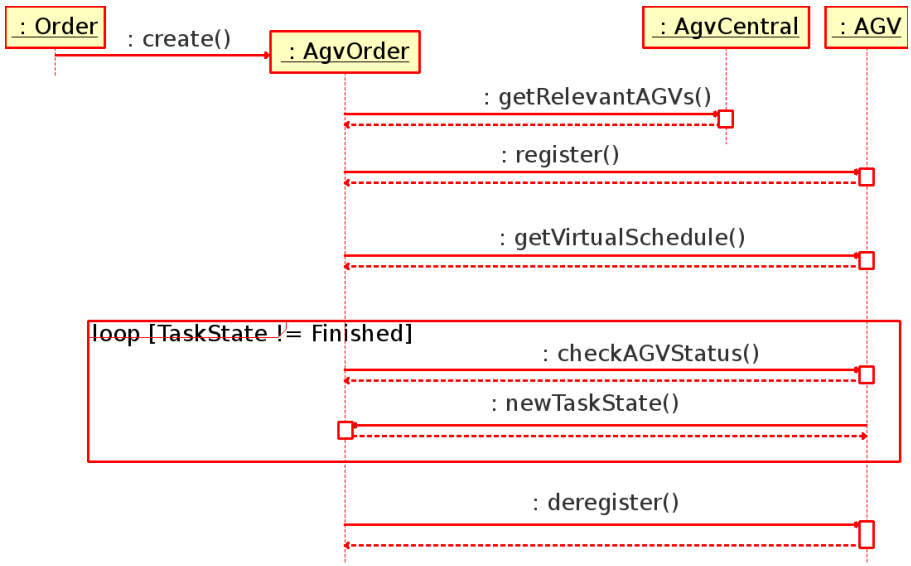


Figure 6. Sequence diagram showing an AGV order holon registering with an AGV holon.

4.2.2. Order Holon

An order holon in the AGV system is created by a generic manufacturing order holon to solve a transport task.

Following the PROSA architecture, the order holons are the driving force of the holonic system; cf. Figure 6. The order holons monitor the AGV system through the AGV-central holons and attempt to (re-)assign themselves to the AGVs with the most advantageous schedules.

4.2.3. Scheduling Holon

Industrial partners often desire central, or at least deterministic, algorithms for critical functions like scheduling. A possible implementation of such critical functions is to use a unique central scheduling holon or several holons implementing the same algorithm. The algorithm must be deterministic so that holons eventually will end up with the same conclusions. This requires a system that is resilient to temporary disagreement.

The scheduling holon is a staff holon providing scheduling data to other holons. It is a necessary complement to local scheduling information from the AGV holons, since the latter have only local insight into the manufacturing system.

By communicating with other holons, a scheduling holon keeps an up-to-date view of the manufacturing system, generates an "optimal" schedule and applies it by trying to re-assign order holons to AGVs.

4.2.4. On-line Traffic Controller Holon

This is a staff holon, the role of which is to survey the AGV traffic in a geographical area and give advices to AGV holons to sort out conflicts and avoid congestion.

The traffic system gathers information from and supplies information to the client AGVs about viable paths in the whole of the roaming area. It will be the coordinator for pre-reservation of long term trajectories of the AGVs, whereas the AGVs themselves can react in the short term; such as emergency trajectory interferences.

Specialized intersection controllers at known trafficked or congested areas are regula-

tory rather than guiding in their relation with the approaching AGVs.

4.2.5. Geography and Vision Holons

Initially it is assumed that all of the AGV localization functionality will be based on vision applications. This soundness of this decision is substantiated by earlier success with developing a vision based AGV localization system.

The geography system is the general global localization system for the AGVs. It requires that the cameras of the geographic localizer servers cover the entire area where the AGVs may roam, and can serve the client AGVs with real-time location information.

Specialized conceptual controllers for local positioning and control may be defined to serve purposes relevant to situations or tasks, partly external to the AGV system. Examples relevant to the laboratory painting system are docking-, trajectory-, and passage-controllers.

4.2.6. AGV-Group Holon

The AGV-group holon applies the holonic fractal concept to aggregate several AGVs to a special task. An AGV-group can appear to the external holonic environment as a specialized AGV holon.

The constituent AGV holons in slave state let another AGV take over the high level motion control. They do not reply to requests as individuals but through the group holon. Example applications are temporary palette conveyors, large object transport and high priority transport chains.

5. Conclusion and Discussion

This paper has presented an automated paint-shop under development. The need for a flexible solution has been identified and a design and an implementation based on the holonic manufacturing paradigm has been proposed. A lightweight holonic framework has been implemented and many hardware devices, including sensors, the PnF-conveyor, AGVs and industrial robots are now exposed to the control system as holons. A low-level emulation platform is also under development and the prototype AGV system is running on it.

Implementation of the proposed holonic control system will continue during the next years in cooperation with the industrial partners. Further development and implementation of hardware and device-specific control software are taking place in parallel. Much work has yet to be done to evaluate the performance, scalability and flexibility of the proposed control system and demonstrate its industrial feasibility.

At the current state of design of the paint-shop prototype, we have gained such confidence in the HMS paradigm, that it will be used for the prototype, and recommended as a good candidate to the industrial partners in the project.

6. Acknowledgements

This paper presents research results obtained through work in the IntelliFeed project, financed by The Research Council of Norway. Thanks to professor Terje Lien, *Department of Production and Quality Engineering*, *NTNU*, for general discussions on manufacturing an for taking an interest. Thanks to Amund Skavhaug at *Department of Engineering*

Cybernetics, *NTNU* for many supportive consultations on distributed systems and computing. Our gratitude for good cooperation goes to the IntelliFeed group at *SINTEF Raufoss Manufacturing AS*.

References

Babiceanu, R.F., 2005. Holonic-based control system for automated material handling systems. Thesis (PhD). Industrial and Systems Engineering, Virginia Tech. 1

Babiceanu, R., Chen, F., and Sturges, R., 2004. Framework for the control of automated material-handling systems using the holonic manufacturing approach. *International Journal of Production Research*, 42 (17), 3551–3564. 4

Davis, S., 1987. Future Perfect. *Reading, Massachusetts/USA*. 1

Koestler, A., 1967. *The Ghost in the Machine*. London : Hutchinson. 1

Leitão, P., 2004. An Agile and Adaptive Holonic Architecture for Manufacturing Control. Thesis (PhD). Department of Electrotechnical Engineering, Polytechnic Institute of Bragança. 1

Leitão, P. and Restivo, F., 2005. ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry*, 57, 121–130. 1

Leitão, P. and Restivo, F., 2008. Implementation of a Holonic Control System in a Flexible Manufacturing System. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38 (5), 699–709. 1

Liu, S., *et al.*, 2000. Holonic manufacturing system for distributed control of automated guided vehicles. *In*: *2000 IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 3, Nashville, TN IEEE, 1727–1732. 1

Quigley, M., *et al.*, 2009. ROS: an open-source Robot Operating System. *In*: *Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*. 4.2.1

Srivastava, S., *et al.*, 2007. Development of an intelligent agent-based AGV controller for a flexible manufacturing system. *The International Journal of Advanced Manufacturing Technology*, 1–18. 1, 4

Van Brussel, H., *et al.*, 1998. Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37 (3), 255–274. 1

Van Brussel, H., 1994. Holonic Manufacturing Systems The Vision Matching the Problem. *In*: *Proceedings of the 1st European Conference on Holonic Manufacturing Systems, Hannover, Germany, IFW-Hannover*. 1

Williams, E.J. and Sadakane, S., 1997. Simulation of a Paint Shop Power and Free Line. *In*: *Winter Simulation Conference*, Atlanta, GA IEEE Computer Society, 727–732. 2.1

Zipkin, P., 1997. The Limits of Mass Customization.. *Harvard Business Review*, 75, 91–101. 1

4.9 Using the Blender Game Engine for Real-Time Emulation of Production Devices

Morten Lind and Amund Skavhaug. Using the blender game engine for real-time emulation of production devices. *International Journal of Production Research*, 0(0):1–17, 2011. ISSN 0020-7543. doi: 10.1080/00207543.2011.601772. Online available, iFirst

Declaration of co-authorship

The basic idea of using the Blender game engine for real-time emulation of production devices and production setups is credited to Morten Lind. The entire software system analysis and design, and development of the principles and mechanisms for obtaining realistic device emulation is credited to Morten Lind. The entire implementation of the demonstration system is a contribution by Morten Lind. All experiments and performance testing was carried out by Morten Lind.

Olivier Roulet-Dubonnet contributed with discussions and comments from using the Blender game engine and the presented principles on an AGV systems emulation. Amund Skavhaug contributed with general discussions on distributed control considerations during the development of the demonstration system and principles.

The paper is entirely written by Morten Lind. All graphics is designed and produced by Morten Lind. The preparation and submission process was handled by Morten Lind. Amund Skavhaug contributed valuable comments in frequent correspondence during the writing of the paper.

International Journal of Production Research
Vol. , No. , 2011

RESEARCH ARTICLE

Using the Blender Game Engine for Real-Time Emulation of Production Devices

Morten Lind[a*] and Amund Skavhaug[b]

[a]*Norwegian University of Science and Technology, Dept. of Production and Quality Engineering, 7491 Trondheim, Norway*

[b]*Norwegian University of Science and Technology, Dept. of Engineering Cybernetics, 7491 Trondheim, Norway*

(*Received 00 Month 200x; final version received 00 Month 200x*)

This paper describes principles, architecture and design details for using the Blender Game Engine in real-time production device emulation. An emulation system for a real material transport and handling production installation was implemented based on this. A prototype of a distributed production control system was run on top of the device emulation system to evaluate feasibility.

A cardinal architectural principle is the clear distinction between production controllers and production devices. This principle, applicable through a flexible communication middleware, enables the implementation of portable production controllers, which execute transparently on either the emulated or the real production system.

Production system engineers may take advantage of this approach, to develop and gain confidence in complex production control solutions.

Keywords: Simulation; Distributed manufacturing control; Shop floor control; Robot applications; AGV.

1. Introduction

Distributed, autonomous production control systems, such as of a holonic or agent-based nature, have the potential to provide much higher flexibility, intelligence, and error recovery than a centralized system (Valckenaers and Van Brussel 2005). However, this comes at the price of a higher development threshold and difficulty of debugging in the run-in phase. Hall *et al.* (2005) discusses more specific obstacles to the use of agent technology

*Corresponding author. Email: morten.lind@ntnu.no

ISSN: 1366–588X print/ISSN 0020-7543 online

in production control system.

Whether developing a control system for a new production installation or for an existing one, there will often be limited availability for experimenting on the real system hardware. In case of a new system with expensive hardware, the more central and expensive components will probably not be ordered until a control system is designed and validated. In case of the development of a new control system for existing production installation, the installation will probably not be recommissioned, until the new control system has been accepted.

Providing true emulators of the production hardware entities (machines, native controllers, devices, sensors, etc.) such that they can co-exist and operate in a virtual real-time world, is a large step towards developing a production control system for unavailable hardware; especially so for an agent-based control system.

1.1. *Simulation Overview*

When faced with the need of simulation for verifying or develop production control systems, it is traditional to consider *Discrete Event Simulation* (DES). A plethora of software applications and platforms exist for the purpose of simulation for manufacturing; e.g. DELMIA QUEST, FlexSim, and PySim. Smith (2003) presents an extensive classification and survey of DES in the field of manufacturing. However, DES is by its nature more suitable for simulating production control systems characterized by determinism. This is a severe restriction on control system design; especially so, when hoping to honour the request for flexibility by intelligence at the production control level.

Moon *et al.* (2006) presents a example of a combined use of a DES system, QUEST, and a robot simulation studio, DELMIA IGRIP. The body shop in an automotive factory is modelled, simulated, and analyzed. Each workstation is modelled, simulated, and optimized in detail using IGRIP, producing offline generated robot motion specifications. The resulting visual model for each workstation is then transferred to QUEST, together with resulting operation and process timings. A grand model of the whole body shop is set up in QUEST, with underlying data for all detailed models. The logic for flow of parts, buffer control, and overall logistics can then be implemented, simulated, analyzed, and optimized. This is a quite traditional separation of process control from production control. It is a time-efficient method for developing factory layout, process logic, and production logic for large batch size production. However, the method loses efficiency for simulating a production system characterized by low or single piece batch size; such as mixed or chaotic production. non-deterministic processes, or autonomous control entities.

DELMIA V5, like QUEST, is aimed at simulation, validation, and optimization of the control logic in automated production (Caie 2008). While QUEST may model the production setup and simulate an implementation of the logic of the production control system, V5 takes further steps towards realistic interaction of the modelled control system, providing realistic virtual integration of the programming aspects of PLCs, sensors, and device controllers. This amounts to a validation at a deeper level of a production control design, based on a library of many frequently used types and brands of production automation devices, such as PLCs, sensors, and device controllers. V5 aims at a maximum possible level of realistic validation in production control systems based on traditional control paradigms rooted in deterministic logic and with standardized control equipment. The work presented in this paper aims at goals of similar nature, but for paradigms of production control based on distributed autonomy and intelligence at all levels in the control system. While V5 may get close to formally validating the logic

operation of deterministic production control systems, realistic real-time emulation and simulation of a control system allowing massive distribution and autonomy may yield only a qualitative and verification.

A distinctive difference which separates agent-based control systems from more traditional ones is that they are characterized by emergent rather than deterministic behaviour (Vrba and Marík 2005a). It is impossible to pose a realistic, deterministic model for a production entity controlled by a truly autonomous agent. Such a deterministic model would be a necessity for the scheduling of events in a standard DES. Real-time, realistic emulation at the device level is thus necessary, since there is no essential logic to model for a distributed, autonomous control system; there is only the actual control code to execute on the real-time emulation.

In distributed and autonomous production control, it is quite easy to argue for the advantages of time-driven, realistic simulation. Vrba and Marík (2005b) uses the term *agent-based simulation* for this type of simulation. An agent-based simulation system was described by Marík *et al.* (2005), and is a software-platform or -framework for developing and modelling an agent-based control system, and deploying it in the system of emulated production devices.

1.2. *Related Work*

Kim *et al.* (2000) implemented virtual machines for simulation of shop-floor control and operator training. They emphasize on the need for real machine emulation, providing an operator or the shop-floor control system with the realistic or real interface. However, the implementation of the virtual machines are based on the Java Web Server, Java Servlets, and communicate over HTTP/XML, which does not add up to a fast response, real-time system. Though functionally giving the appearance and true emulation of a machine, the whole range of machines requiring even soft real-time control will be excluded by such platform and communication technologies.

Park *et al.* (2009) presents an overview of principles for verification of PLC programs and commercial software for simulating production control. Though not strictly limited to PLCs, their work is focused on simulation-based verification methodology for PLC programs. The suggested method describes how to build state machines for emulation of the input-output of all production devices connected to the (real or simulated) PLC. By analyzing the logged state changes and IO-signalling, it can be established whether the PLC program satisfied the specified behaviour or if it must be improved. This resembles the methods of the present paper, but rather than building a pocket of IO-emulation for an isolated PLC, the present work tries to build a physically realistic model of all devices to be controlled by the entire (distributed) production control system. By suitably providing interfaces in software for the simulated PLC, or hardware for the real PLC, it may in fact be hosted on an emulation system based on the presented work, as part of the production control system. A further distinction between the presented work and the work of Park *et al.* (2009) is the emphasis on qualitative and quantitative verification, respectively. When using the real hardware PLC with the real production or process control program, executing with the emulated device IO state machines, this is a very thorough test of the isolated operation of the PLC program. In contrast, the approach presented in this paper may suffer from computational exhaustion in the device emulator. Additionally, the deployment systems and topology of the production control in the present work may not be the same in real-time simulation as the systems and topologies used in the production system.

Pannequin and Thomas (2010) describe *Emulica*, an architecture and implementation of a system for emulation of the production operating system on which the production control system can execute. Their framework focus on the process and flow control where parts are of transformed into products; like assembling and disassembling processes. Their emulation is at the abstract process level, providing an interface with methods of the nature of "Do the assembly" or "Move the part". Their framework is not concerned with controlling the devices and machines, performing the requests, and the related real-time aspects. In contrast, the work presented here achieves emulation of the real-time, geometric, mechanical, and control aspects of production devices.

1.3. *Outline*

The remainder of this paper is organized as follows: Section 2 gives an overview of the basic software tools and technologies used. Section 3 gives an operational overview of the demonstration system for which the emulation system was built. In Section 4 the architectural design of the implementation is explained. Section 5 presents some design and implementation details. Discussion and conclusion is presented in Section 7.

2. Tools and Technologies

The main software platforms for achieving the emulation system has been Blender 3D for modelling and its game engine as execution platform; the Ice™ communication middleware; and the Python interpreter. This section gives an brief introduction to these third-party software systems on which the current implementation rely.

2.1. *Implementation Overview*

The implementation of the systems for emulation and simulation is done entirely with Python[1]. The implementation is quite large and Python is a good choice as a code-efficient and intuitive language.

The separate systems that have been implemented and integrated as part of this work are[2]:

- *Math3D* (∼ 1000 lines of code) is a basic Euclidean mathematics library for working with positions, vectors, orientations, rotations, reference systems, homogeneous transforms, and linear interpolations.
- *PyMoCo* (∼ 2000 lines of code) is a motion controller framework implementing kinematics and various robot motion controllers, supporting real-time code and sensor interaction. The framework was developed and tested by Lind *et al.* (2010).
- *Emulation System* (∼ 2000 lines of code) is the code layer for all devices that can be part of a model of a production installation. For a given modelled device, a controller is implemented in the emulation layer, and the emulated controller must recognize the device configuration from the model. Any interactive device system is represented over the production control network by at least one Ice-server.

[1] http://python.org
[2] Generated using David A. Wheeler's 'SLOCCount'; confer http://www.dwheeler.com/sloccount/.

- *Production Control System* ($\sim$ 2000 lines of code) is the software layer of application control of the production installation; be it emulated or in reality. It is the objective of the production control system to operate the devices, cooperating among themselves.

2.2. *Middleware and Distributed Framework*

The choise of network communication and connectivity, i.e. the middleware, for a distributed control system is a central issue; especially so for one with real-time requirements.

The chosen middleware and distributed control framework was developed by Olivier Roulet-Dubonnet, and will be described in a later publication. The software is available for download and review from the *ColdHMS* project page on SourceForge[1]. The framework is called *IceHMS* in reference to the Ice™ (*Internet communication engine*) middleware and *Holonic Manufacturing Systems*. The reference to HMS is due to the original inspiration for the framework.

Ice™ is well motivated (Henning 2006, 2007), well described (Henning 2004), and well documented[2]. Good examples, performance towards the limit of no overhead[3], high availability, and clear semantics and syntax made Ice™ an favourable candidate.

A determining factor for choosing Ice™ is the cross-language and cross-platform properties. This will allow separate developer groups to stick with the platform and language they favour, or chose the platform and language best suited for their pertinent system implementation.

2.3. *Blender Game Engine*

Blender[4] is a (mesh) modelling and animation software studio. It comes with a quite advanced and efficient game engine, and integrates the Bullet physics library[5].

Through the embedded Python interpreter, the game engine is wide open for implementing real-time interaction of external control software with the internal game engine logic. The game physics may be utilized whenenver some mechanical part is under uncontrolled motion, like sliding or falling.

3. Demonstration System

The prototype production system, around which the demonstration emulation and simulation system is designed and implemented, is described by Lind *et al.* (2009). A preliminary status of the implementation was presented by Lind and Roulet-Dubonnet (2010).

The demonstrator is a realistic case of the logistics around production painting systems. The painting system consists of a chained trolley-conveyor, carrying workpiece carriers through the sites of the processes; such as washing, drying, painting, heating, and hardening. At certain areas along this main conveyor the up- and downloading of workpieces, and the occasional change of carriers take place. Workpieces are up- and downloaded

[1] http://sourceforge.net/projects/coldhms/
[2] http://zeroc.com/download/Ice/3.4/Ice-3.4.1.pdf
[3] http://zeroc.com/articles/IcePerformanceWhitePaper.pdf
[4] http://blender.org
[5] http://bulletphysics.org

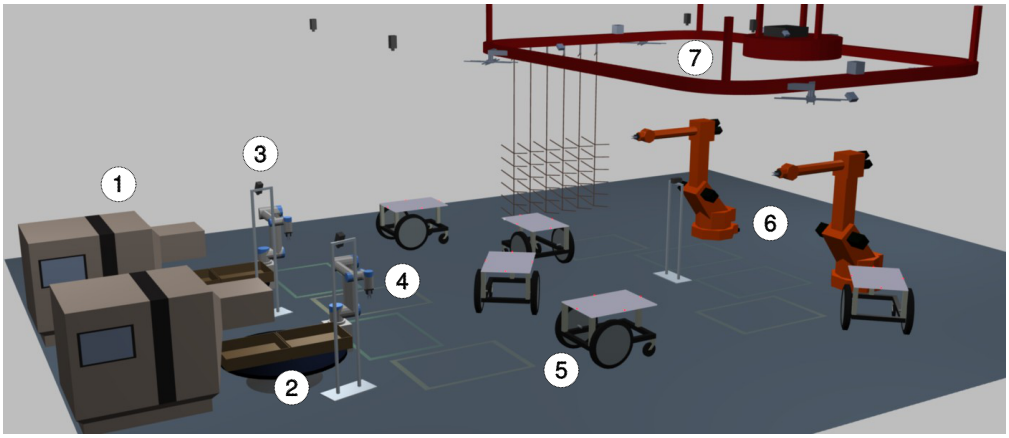


Figure 1. Rendered image from the Blender 3D model of the demonstration system.

for natural reasons, since every workpiece is to take only one tour of the painting system. Carriers are switched whenever a changeover rolls in a workpiece type with which the present carrier type is incompatible. The implemented demonstrator system only concerns uploading of workpieces.

The demonstration system is an extended version of the prototype system and is seen in Figure 1. In the order of flow of workpieces in the production system, the following gives a description of the mechanical devices and their operation.

(1) Workpiece machining, such as bending, turning, milling, extrusion, cutting, etc., is performed to single-piece order, in the CNC-machines at the far left.
(2) From a CNC-machine, workpieces drop from the outlet into a box fixed on a turntable. The turntables have two opposing boxes, one servicing a CNC-machine and the other servicing a robot.
(3) A vision system observes the box on the turntable-position that services the robot. As long as workpieces are localized in the box, the robot system can chose one and pick it.
(4) The robot opposing the CNC-machine at a turntable serves to arrange the workpieces into a specified layout on a docked AGV.
(5) The AGVs are made with a flat top-plate covered with a non-slip material suitable for workpieces to rest freely and steady onto during transport. They are able to dock into the faintly marked (yellow) buffer and (green) task areas on the floor, near the robots.
(6) On the far right, two robots are placed for handling workpieces off the AGVs and attaching them onto painting system carriers on the overhead Power-and-Free (PnF) conveyor.
(7) The PnF-conveyor is a decoupling installation from the main painting system conveyor; the latter is not modelled in the demonstration system. Empty carriers are transferred from the main conveyor to the PnF-conveyor and filled carriers the opposite way. A set of PnF-stops and associated presence-sensors enables the conveyor control system to manage the PnF-trolleys.

To give an impression of the resulting operation of the implemented demonstration system, some short video clips have been made. They may be downloaded in Xvid and Ogg Theora formats from `https://automatics.no-ip.org/~ml/intellifeed-emu/`.

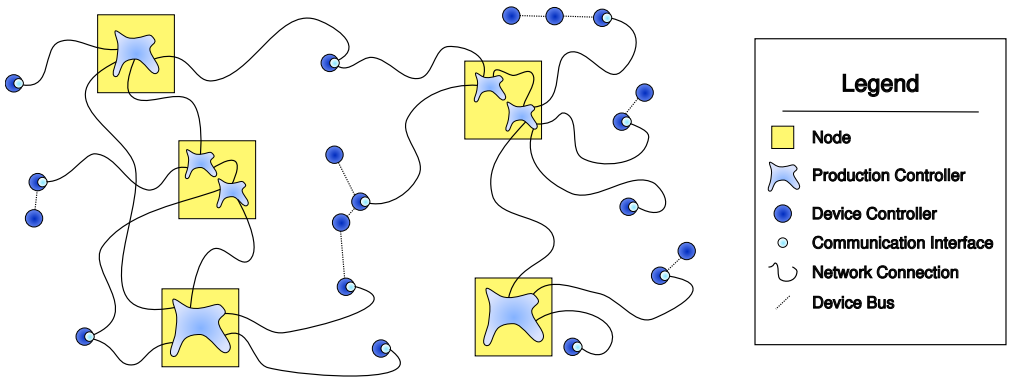


Figure 2. Example of connectivity in a distributed control system.

4. Emulation System Design

This section gives an overview of the architecture and design of the emulation system, and the technologies used for implementing it.

4.1. *Architectural Overview*

Conceptually the device controllers of the production system needs to be encapsulated in an emulation platform, and exposed appropriately to the production control system for simulation. An overview of the structure of a distributed control system, and how it is wrapped up for being emulated, is given here

4.1.1. Distributed Control Systems

Figure 2 shows an abstract view of a distributed production control system. The only concepts represented are control nodes, production controllers, device controllers, and connections. In addition there are two kinds of communications of different nature.

Most device controllers present a general communication interface, through which the production controllers may connect. The same kind of connection enables production controllers to interconnect for coordination and cooperation. Some device controllers are naturally distributed, and they communicate by some closed network or bus. This latter connection type is not of particular interest when developing a production control system, but this internal device controller network may be important when emulating the device.

There are no general constraints on the network connection topology. The specific types of production controllers to connect is naturally limited by semantics and types. This is determined by the setup in a production system, where the organization into factories, plants, lines, stations, and cells will play a large part in the particular connectivity and partitioning of the control nodes and production controllers.

4.1.2. Emulation and Simulation System

The objective of creating a system of emulated device controllers for the production controllers, shown in Figure 2, has been achieved with an architecture based on the Blender game engine. Figure 3 shows this architecture with the game engine residing on some central node. All device controllers are emulated in the embedded Python interpreter in the game engine. The embedded Python interpreter has access to the virtual reality of all the Blender objects modelled in the scene.

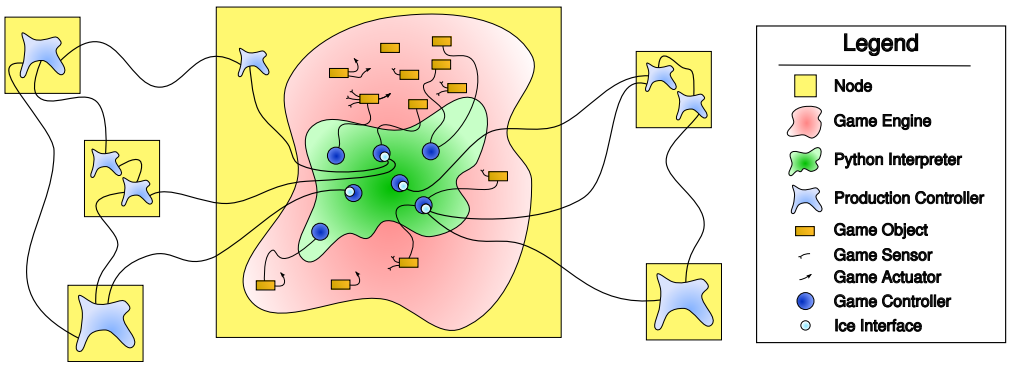


Figure 3. Overview of the architecture and connectivity in an emulated system.

Due to the versatility of the Python interpreter, the liberty given to it by the Blender game engine, and the cross-language capability of Ice™, it is possible to provide the same interfaces and functionality on the emulated device controllers, as on the real. Performance factors are described in the following:

- The frame-rate of the game engine decreases with increasing number and complexity of emulated device.
- Increasing the number of device controllers, or their communication load, may saturate the network bandwidth and latency capacities of the game engine node.
- Increasing the number of objects with enabled game physics decreases the frame-rate of the game engine.

These limitations on the game engine node capacities implies a bound to the production system complexity that will be feasible for emulation.

4.2. *Blender*

The Blender 3D application comprises modeller, animation engine, game engine. In the modeller, 3D geometric objects are drawn and annotated with information, configuration, and setup for animation, game execution, and physics. Game physics is a feature to be used in either animation or game mode.

4.2.1. Modelling

Modelling a production installation means making geometries representing visual objects, collision objects, physics objects, objects for mechanical sensors, etc., for the shop-floor scene. In addition, purely virtual objects must be modelled for the sake of emulating reality. Examples of the latter, purely virtual objects, are given here:

- For a vision system to localize certain workpiece types, a camera house object may be modelled for pure visualization. For the virtual vision system to operate an object is used to represent the frustum. Thus, the operation of the vision system is to find all object of the requested workpiece type within the frustum. Tests for visibility from the camera is done by ray-casting from the apex of the frustum to filter out occluded workpieces.
- A proximity sensor can be modelled by adding an invisible object, with an associated collision sensor, the mesh of which spans the region of sensing. If the sensing should be selective on material properties, like capacitive or inductive sensors, the objects to

sense should be marked with a material property to be distinguishable.

4.2.2. Game Engine

The game engine, once started, turns every model object of the current scene into a game object. A game object may contain any number of game sensors, game controllers, and game actuators. These are pure Blender game engine concepts, unrelated to production devices.

The game sensors of various types relate the associated game object to user input, message reception, collisions, timing events, ray tracing, and actuator events. The game actuators affect objects in the scene, like moving, creating, destroying, changing properties, etc. The simple game controllers are Boolean operators, which test their connected game-sensors and translates the resulting truth value to triggering of the connected game actuators.

For simple usage of the game engine the Boolean controllers may be adequate. However, for implementing advanced behaviour, the game controller can be chosen to be a callable object in a Python module. This type of controller will, when triggered by any sensor, execute the callable Python object in the Python interpreter embedded in the game engine. There is no limit to what a Python callable is allowed to access, in the game engine as well as with the general operating system environment.

A game controller on some object can access all state information in all of the sensors, regardless of how it was triggered. For instance, a game sensor set up for triggering its game controllers on some event in the game engine is an efficient event-based method for activating the controllers. Functionally the same behaviour may be achieved by polling of the sensor by the interested controllers. This can be more flexible, whenever the controller has multiple concerns, not all of which are related to the mentioned sensor.

4.2.3. Game Physics

Game physics gives realistic dynamic behaviour for objects under its control. It gives a generic method to emulate reality for objects which are not under explicit motion control in reality. I.e. objects that in the real world have no controller or mechanical manipulator. There are plenty of examples thereof, like passive joints and links in robots; workpieces falling from machines, grippers, or conveyors; obstacles being bumped into by AGVs; textile or other flexible material which is only partially fixed; etc.

One strategy for such free objects in the emulated production system is to explicitly control them. I.e. to implement controllers that execute the behaviour of the free objects. This requires a great deal of object-specific knowledge accessible to the controller.

Consider the example of machined workpieces dropping out of a machining station. Such workpieces may have highly complex geometries, and they may even be a one-shot batch. The stable stances of such workpieces on a flat support may be easily enumerated. The transitions from the dropping phase-space state (trajectory) to the final stance may be somewhat more complicated, even when landing on a flat surface; at this point friction and elasticity, or even plasticity, come into play. Finally, since there will typically be several workpiece in the container under the machine outlet, most of the workpieces drop on top of some structure of other workpieces, scattering and bouncing before settling in the structure. Generating offline-knowledge for the controller to use for generating realistic scenarios in real-time for such, typical situations seems impossible in the general case. If possible, the solution will be very unrealistic; the knowledge of the controller will be more a kind of simulation scenario, specific not only for the workpieces but also for the environment; or there are some very simplifying features or assumptions of the

specific workpieces and the specific container.

The game physics in Blender gives the opportunity to undertake an alternative strategy. By modelling a collision body as a compound of convex rigid bodies, alongside the visible, geometric skin of the workpiece, it is possible to achieve physically realistic dynamic behaviour of the workpieces with their environment and among them.

There are some drawbacks with using game physics.

- The Bullet physics library does not feature static friction. This may not seem severe at first, but it implies unrealistic scenarios for parts lying at rest on a support. If the support accelerates however weakly horizontally or has even the slightest inclination, the supported parts slide on the surface.
- The physics computation, even for a moderate amount of bodies, is quite heavy; and it must be kept in mind that this computational load can currently not be distributed out of the game engine process. To achieve acceptable performance for a complex production scenario with physics based dynamics, high-performance computing hardware may be necessary.

The two drawbacks of using game physics can be resolved by the same solution. The solution is based on separating the physics dynamics body from the workpiece geometry and appearance, and to be able to *freeze* the workpiece geometry to where it settles, freeing up the physics dynamics body for use by other workpiece geometries. By maintaining a pool of and reservation system for physics dynamics bodies in the game engine, only a limited number is necessary, and thus limits the load on the CPU. Simultaneously it resolves the problem with slipping workpieces, since they will freeze onto their supporting surface when below some motion threshold.

5. Device Emulator Designs

This section describes and discusses some aspects of lower level (mechanistic) design, towards implementation.

The controllers of devices in the real production system are the targets for implementation of device emulators. A device emulator interacts with the game model, i.e. its own device model, other device emulators, and other modelled elements, in much the same way as the real device interacts with its surroundings in reality.

A device is modelled geometrically with Blender, and annotated appropriately for functionality. The device control emulator is then implemented in Python and associated with the geometrical models in the game engine.

5.1. *Implicitly Controlled Devices*

The virtual production reality in the game engine contains activities for which there are no explicit controllers in the real production system. Such natural activity in reality may play an implicit role in the design of the production control. An example is that workpieces released from a gripper will drop downwards, and it will come to rest if supported by a level surface or a fixture. This mechanism is implicitly used in most pick-and-place systems in production automation.

As suggested in Section 4.2.3, and depending on the implicated complexity, the strategy of resolving the dynamic control of free objects may, or may not, involve game physics. Thus, either the controllers of the solution takes on the explicit dynamics of the emulated

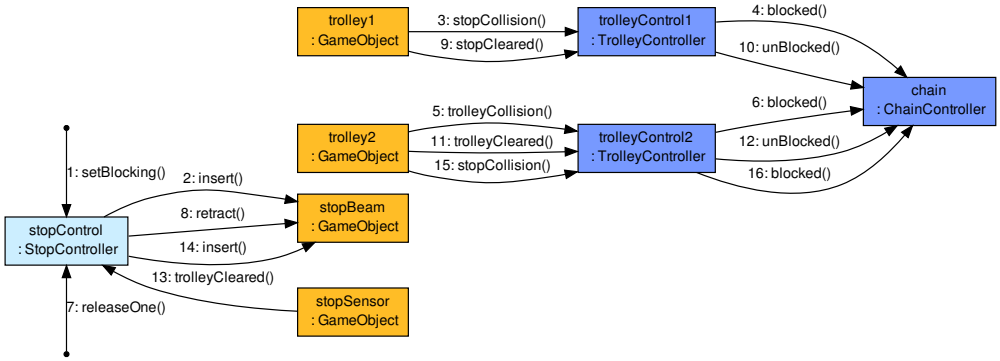


Figure 4. Communication diagram for the simple scenario of two trolleys approaching a PnF-stop. Exposed game engine controllers are shown in light blue, unexposed controllers in blue, and game engine objects in orange.

subsystem in its entirety, or the dynamics of the subsystem objects are left entirely to game physics. As it turns out, a hybrid solution is possible, where a set of controllers manages the deployment and activation-states of physics enabled devices.

In the following, two mechanisms for handling uncontrolled objects will be exemplified. One deals with trolleys on the PnF-conveyor, using direct, unexposed controllers for driving the trolleys and interacting with other trolleys and PnF-stops. The exact same mechanism is applicable on pallet conveyors. The other example is game physics control of workpiece objects.

5.1.1. PnF-Trolleys

The dynamics of the PnF-trolleys in the PnF-conveyor of the demonstrator system, the elevated red track in Figure 1, are handled by direct, unexposed control. The control of the trolleys are unexposed to the production control system, and simply realizes the physical constraints under which they move on the conveyor.

Trolley controllers move their game objects upon reception of motion updates from the conveyor chain controller. In case a trolley is blocked, it unsubscribes from such motion updates. Trolleys can be physically blocked by either bumping into the blocking-beam of a PnF-stop, or by bumping into the rear of other blocked trolleys. Either situation generates a collision event from the trolley game object, and the trolley controller unsubscribes from the motion updates. Whenever the interference from the preceding collision is cleared, the trolley controller receives another collision event from the trolley game object, and it resubscribes to motion updates.

The motion updates, tied to the frame-rate of the game engine, from the chain drive gives enough information to the trolley controller, that it can update its position along the conveyor track. The conveyor track is simply modelled as a closed polygon-mesh in Blender. In the demonstrator, the conveyor is a single circular loop, but for more advanced overhead- or pallet-conveyors, branches in the track may occur, and diversion-controllers must be implemented for the intersections on the paths.

Figure 4 is a communication diagram for controllers and game objects in a scenario involving one PnF-stop and two PnF-trolleys. The scenario assumes initially that two trolleys are approaching the PnF-stop.

(1) The PnF-stop is ordered, by external activation from the production control, into the blocking state.

(2) The stop controller inserts its stop beam into the track.
(3) As the first trolley interferes with the beam, its collision sensor triggers the trolley controller.
(4) The trolley controller unsubscribes from motion updates from the conveyor chain controller.
(5) As the second trolley gets into collision with the first trolley, its collision sensor triggers.
(6) The second trolley controller unsubscribes from motion updates from the chain controller. The state is now that both trolleys are blocked at the PnF-stop.
(7) The external production control orders the PnF-stop controller to release one trolley.
(8) The PnF-stop controller orders the retraction of the beam from the track.
(9) This collision sensor of the first trolley triggers its controller, signalling that the stop beam interference has been cleared.
(10) The trolley controller resubscribes to motion updates from the chain drive controller, and it starts moving the trolley again.
(11) As the first trolley moves out of interference with the second trolley, the controller for the second trolley gets triggered, signalling the cleared interference.
(12) The second trolley controller resubscribes for motion updates.
(13) The presence sensor at the PnF-stop now triggers as the first trolley completely passes the PnF-stop.
(14) The PnF-stop controller reacts to this by re-inserting the beam into the track.
(15) As the second trolley object reaches the beam, its collision sensor triggers the controller.
(16) The second trolley controller unsubscribes for motion updates.

Thus the final state is that the first trolley is moving along downstream of the PnF-stop, while the second trolley remains blocked at the PnF-stop.

The dynamics of the trolleys could have been designed by using constrained motion of physics objects, and would then have been even more realistic. However, the chosen design of direct control is sufficiently generic and less CPU-intensive.

5.1.2. Workpieces

The simplicity of the control logic and emulation of trolley- and pallet-conveyors, being fixed installations within one production application and varying little across applications, favours the direct control strategy. Workpiece types and environments, on the other hand, may exhibit high variation even within a single production application in flexible manufacturing. If opting for the direct control approach, as mentioned in Section 4.2.3, the designers and implementers of the emulation system faces high complexity of knowledge management regarding stable states, mechanical interactions, and dynamic control of the workpieces.

For these reasons, the game physics approach is a natural choice for workpiece management. To limit the computational load, the hybrid approach was chosen. The technique is to manage a limited pool of dynamic bodies, where attachment and control of a workpiece skin is done on request.

The following describes a scenario where a new workpiece is requested to enter the production scene. In the production demonstrator, cf. Figure 1, this occurs at the CNC-machines on the far left.

The scenario in Figure 5 involves the singleton objects for workpiece management and workpiece body pool. A workpiece body object is a compound of physics enabled objects,

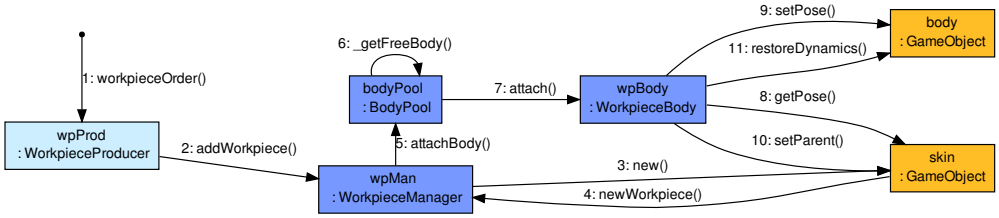


Figure 5. Communication diagram for entering a new workpiece into the emulation.

and is owned by a workpiece body controller. The body controller handles the logic of suspending and resuming dynamics for its body object, as well as placing the body at, and attaching the workpiece skin to it. For every site in the production scene where workpieces can be generated, there exist a workpiece producer. The workpiece producer initiates the whole process of creating and handling a newly created workpiece skin, as the following description illustrates.

(1) From the external production control, an order to create a new workpiece arrives at the workpiece producer.
(2) The workpiece manager is instructed to create a new skin.
(3) A new skin object comes to existence in the game engine.
(4) Any new skin starts its life cycle by informing the workpiece manager that it has come into existence.
(5) The body pool is instructed by the workpiece manager to attach a body of the correct type for handling the dynamics of the free skin.
(6) The body pool internally allocates a free body controller of the appropriate type.
(7) The body controller is handed a reference to the pertinent skin.
(8) The pose of the given skin is retrieved.
(9) The body object is set in pose to coincide with the skin.
(10) The skin is parented to the body, which effectively makes it move with the body.
(11) Finally the physic dynamics of the body object is reestablished, making it interact in a mechanically realistic manner with its surroundings.

The implemented logic that takes over after the described scenario, in the workpiece body control, is to first test whether the body is released in collision. If so, the dynamics is immediately suspended, the workpiece skin is frozen onto one of the objects it interferes with, and the body is freed at once. This behaviour is deliberately implemented for easy emulation of an attachment process. On the other hand, if the body is not initially in collision, dynamics ensures that the workpiece will start falling, tumbling, sliding, as long as the amount of motion exceeds a certain threshold. When the amount of motion goes below the threshold, the same method of freezing the workpiece skin is employed, as if there was an initial collision.

5.2. *AGV Control*

The devices internal to a minimally modelled AGV are a velocity controller and a localizer. These devices are easily emulated in the game engine. The velocity controller presents an interface with a single method for setting a 2D velocity screw, composed of two linear and one angular velocity components. At every time frame, the position and orientation of the AGV object is displaced according to the 2D velocity screw. The

localizer presents an interface of one method for retrieving the world pose of the AGV, which is read out as the pose of a central geometry of the AGV in the game engine.

These basic devices are realistically emulated in terms of interface, but too ideal in terms of precision and stability. A real velocity controller runs on data from odometers and inertial sensors, the precision and stability of which may be far from ideal. The real external positioning system uses vision-systems for recognition and tracking of LEDs on the AGVs, for which there may be abundant calibration and stability issues. Thus, it is mostly the interface and the nature of the data from the emulated devices that are realistic, rather than the detailed behaviour.

Based on these basic emulated devices, the AGV controller itself, composed of more abstract controllers like motion control, tasking control, traffic and navigation management, etc., may be implemented outside of the game engine. This is an example of crossing the game engine boundary for implementing the devices, and will help distribute the computational load on the game engine node in the system of simulation nodes. In a real application, this implies that the whole AGV controller system, except for the fundamental velocity controller and localizer, can be ported directly from the simulation to the real AGV platforms.

It is worth noting that even the velocity controller could be decomposed and moved to an external node. This requires leaving the set of motor controllers, and the odometers as the elementary devices to be emulated. Though this design may eventually alleviate further the computational load on the game engine node, the frequency of communication among velocity controller, odometers, and motor controllers may be so high that the communication load on the game engine node becomes the limiting factor for performance. This may not only be the case for the emulation system, but may also be true for the real AGVs.

5.3. *Robot Control*

The robot controllers and kinematics are based on a framework called *PyMoCo* (Python Motion Control). For a general description of PyMoCo, confer Lind *et al.* (2010), and for a demonstration in application, confer Schrimpf *et al.* (2010). Motion control in the PyMoCo framework is based on UDP-socket interaction with the native robot low-level controller.

Each robot low-level emulator in the emulation system initializes by finding the game engine objects belonging to its particular robot, gets allocated a socket port pair, and starts up listening for control input on the game engine node host. At some node in the production control, a program can be started for setting up a motion controller with PyMoCo, or any other motion control method, and provide a motion control interface over Ice™ to the production control system at large.

The implemented design for robot control in the emulation system is shown in Figure 6. At the highest level, the *Task Handler* receives macroscopic motion commands, which amounts to moving the robot tool frame linearly to a given 3D-pose. The *Task Handler* sets up the *Task Space Interpolator* for the *Joint Data Controller* to use. The *Joint Data Controller* requests task space poses from the *Task Space Interpolator* in real time, and uses the *Inverse Jacobian* to compute the corresponding joint motion to achieve the interpolated task space pose. The computed joint target is sent to the *Low-Level Controller*, which, as a game engine controller, resides inside the game engine. In case of the real low-level controller, the joint targets are adjusted and immediately distributed to the individual servos of the robot arm. In the emulated robot device, the *Forward Kinematics*

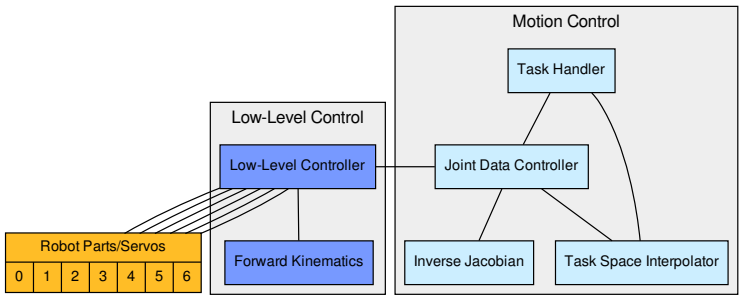


Figure 6. Object diagram for the implemented design for motion control.

is used to compute where to place the geometric objects, representing the parts of the robot, according to the given joint target.

The discussion pertaining to deployment and distribution of control elements for AGVs applies equally to the robot emulation; confer Section 5.2. It is possible, but with high computational load ensuing, to implement the motion control inside the game engine. This computational load of the game engine node is traded off for higher communication load, by having the motion control outside of the game engine. In effect, this leaves the low-level control and the associated computation of kinematics in every time frame inside the game engine. The separation at this level, i.e. low-level control inside and motion control outside of the game engine, is natural since that separation exists in real robot controllers. However, at no considerable extra communication overhead, the forward kinematics computation could be moved outside the game engine by simply sending poses of robot parts rather than joint data over the low-level controller input socket. In the current implementation this has not been considered due to the implied violation of the philosophy of having raw devices truly emulated in the game engine.

6. Performance Indication Experiments

As touched upon in Sections 4.1.2, 4.2.3, and 5.2, a number of parameters and circumstances affect the performance of an emulation setup, and thereby the entire surrounding simulation. Whether an obtained actual performance is acceptable for a given emulation and simulation setup will depend on the real-time nature of the pertinent application.

Absolute performance of a system setup executing with the Blender game engine is heavily depending on the quality and performance of many parts of the computer system setup as a whole. As is often the case in such situations, it is better to focus on the relative behaviour of certain interesting or indicating observable quantities as a function of problem size; conveying an impression of the performance *scalability* of the principles and technologies.

6.1. *Test Considerations*

With the possible exception of physics-enabled, colliding objects, it is clearly most interesting to make performance test of scalability on independent devices. The performance observables will be presented as amortized values, to clearly emphasize deviations from linear scaling. Complexity arising from interaction between controllers of emulated

production devices is not to be attributed to the emulation system, but rather to the simulated production control system.

Real-time emulation clearly distinguishes itself from discrete event simulation by not controlling real time. The factors that may be controlled inter-dependently in real-time emulation, and are critical indicators for performance, are CPU-load and frame-rate[1] The Blender Game Engine has a switch called *"Enable All Frames"*, which when set drives the emulation in a "free-running mode". When set, the frame-rate is in principle unlimited, and determined by the system resources such as performance and availability of network, CPU, GPU, RAM, bus speeds, etc. Depending on the nature of the executing emulation together with all other resource loads on the, possibly distributed, computing system, one of these will, at any given time, be the limiting factor on the achieved frame-rate. If *"Enable All Frames"* is unset, a parameter in the game engine sets a fixed upper limit of the frame-rate. Naturally, any of the aforementioned system resources may be fully loaded at a lower frame-rate; which then becomes the actual frame-rate in the emulation.

This leads to two pure performance indication principles.

- **CPU Consumption:** With *"Enable All Frames"* unset and a fixed target frame-rate, the CPU consumption for a set of tasks, of a given duration, may be considered a performance indicator of the game engine among the tasks. The CPU consumption is measured by reading the process resource statistics from the operating system before and after the task, and is taken as the sum of both system and user times consumed for the task period. Obviously, a lower CPU consumption is considered better performance.
- **Achieved Frame-Rate:** With *"Enable All Frames"* set and continuous monitoring of frame-rate and task specific state or quantity, these may be correlated to give a performance indication. A higher frame-rate indicates better performance. Since frame-rate is not an easy quantity to amortize over a problem size value, a more appropriate measure is the reciprocal of the frame-rate; the *emulation cycle time*.

Both of the CPU consumption and the achieved cycle time are easy to test for scalability by amortizing them over the problem size.

6.2. *Experiment Setups*

Three production-relevant experiment setups have been used for performance-testing the use of the Blender game engine. Two of these setups are presented in the following sections and they are used to test directly the scalability in terms of number of devices and physics-enabled objects in the game engine.

The experiment setup that is not presented in detail here was set up for running a variable number of emulated robot controllers. The robots were controlled from a remote PC, over a switched $100Mb/s$ network, in one single process running the corresponding motion controllers. The task for each motion controller was to drive its robot through a fixed number of pick-and-place cycles and with all tasks running simultaneously. Such a setup characterizes a performance test of the motion control framework set up on the motion control PC, rather than a performance test of the game engine setup. The motion control PC managed to control up to ten robots before being too loaded to respond in a reasonable real-time manner; i.e. when the motion of the robots in the game engine scene started to exhibit an observably jagged motion. Up to this maximum of robots, only a

[1]The frame-rate is the frequency with which the scene is displayed and the maximum triggering-rate of the game sensors.

very slight increase in amortized CPU consumption was observed on the motion control PC. It has not been investigated if the total CPU load on either the motion control PC or the game engine PC was the limiting factor. This will be an objective for a future experiment.

The remaining two experiment setups that were isolated to, and targeted the internals of the game engine, were both deployed on four different hardware setups with near-identical systems environments. All four systems used the Debian GNU/Linux "testing" distribution and the same versions of Blender, Python, ZeroC Ice™, etc. The configurations of the systems environments were left very close to the defaults for a freshly installed Debian GNU/Linux system. This is not by any means a guarantee that the performance test results may be used for comparing the four hardware platforms; since the installation process performs a lot of auto-detection and auto-configuration, and because drivers for different hardware peripherals may exploit or disregard a multitude of features. However, it does indicate what is to be expected out-of-the-box and it makes it quite easy to repeat and verify the results.

The four hardware platforms consisted of two desktop PCs and two laptops; all with enough RAM that no paging occurred during any experiment. The rough platform specifications are given here with a mnemonic label for each hardware system:

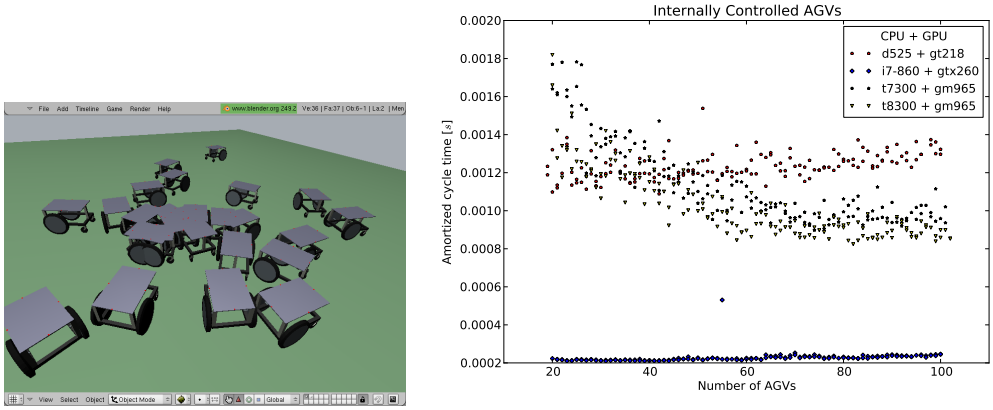
- **i7-860+gtx260** A high-end desktop with the Intel i7-860 processor and a graphics card based on the NVidia GTX260 GPU. This is the absolute top line of the four systems in any respect.
- **d525+gt218** A very low-end mini-desktop PC, with the Intel Atom D525 processor and an integrated NVidia GT218 GPU. Though the GPU is fair, the processor makes this the system with the lowest expectations,
- **t8300+gm965** A Lenovo Thinkpad X61 laptop with the Intel Core 2 Duo T8300 processor and the integrated GMA X3100 GPU in the Intel GM965 chipset.
- **t7300+gm965** The same configuration as the Thinkpad X61, but with a lower performing Intel Core 2 Duo T7300 processor. The laptop is a Lenovo 3000 V200.

All experiments presented in the following sections are based on one single run of the setup for each system, with *"Enable All Frames"* set and continuous logging of the number of devices or objects together with the actual frame-rate. Statistics is generated by steadily increasing the number of objects while ensuring that an as compact as possible coverage of the achievable domain is sampled.

6.3. *Experiment Setup: Internally Controlled AGVs*

The most central question for any production emulation scenario is how the performance scales with the number of active, production related, controlled devices in the game engine. An emulation experiment has been set up, where the number of emulated AGVs is automatically and continuously increased with regular intervals. The experiment is carried out in one single emulation run, with frequent logging of the number of AGVs and measured frame-rate. The setup is displayed in Figure 7(a).

The measurements were made in the range of 1 to 100 AGVs, and the results are displayed in Figure 7(b) as amortized cycle time against number of AGVs. In order to increase identifiability of the higher-count end of the measurements, the lower, peaking tail at the low-count end has been truncated at approximately 20 AGVs. The reason that the amortized cycle time peaks at the low-count end is that few devices each gets a considerable contribution in the amortization of the constant load of the game engine;



(a) AGV performance experiment scene.

(b) Amortized cycle time over number of AGVs.

Figure 7. Game engine setup and result for performance of internally controlled AGVs.

i.e. the part of the game engine load not associated with AGV control.

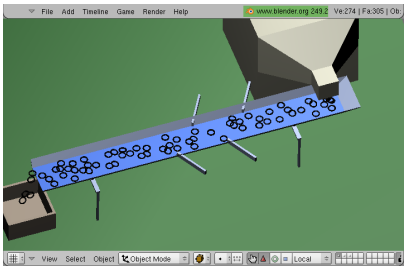
It is not surprising that the high-end desktop PC, i7-860+gtx260, is far superior to the other systems. The desktop systems, i7-860+gtx260 and d525+gt218, exhibit an expected behaviour with a slight super-linear computation time, i.e. slightly increasing amortized cycle time. It is, however, surprising that the two laptop systems, t8300+gm965 and t7300+gm965, exhibit a slight decrease in amortized cycle time, i.e. sub-linear increase in computation time.

6.4. *Experiment Setup: Physics-Enabled Objects*

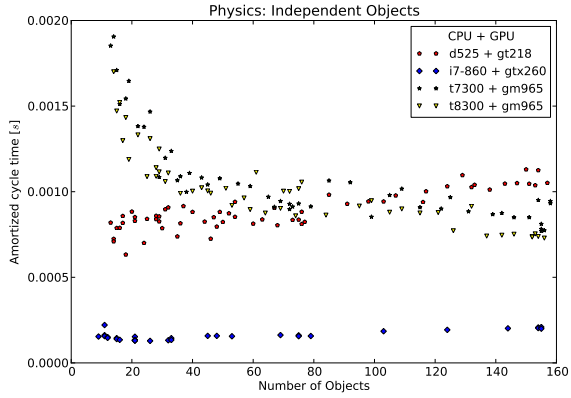
Though physics-enabled objects, categorized as "Dynamic Body", "Rigid Body", and "Soft Body" in game engine terms, are not strictly necessary for most production system emulations, it may be an important contribution to the realism of the emulation, and in many cases it directly alleviates the need for implementing an artificial virtual-physical workpiece management system. In certain production settings, or aspects thereof, the physical properties of bouncing, tumbling, slipping, and sliding are of paramount importance to the usefulness of an emulation of the production system. This section presents an experiment setup and two experiments for performance testing of physics-enabled objects. This may be characterized not as an isolated performance test of the Blender game engine, but rather as a combined test with the integrated Bullet physics library.

A device type that is found abundantly in production settings due to its low cost, flexibility, and reliability, is the *Vibrating Feeder*. It is well suited for performance testing of physics-enabled objects, because it bases its operation massively on physics properties like friction and elastic collisions, involving many interacting or separated workpieces.

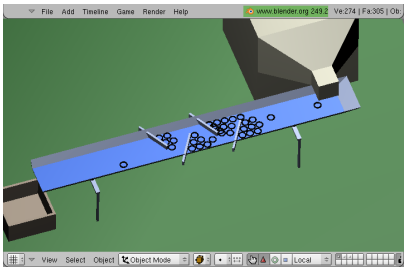
A simple model of a vibrating feeder is made up by a surface of friction bounded on the sides by low-friction inclined plates. By simply toggling the feeder plate between two narrowly displaced positions in any game engine time frame, objects will vibrate or bounce, depending on the size and direction of the displacement. Many aspects and parameters are possible to model, such as frequency, number of positions and the pattern of motion among them. Such effects and parameters will affect the motion of the supported objects, very much so in the game engine as in a real vibrating feeder. For the purpose



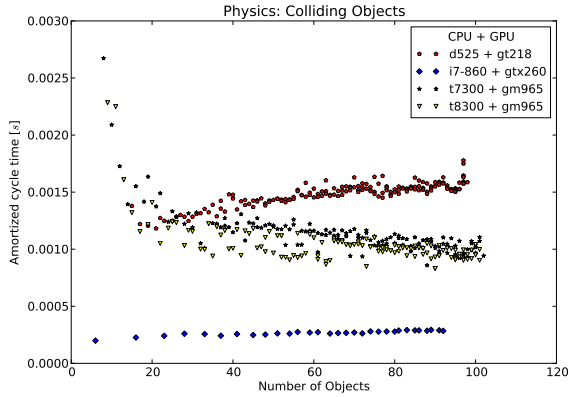
(a) Physics objects performance experiment scene for limited inter-object collisions.



(b) Amortized cycle time over number of objects for light (*rate controlled*) inter-object collisions..



(c) Physics objects performance experiment scene for heavy inter-object collisions.



(d) Amortized cycle time over number of objects for heavy (*barrier controlled*) inter-object collisions..

Figure 8. Game engine setup and results for performance of physics-enabled objects. The two cases refer to controlling the number of physics-enabled objects, O-rings on a vibrating feeder, by either putting barriers into the flow of O-rings or increasing the production rate of O-rings.

of the experiments presented here, it suffices that an adequate direction and length of displacement may be configured in the local reference system of the feeder surface.

There are two experiments performed in the vibrating feeder setup. The first experiment sets up the objects for being fed separately along the feeder surface, in principle only interacting with the feeder surface; the ***Free Flow*** experiment. This setup is shown in the screen-dump in Figure 8(a). The second experiment provokes massive inter-object collisions in addition to the interaction of each object with the feeder surface; the ***Barrier*** experiment. This is obtained with the same setup, but with a set of barriers configured to be slid into the flow of objects, creating pockets where objects are trapped. The experiment setup is seen with the barriers shifted into the object-flow in Figure 8(c). To

the far right in the scenes is seen an emulated machine, that produce workpieces[1] at a dynamically controlled rate. The O-rings drop from the machine outlet onto the vibrating surface, being transported along to the far left where they drop freely over the end of the feeder into a box where they are consumed; and destroyed as game engine objects.

The free flow experiment is carried out by a continuous increase of the steady state number of objects on the feeder, by manually adjusting the production rate of the O-ring producer. The process is allowed to run from a few objects and until the frame-rate drops to a few Hertz. The barrier experiment runs automatically with constant production rate of O-rings, and the barrier pockets fill up and trickle over one by one. As the last barrier overflows, and objects begin to fall off the feeder, the number of objects enters a steady state and the experiment is stopped.

The two experiments are highly relevant to real application scenarios of production, where transport and dynamic storage of small workpieces are frequent uses of vibrating feeders.

Results for the experiments are seen in Figures 8(b) and 8(d). The results exhibit much the same characteristics as was the case for the experiment setup with AGVs and the same comments and conclusions apply here.

7. Discussion and Conclusion

The game engine for use in real-time emulation and simulation of production systems is an example of *preadaptation*; a term borrowed from evolutionary biology. The designers and developers of the Blender game engine did not consider the domain of production control simulation. However, as there is a large overlap in geometric, logic, and real-time concerns of a 3D computer-game and production device emulation, the game engine platform turns out to nicely support this domain.

The proposed principle of a clean separation of device-emulation and production-control is considered a major contribution of this paper. The level of complexity of the demonstrator and implementation is almost complete with regard to architecture and design. There seems to be no hindrances for expanding to the complexity-level of the real production system; i.e. scenarios involving changeovers where painting system carriers, robot grippers, and workpiece types are changing during operation. Specifically, a few extra, uncomplicated devices will have to be added to the emulation system, whereas the real complexity of the full-scale scenario will appear at the production-control level.

The currently implemented production control system for testing the emulation system follows the operations described in Section 3. It does not perform any kind of scheduling and there exist only few instances of device reservation. These are central concepts for a production system that support device and machine utilization across several coincident product-setups over the same production resources. In future work, the production-control system will be targeted for full implementation, and the emulation system will be of central value for that development.

While the implemented AGV-system serves the purpose for logistics operation in the demonstration, it is much too simplified for realistic emulated operation. The value of the current implementation of the emulation system is also apparent here, used as development and test platform for implementing missing functionalities; e.g. trajectory planning and control, external localization system, and obstacle identification and avoidance.

[1]The workpiece used is an O-ring modelled by a torus with approximately 1600 polygon faces.

The implemented demonstration runs at several tens of Hertz on a modest, contemporary PC. While a severely wrongly balanced load of computation may impair the frame-rate to the level of being unusable for robot control or interference detection, this is easily avoidable. The ever increasing power of common PCs together with game engines taking advantage of new and emerging parallel processor architectures, e.g. GPUs, will enable acceptable performance of real-time emulation of entire flexible factories.

Production system engineers may take advantage of an emulation-setup, such as the demonstrator presented in this paper, as a training platform for conceptual design, implementation, commissioning, and run-in of a new production control system. Remaining issues during real commissioning may even have been foreseen during emulation-training; which is highly preferable to dealing with unknown issues during expensive downtime of a production installation.

Three experiments have been carried out and presented to address the performance of the emulation execution in the game engine; confer Section 6. Each experiment was carried out on four different computers, comprising two desktop PCs and two laptops. All experiments exhibit a reassuring, near-linear performance complexity. A consistent slightly sub-linear complexity in all experiments on the laptop platforms is the most surprising result from the experiments. This phenomenon has not been explained or investigated. There are two immediately plausible explanations, which will be investigated in the future:

- Even though the operating system and run-time environments are close to identical on all four computer platforms, the laptops are identified automatically as targets for power saving. Thus, with increased load on the system resources during an experiment, the number of active power saving features decrease, and the system performance increases.
- The GPUs of the laptop systems are inferior to those of the desktop systems, and they are the limiting factor at low problem sizes where the game engine runs with a high frame-rate. As the problem size increase, the CPUs become more loaded and they eventually become the limiting factor at the higher problem sizes.

Both explanations are consistent with poor performance at low problem sizes and with the slightly sub-linear overall behaviour. The test results lead to the overall conclusion that there are, within the domains tested, no super-linear effects that hampers the scalability of performance with the problem size in an emulation.

The suitability of the developed principles, architecture and chosen technologies, has been successfully demonstrated by implementation for a medium sized section of a production system. Hence, we conclude that our approach is applicable as a development support within the domain of distributed, autonomous production control.

Acknowledgements

The authors wishes to thank The Norwegian Research Council for funding this work through the IntelliFeed research project. Thanks also go to the industrial project partners Glen Dimplex Nordic AS and Scandinavian Business Seating AS for good input and interaction. Finally, thanks to Production Technology, SINTEF Raufoss Manufacturing AS for good support throughout the project.

References

Caie, J., 2008. *Discrete Manufacturers Driving Results with DELMIA V5 Automation Platform* [online]. : ARC Advisory Group. ARC White Paper on behalf of Dassault Systèmes [2011-04-01].

Hall, K.H., Staron, R.J., and Vrba, P., 2005. Experience with Holonic and Agent-Based Control Systems and Their Adoption by Industry. *In*: V. Marík, R.W. Brennan and M. Pěchouček, eds. *HoloMAS*, Vol. 3593 of *Lecture Notes in Computer Science* Springer, 1–10.

Henning, M., 2004. A New Approach To Object-Oriented Middleware. *IEEE Internet Computing*, 8 (1), 66–75.

Henning, M., 2006. The Rise and Fall of CORBA. *ACM Queue*, 4 (5), 28–34.

Henning, M., 2007. API Design Matters. *ACM Queue*, 5 (4), 24–36.

Kim, H., Zhou, C., and Du, H.X., 2000. Virtual Machines for Message Based, Real-Time and Interactive Simulation. *In*: *Winter Simulation Conference*, Vol. 2, Orlando, Florida, USA San Diego, California, USA: Society for Computer Simulation International, 1529–1532.

Lind, M. and Roulet-Dubonnet, O., 2010. Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control. *In*: T.K. Lien, ed. *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, Trondheim, Norway NO-7005, Trondheim, Norway: Tapir Academic Press, 67–72.

Lind, M., *et al.*, 2009. Holonic Manufacturing Paint Shop. *In*: V. Marík, T. Strasser and A. Zoitl, eds. *Holonic and Multi-Agent Systems for Manufacturing*, Vol. 5696 of *Lecture Notes in Computer Science* Springer Berlin / Heidelberg, 203–214.

Lind, M., Schrimpf, J., and Ulleberg, T., 2010. Open Real-Time Robot Controller Framework. *In*: T.K. Lien, ed. *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, Trondheim, Norway NO-7005, Trondheim, Norway: Tapir Academic Press, 13–18.

Marík, V., Vrba, P., and Fletcher, M., 2005. Agent-Based Simulation: MAST Case Study. *Emerging Solutions for Future Manufacturing Systems*, 159, 61–72.

Moon, D.H., *et al.*, 2006. A case study of the body shop design in an automotive factory using 3D simulation. *International Journal of Production Research*, 44 (18–19), 4121–4135.

Pannequin, R. and Thomas, A., 2010. Emulica: an emulation-based benchmarking framework for production control experiments. *In*: *10th IFAC Workshop on Intelligent Manufacturing Systems*, Lisbon, Portugal, Jul.. IFAC.

Park, C.M., Park, S., and Wang, G.N., 2009. Control logic verification for an automotive body assembly line using simulation. *International Journal of Production Research*, 47 (24), 6835–6853.

Schrimpf, J., *et al.*, 2010. Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control. *In*: T.K. Lien, ed. *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, Trondheim, Norway NO-7005, Trondheim, Norway: Tapir Academic Press, 19–23.

Smith, J.S., 2003. Survey on the Use of Simulation for Manufacturing System Design and Operation. *Journal of Manufacturing Systems*, 22 (2), 157–171.

Valckenaers, P. and Van Brussel, H., 2005. Holonic Manufacturing Execution Systems. *CIRP Annals - Manufacturing Technology*, 54 (1), 427–432.

Vrba, P. and Marík, V., 2005a. From Holonic Control to Virtual Enterprises: The Multi-Agent Approach. *In*: R. Zurawski, ed. *The Industrial Information Technology Hand-*

book. CRC Press.

Vrba, P. and Marík, V., 2005b. Simulation in Agent-based Manufacturing Control Systems. *In*: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, Oct.. IEEE, 1718–1723.

4.10 PyMoCo – Python-Based Robot Motion Control

Morten Lind and Johannes Schrimpf. PyMoCo – Python-Based Robot Motion Control. *Journal of Software Engineering for Robotics*, 2011. ISSN 2035-3928. In preparation

Declaration of co-authorship

The analysis, design, and implementation of the PyMoCo motion control framework is the main effort of Morten Lind. Several persons have been peripherally involved during its development. The single most contributing person among these is Johannes Schrimpf. Many of the underlying ideas and mechanisms of PyMoCo has come to expression by conversations during cooperative development in unrelated work.

By far, most of the mechanistic design, and the entire implementation is the contribution from Morten Lind. All of the live testing, i.e. controlling a real robot system, and debugging have been performed in close collaboration between Morten Lind and Johannes Schrimpf.

The entire development of the hardware and software system for intercepting the native Nachi controller, the first "accessor" which was the kickoff to start development of PyMoCo, is credited to Johannes Schrimpf.

Morten Lind is the sole writer of the paper, and designer and producer of all graphics and diagrams. Several reviews and comments during the final revisions was contributed by Johannes Schrimpf. Trygve Thomessen contributed some important comments about the Nachi controller description and supported the publishing of the paper. Valuable comments about reorganizing some of the contents were also contributed by Terje Lien to the final, submitted version. Submission and preparation of the manuscript was contributed by Morten Lind.

Journal of Software Engineering for Robotics 0(0), – 201x, 0-0
ISSN: 2035-3928

PyMoCo – Python-Based Robot Motion Control

Morten Lind[1,*] Johannes Schrimpf[2]

[1] Department of Production and Quality Engineering, Norwegian University of Science and Technology, 7491 Trondheim, Norway
[2] Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7491 Trondheim, Norway

Abstract—Design and experience with a framework for external motion control of industrial robots is presented. The framework is implemented entirely in Python. It serves as a proof of concept for performing motion control in Python, from a PC with connection to an industrial robot controller. Robotic applications requiring advanced, custom motion control, and a high level of integration with sensors and other external systems, may thus benefit from the efficiency of Python in terms of reduced development time and code complexity. The framework, dubbed PyMoCo, supplies a set of canonical motion controllers, which are comparable to those found in contemporary industrial robot controllers. Designing and implementing new motion controllers, and integrating or extending the included canonical motion controllers, is part of the architectural design of PyMoCo. Laboratory applications of sensor- and vision-based control demonstrate the usability of PyMoCo as a motion control framework and Python as a robotics application platform. For motion control applications with a control frequency not exceeding a couple of hundred Hertz, computation deadlines no shorter than some milliseconds, and latency and jitter at the order of some milliseconds, PyMoCo may be considered a feasible framework.

Index Terms—Motion control, networked robots, real-time distributed, frameworks

1 Introduction

1.1 Motivation

Advanced motion control applications for industrial robot controllers are hard and time consuming to develop and integrate with other equipment, information systems, and sensor systems. They are mostly associated with all of the following technical challenges and burdens such as complex mathematics and distributed deployment. The mathematics involved with 3D geometry and rigid body motion for describing tool motion and the kinematic Jacobian relation between joint space motion and operational space motion, are quite complex (see *Stramigioli and Bruyninckx (2001)* [1] and *Dam et al. (1998)* [2]). Robot controllers, like other embedded systems integrated in a robotic application, are to be configured, programmed, and started using its own set of technologies and systems. The mathematical complexity is inherent when operating controllable linked mechanisms, but the software system aspects of developing and integrating a set of hardware devices has the potential to be alleviated considerably compared by considering alternative deployment, control, and application platforms.

An increasing number of industrial robot controllers allow access to what may be called the *Low-Level Controller* (LLC). A low-level controller can be described as an entity providing a real-time interface for addressing the joint configuration space of the robot arm at an intermediate-level frequency; in the range from $100Hz$ to $1kHz$. Applications that utilize interfaces of such characteristics are usually implemented with compiled, intermediate-level languages, such as C or C++, and deployed on some real-time operating system (OS) platform, such as VxWorks, QNX, OS-9, RTAI+Linux. The obvious reasons for these choices are among hard requirements to real-time performance; efficiency of computation with low cycle times; and latencies on the time scale of microseconds.

In the robot application and system domain characterized by

- non-critical operations,
- moderate tolerance in the motion tasks,
- moderate tolerance to real-time performance,
- and of moderate control cycle time,

a simple and high-level, but general, programming platform may be viable to use. Experimentation with and development of advanced motion control applications on such a programming platform may be highly efficient compared to integration and development in the appli-

- *The presented work has been financially supported, and supported with equipment, by the SFI Norman programme, funded by the Research Council of Norway.*

- *Corresponding author eaddress:* `morten.lind@ntnu.no`

2

cation platform of the native robot controller. This has led to the presented attempt at enabling general motion control application development and execution in the Python Programming Language[1] in user-space of the OS; resulting in a framework called *PyMoCo*.

1.2 Technologies For Implementation

The choice of Python as the programming language and run-time platform was based on a number of factors, such as:

- All involved developers and immediately potential users of PyMoCo had experience and expertise with Python.
- The software that PyMoCo was required to be integrated with in the laboratory was mostly implemented in Python.
- Python is object oriented, and the code is highly compact and tidy.
- Several mechanisms exist for increasing performance for selected sections of code: NumPy[2], Cython[3], CXX[4].
- Python is in widespread use and has a good scientific computing community[5].
- The standard library of Python is extensive.
- Many off-the-shelf software-technologies that are generally relevant for robotics applications include, or have third party, bindings to Python. E.g. ROS[6], Blender[7], ZeroC Ice™[8], OpenCV[9].

Other programming languages and run-time platforms may be equally justified for implementing and running PyMoCo, and the choice of Python was circumstantial. I.e. there is nothing inherent in Python which makes it the obvious choice, and there is no element of the design or architecture of PyMoCo which ties it to the Python language or run-time platform. However, some of the above mentioned reasons ensured that Python was adequate.

While, e.g., the standard Linux kernel has gained in real-time performance by the PREEMPT_RT patch (see e.g. *Sally (2010)* [3]), there is no near-future outlook that anything better than soft real-time performance will be supported by the Python interpreter. This leaves PyMoCo aimed only at non-critical applications that have moderate soft real-time and computational requirements. For laboratory experimenting, testing, and prototype systems development, the computational and real-time performance of PyMoCo have shown to be adequate. This limit on applicability is traded off for an increase in application development efficiency.

In most fields of robot applications there is a tradition for requirements of extremely high reliability, stability, and safety. In fields such as robotic surgery these are all necessary factors due to the potential consequences. However, in manufacturing automation the requirement to reliability is solely attributed to the profit loss and the loss of customer confidence associated with downtime on tightly coupled, high-throughput production lines. In the not so distant future, a long-awaited change of paradigm in production automation and manufacturing control may be applied, putting more emphasis on flexibility and capability than reliability and performance of individual subsystems and production devices (*Leitão (2009)* [4]). Such a change of paradigm may render the currently closed domain of robot application- and motion-control open for non-real-time platforms like Python and control frameworks like PyMoCo, respectively.

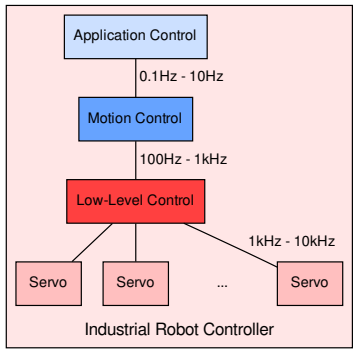
1.3 Background

In the traditional robot controller, illustrated in Figure 1(a), only the application control level is accessible for user applications, with very limited access to the motion control level. Newer robot controllers allow a wider, but still restricted, control to lower levels from external computational entities. The future may see "barebone" robot controllers, as illustrated in Figure 1(b), where only the low-level controller is provided and the application and motion control levels must be provided by the customer, an automation supplier, or sold separately by the controller manufacturer.

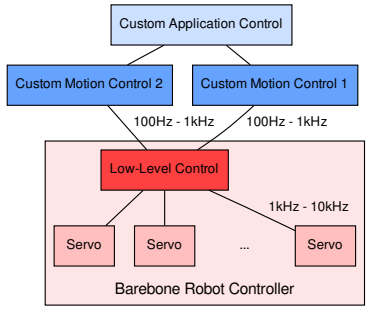
There are several robot controllers which provide access to the low-level controller. Some of the low-level controllers are tolerant in terms of real-time requirements and control frequency, and are directly usable with PyMoCo. Others are highly demanding, requiring hard real-time and control frequencies in the order of $1kHz$. Using PyMoCo with such "hard" low-level controllers require the implementation of a "decoupling"-controller, which must provide a less demanding interface for PyMoCo, while, by various strategies, meet the demands of the low-level controller. At the cost of degrading the potential performance of a class of the low-level controllers, it is thus possible to use PyMoCo universally in principle. However, as PyMoCo may not itself meet the requirements of a corresponding class of applications, it should not pretend to be considered universal in a wider sense.

Kröger and Wahl (2010) [5] give a contemporary overview of the directly available low-level accessibility

1. http://www.python.org
2. http://docs.scipy.org/doc/numpy
3. http://cython.org
4. http://cxx.sourceforge.net
5. http://www.scipy.org
6. http://ros.org
7. http://blender.org
8. http://zeroc.com
9. http://opencvlibrary.sourceforge.net



(a) A traditional industrial robot controller "stack" architecture.



(b) A "barebone" type of architecture for a robot controller.

Fig. 1: Simplified structural illustrations of the traditional industrial robot controller and the "barebone" robot controller. The barebone controller will hopefully become commercially and widely available in the future.

in some industrial robot controllers. *Dallefrate et al. (2005)* [6]) used the Mitsubishi PA-10, a slim robot arm with 7 DoF and an highly open controller. It may be controlled either in operational space at $100Hz$, or by joint velocity or torque control in $1kHz$ over ARCNET.

1.4 Scope and Limitations of PyMoCo

PyMoCo can be used in control of industrial robots, the controller of which

- offer a real-time interface to joint level motion control, and
- where the computation cycle time of

1.5 Related Work

Laboratory research and development for gaining low-level access to industrial robot controllers aims at a multitude of levels. It is fundamental field work, paving the way for increasing the value and widening the applicability of general robot motion control frameworks and platforms.

One approach is to simply use the existing facilities in the native robot controller at hand, and obtain the best possible result without intrusion. *Cederberg et al. (2002)* [7] used ABB S4C+ controller and obtained a $10Hz$ control frequency.

Obtaining sufficient insight into the workings of the proprietary controller components, by experimentation, by getting help from the manufacturer, or both, leads to another approach. This approach may be taken to either enhance the capability and performance for applications deployed on the native controller, as described by *Blomdell et al. (2005)* [8], or it may be used to open a peephole for directly addressing the native low-level controller, as mentioned in the present paper.

An extreme approach is to "cut the cables", by which the full flexibility and capability of the robot arm become accessible at the cost of an obligation to also handle the full complexity of low-level and servo control. This implies designing and implementing everything, interfacing downwards to power amplifiers and joint encoders, and upwards to the application layer. An example of such an attempt is described by *Jensen (1998)* [9].

The users of such industrial robots with accessible low-level controllers are open frameworks and platforms for robot application- and motion-control. Among several frameworks and platforms, the Orocos framework (*Bruyninckx (2001)* [10] and *Bruyninckx et al. (2003)* [11]) is a survivor in terms of active development, community involvement, availability, and universality. Orocos provides tools and facilities for communicating with and controlling robots, through motion control, and upwards for implementing real-time application control. PyMoCo may not even pretend to compete with Orocos in performance and capability; the main reason being the absence of hard real-time capability of Python.

1.6 Notation and Symbols

The symbols and notation used by the mathematical descriptions in this paper are summarized in Table 1. The notation is by no means strictly specified, but is illustrated and explained by examples. Further semantics of the symbols and notation is explained in the text near the place of usage.

The most unusual of the conventions that may be extracted from Table 1 is the way of expressing the coordinate transform, a homogeneous transform, from one reference frame to another. The notation ${}^{\mathcal{B}}\mathcal{T}$, representing the transformation of coordinates from $\mathcal{T}$ to $\mathcal{B}$ reference, is normally expressed as $T_{\mathcal{B}}^{\mathcal{T}}$. The notation used in this paper is akin to the standard way

4

$\mathbf{q}$	Abstract joint space configuration.
$\mathcal{S}$, $\mathcal{A}$, $\mathcal{E}$	Joint configuration spaces: serial, actuator and encoder.
${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}}$, ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{E}}$ ...	Joint space transforms among $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{E}$.
${}^{\mathcal{S}}\mathbf{q}$, ${}^{\mathcal{A}}\mathbf{q}$, ${}^{\mathcal{E}}\mathbf{q}$	Joint space configuration $\mathbf{q}$ expressed in $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{E}$.
${}^{\mathcal{S}}q_i$, ${}^{\mathcal{A}}q_i$, ${}^{\mathcal{E}}q_i$	Component i of ${}^{\mathcal{S}}\mathbf{q}$, ${}^{\mathcal{A}}\mathbf{q}$, and ${}^{\mathcal{E}}\mathbf{q}$.
$\mathbb{E}^3$	3D Euclidean space.
$\mathbf{v}$, $\hat{\mathbf{x}}$	$\mathbb{E}^3$ vector and unit vector.
$\mathcal{L}_i^{in}$, $\mathcal{L}_i^{out}$	Inward and outward reference frames for link i.
$\mathcal{W}$, $\mathcal{B}$, $\mathcal{T}$	Reference frames for world, base and tool.
$\mathbf{T}_{tool}$, $\mathbf{L}_i$, $\mathbf{J}_i$,	Homogeneous transforms for tool, link i, and joint i.
${}^{\mathcal{B}}\mathbf{v}$, ${}^{\mathcal{B}}\mathcal{T}$	Vector $\mathbf{v}$ expressed in $\mathcal{B}$ and homogeneous transform from $\mathcal{T}$ to $\mathcal{B}$ coordinates.

TABLE 1: Notational conventions used in this paper.

of expressing a vector, $\mathbf{v}$, in a reference frame, $\mathcal{F}$, by attaching the reference frame symbol as pre-superscript, i.e. ${}^{\mathcal{F}}\mathbf{v}$. Analogous to the expression of a vector in a reference frame, a reference frame, $\mathcal{A}$, is considered to be expressed in another reference frame, $\mathcal{B}$, by its homogeneous transform, denoted by ${}^{\mathcal{B}}\mathcal{A}$

The design diagrams in this paper are based on the Object Management Group (OMG) Unified Modeling Language (UML).

1.7 Paper Outline

Section 2 touches upon some fundamental prerequisite software systems for the PyMoCo framework. These have not been described elsewhere, but are not considered constituents of PyMoCo. Section 3 contains the substance of this paper, describing the design and implementation all around the PyMoCo framework. Section 4 shortly describes the applications in which PyMoCo has hitherto been used. Section 5 presents a discussion on the current design of PyMoCo, and directions for future work. Lastly, Section 6 summarizes on the conclusions regarding the use and feasibility of PyMoCo.

2 Prerequisites to PyMoCo

A necessary prerequisite for experimentation and testing of real motion control is a robot controller with low-level access. The original robot used with PyMoCo is the Nachi SC15F robot arm and the modifications to the Nachi AX10 controller for low-level access is briefly described.

Another natural prerequisite for motion control of articulated robots is a computational toolbox for rigid bodies in 3D Euclidean space. Thus, the Python Math3D module is shortly described.

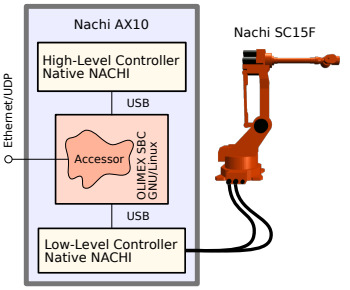


Fig. 2: Modified setup for the Nachi AX10 controller. A Single-Board Computer (SBC) hosting GNU/Linux OS is inserted to intercept the USB connection between high-level and low-level controllers.

2.1 The Nachi AX10 Controller

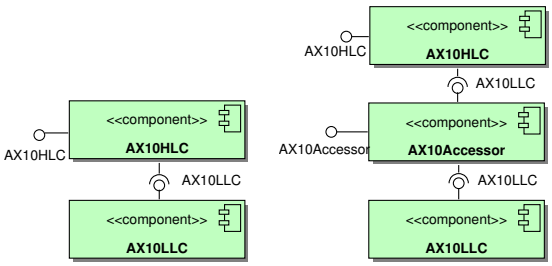
In the Nachi AX10 controller, the application and motion controllers, from the frequency hierarchy view of Figure 1(a), make up the high-level control node. The high-level controller is connected by a standard USB to the low-level controller. The exchange of commanded joint encoder positions from the motion controller and the actual values from the low-level controller happens at a configurable frequency, defaulting to $100Hz$.

The setup used for achieving external motion control of the native low-level controller is illustrated in Figure 2. The solution was based on installing a *Single-Board Computer* (SBC) which could effectively intercept the communication between the high- and low-level nodes and expose it over Ethernet/UDP. By knowing the structure of the exchanged control packages over USB, the commanded and actual joint encoder values may be read off and manipulated freely.

2.1.1 Components and Operation

A UML component representation of the native Nachi controller is given in Figure 3(a). The high-level and low-level controllers are in each their components, connected by the interface provided from the low-level controller. The ensemble of functionality and facilities available in the application environment in the high-level component, AX10HLC, is modelled as the interface *AX10HLC*. The communication with the low-level component, AX10LLC, is modelled as the interface *AX10LLC*.

With knowledge of the protocol, the communication between the native controller components was intercepted, and the component diagram in Figure 3(b) shows the modified Nachi AX10 controller. The intercepting access controller, AX10Accessor, uses the interface *AX10LLC* downwards from the native low-level controller and provides the same interface upwards to the high-level controller.



(a) The native controller, divided into the high-level and low-level controller components.

(b) The modified controller, reusing the native controller components, while intercepting their connection by an accessor component.

Fig. 3: Component diagrams for the native and modified versions of the Nachi AX10 industrial robot controller.

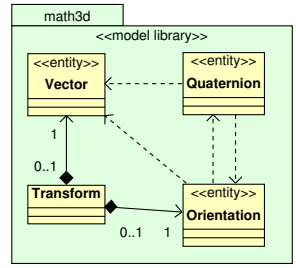


Fig. 4: Class diagram for the basic MATH3D classes for representing positions, vectors, orientations, rotations, and homogeneous transforms.

The interface *AX10Accessor* models the implemented Ethernet/UDP communication protocol, provided to an external computer. There are in practice no limitation on what the accessor component, and hence the external control application, can do to the communication between the high-level and low-level controllers, as long as it is not disrupted, compromised, or violates kinematics and dynamics limitations.

Pure low-level control is thus established in the modified controller, see Figure 3(b), by letting the AX10Accessor feed back the same commanded encoder values to the AX10HLC in the exchange over the upper *AX10LLC* interface. Simultaneously and synchronously, the AX10Accessor directly forwards the latest received encoder command values, received over the *AX10Accessor* interface, to the AX10LLC over the lower *AX10LLC* interface.

2.1.2 Characteristics and Performance

The setup and communication mechanisms described in Section 2.1.1 was fundamental to initial experimentation with trajectory generation and motion control. Using Python as a platform for trajectory and motion generation, experimental code was developed for executing of motion by addressing the AX10Accessor. It was quickly realized that the low-level controller was unrealistically tolerant to accelerations in the trajectories. In fact, there was no need for ramping up or down of velocity profiles, and no need for corner blending. This conceived the theory that the low-level controller was filtering the received input commands to build a smooth and acceptable trajectory within a moving time window. This would necessarily imply that there would be some fixed delay in the execution of the motion, compared to the stream of commanded joint positions.

A response analysis was set up for three different externally controllable robot controllers, to clarify their characteristics (Lind et al. [12]). The results showed that the modified Nachi controller has a *tracking delay*, the delay with which an actual joint position trails the commanded, of approximately $120ms$. The *response time*, the time from a commanded, large step to a joint starts affecting the actual joint value, turned out in the range of $40ms$ to $50ms$. The values found were constant over a wide range of parameters for the dynamics of the commanded motion.

These values for tracking delay and response time are quite high, and their independence from the dynamic characteristics of the trajectory support the conjecture of a trajectory filter in the low-level controller. This made sense with the observation that the tolerance to the generated trajectories was very high.

2.2 PyMath3D: SE(3) Computations in Python

Due to the lack of a general, stand-alone library for 3D Euclidean space computations, i.e. computations on $\mathrm{SE}(3)$, the "PyMath3D" library was implemented. The PyMath3D library provides the Python package MATH3D. There are various resources for the same functionality, but it is tied to the respective systems or libraries in which they are contained; e.g. OpenGL, Orocos, and Blender.

PyMath3D is quite minimal, representing the cardinal objects for positions, vectors, orientations, rotations, and homogeneous transforms. For different purposes, orientations are represented by both matrices, in the Orientation class, and quaternions, in the Quaternion class. Positions and vectors are represented by the same class, Vector. To distinguish positions from vectors, a Vector object has a protected member, isPosition, for use when deriving or transforming a Vector object.

The relations among the fundamental classes of MATH3D are seen in Figure 4.

Basic interpolation on $\mathrm{SE}(3)$ and its subspaces are implemented in the sub-package INTERPOLATION. Interpolation classes for $\mathbb{E}^3$, $\mathrm{SO}(3)$, and $\mathrm{SE}(3)$ are implemented in

6

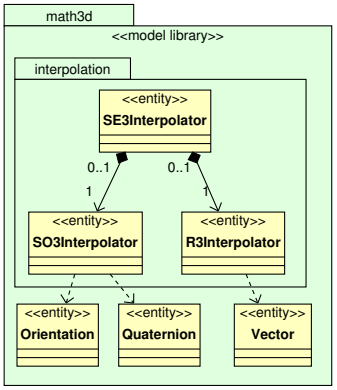


Fig. 5: The MATH3D.INTERPOLATION package elements, and their relation to elementary classes from MATH3D.

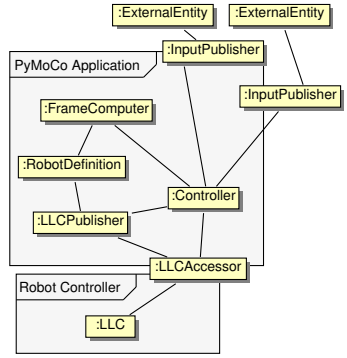


Fig. 6: UML object diagram for illustrating the basic architecture of a PyMoCo application.

R3Interpolator, SO3Interpolator, and SE3Interpolator, respectively. R3Interpolator implements the natural linear interpolation on $\mathbb{E}^3$, SE3Interpolator implements the spherical linear interpolation (*Slerp*) on $\mathrm{SO}(3)$, while SE3Interpolator is the direct combination of the two subspace interpolators. The class relations for the INTERPOLATION package is seen in Figure 5.

A good guide to 3D Euclidean mathematics is found in Stramigioli and Bryninckx [1] and a good guide for understanding the spherical linear interpolation is found in Dam et al. [2].

3 Design and Implementation

This section presents the design and implementation of PyMoCo as it has evolved to its current state. The description is organized along the three packages providing the bulk of the entity functionality of PyMoCo.

3.1 Application Architecture Overview

The basic architecture of a PyMoCo application is illustrated by the UML object diagram in Figure 6. It is centred around a specific Controller instance, which may be any of the canonical controllers in Section 3.5, any further specialization of these provided by an application developer, or any free implementation specializing the base class Controller. The controller is connected to a low-level publisher, a LLCPublisher instance. For performing kinematics related computations, the controller is associated with a FrameComputer instance. The low-level publisher is responsible for publishing the joint status, such as encoder readings, to any subscriber in the PyMoCo application. The controller, frame computer, and low-level publisher instances are associated with a specialization of a RobotDefinition instance. The robot definition provides fundamental information about kinematics of the specific robot, facilities for transforming among joint spaces and encoder spaces, and communicating with the robot controller accessor.

The LLCAccessor instance is a specialized entity for the real-time interaction with the pertinent robot used in the PyMoCo application. The LLCAccessor class for a native robot controller may reside in the PyMoCo process, in another process on the same computer, on a remote computer, or on the computer in the native robot controller. It acts as a driver for a specific robot in PyMoCo, providing the high-frequency, real-time mechanism for exchanging joint status and control commands between the PyMoCo control application and the native robot controller.

The input publishers, represented by their base class InputPublisher, are similar in principle and operation to the low-level controller publisher (the LLCPublisher) but their specializations are for the purpose of publishing measurement and information from all external systems available in the application setup.

3.2 Package Overview

PyMoCo consists of the top level PYMOCO package, with four subpackages. An overview of the individual packages are given in the following, and the package diagram in Figure 7 summarizes the inter-package dependencies.

INPUT
: The input mechanisms for external entities, such as sensors and information services, are implemented in this package.

ROBOT
: Utilities and definitions relating to specific robots. The use of a specific robot requires defining and implementing it here.

KINEMATICS
: General kinematics facilities. Classes for joints

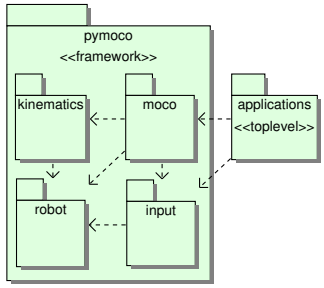


Fig. 7: Package overview of PyMoCo and the main relations for applications using it.

and kinematics computation are implemented here.

MOCO
: Base classes for motion controllers and the canonical motion controllers. Currently there are six canonical motion controllers.

The INPUT package is a container for standard input mechanisms from external equipment. Notably it contains the LLCPublisher class for distributing status information from the connected low-level controller. An important role played by a LLCPublisher object is to let motion controllers synchronize their motion computation with the low-level controller. A ForcePublisher class is implemented in the INPUT package, for supporting the general input from a force-torque sensor. Both input classes are based on the publisher-subscriber pattern.

In the following sections, the remaining packages and their underlying analysis will be described.

3.3 Robot Utilities: The ROBOT Package

The ROBOT package contains the robot-specific functionalities, information, and definitions. A robot controller that is to be used with PyMoCo must have definitions and functional implementations set up to support interfaces in the ROBOT package. A standard way of making a robot available is to implement a specialization of the ROBOT.RobotDefinition; thus encapsulating the functionality for the specific robot, and exposing a compliant interface.

3.3.1 Joint Configuration Spaces

A basic assumption in PyMoCo kinematics is that, dealing with standard industrial robot arms, all defined robots may be represented by a serial kinematics. The space of serial kinematics joint configurations, $\mathcal{S}$, is thus generally available. The serial space operates with the direct angular and linear positions of the (abstract) joints in the serial backbone kinematics structure for the robot.

For reasons of stiffness and moment of inertia, many intermediate to large industrial robots are not entirely serial in structure, having a parallel linkage on the upper arm, connecting shoulder and elbow. Some of the larger robots also have a lower-arm parallel linkage, connecting elbow and wrist. Also abundant are robots with retracted servo motors located at the elbow, which control the three wrist joints by axles through the lower arm. The retraction of these motors does nothing for stiffness, but lowers the moment of inertia. Both of these mechanisms affects the real, *actuator-related* kinematics of the robot. The actuator configurations space, $\mathcal{A}$, is the domain of actuator positions in the true mechanical structure.

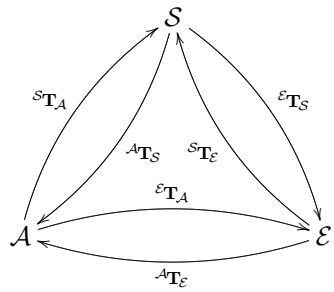


Fig. 8: Joint spaces for serial ($\mathcal{S}$), actuator ($\mathcal{A}$), encoder ($\mathcal{E}$) representations and translation operators.

The encoder configuration space, $\mathcal{E}$, is closely related to the actuator space, representing the encoder values for the actuator positions. The raw low-level controller may not communicate in the domain of angular and linear positions, but in encoder positions. There will typically be some gain-offset relation between the actuator angles and the actuator input encoders.

The motion control and kinematics computations will work naturally with $\mathcal{S}$, while underneath the low-level controller will typically require and provide robot arm configurations only in $\mathcal{E}$. Thus, at least a bijective mapping between $\mathcal{S}$ and $\mathcal{E}$ must be defined. The actuator space, though not necessarily used directly, is simply an intermediate space, true to the mechanical structure of the controlled robot. In certain situations, however, for more complicated linkages than the parallelogram or for reasons relating to kinematic limits, it may be necessary for specific motion controllers to use the actuator space.

Mappings relating the three joint spaces are illustrated in the graph in Figure 8. It is a requirement to the implementation of the RobotDefinition class for a specific robot to implement the six mappings to translate among the three joint spaces.

The serial configuration, ${}^{\mathcal{S}}\mathbf{q}$, of the robot mechanism is necessary for performing kinematic computations in the controller. The transform ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{E}}$ may be used com-

putationally in the PyMoCo-controller, when receiving an encoder configuration from the low-level controller, ${}^{\mathcal{E}}\mathbf{q}$, to translate into a serial representation, ${}^{\mathcal{S}}\mathbf{q}$, by the operation

$$ {}^{\mathcal{S}}\mathbf{q} = {}^{\mathcal{S}}\mathbf{T}_{\mathcal{E}}{}^{\mathcal{E}}\mathbf{q} \tag{1} $$

Similarly, when a PyMoCo-controller has a new computed serial configuration to submit as the commanded encoder configuration to the low-level controller, the inverse relation is applied as

$$ {}^{\mathcal{E}}\mathbf{q} = {}^{\mathcal{E}}\mathbf{T}_{\mathcal{S}}{}^{\mathcal{S}}\mathbf{q} \tag{2} $$

The transforms ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}}$ and ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{S}}$, are typically linear operators of rank equal to the DoF of the robot arm. However, non-linear operators may be the case for a specific model of a robot, as long as they are bijective and continuous or, preferably, diffeomorphisms between $\mathcal{E}$ and $\mathcal{S}$. The fundamental assumption of the existence of a serial equivalent to the real mechanical, actuated structure must be kept in mind, since this is a matter of definition of the existence of the bijective transforms. The linear transforms may be represented by non-singular $N \times N$ matrices, with N being the number of DoF of the robot arm.

Couplings other than the direct correspondences between $\mathcal{A}$ and $\mathcal{S}$ will be reflected as off-diagonal elements in the matrix representations of ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}}$ and ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{S}}$.

The operator forms of ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{E}}$ and ${}^{\mathcal{E}}\mathbf{T}_{\mathcal{A}}$ will typically be a simple gain-offset (i.e. affine) relation. Robots using simple linear encoders attached to the input side of the actuator gearboxes fulfil such a relationship. It may be characterized by real vectors $\mathbf{g}, \mathbf{o} \in \mathbb{R}^N$ in an affine transform

$$ {}^{\mathcal{E}}\mathbf{q} = \mathrm{diag}(\mathbf{g}){}^{\mathcal{A}}\mathbf{q} + \mathbf{o} \tag{3} $$

The values of $\mathbf{g}$ and $\mathbf{o}$ for a specific robot depends on the characteristics of the encoders and on the calibration. They may either be read out from the native robot controller or obtained by external observation and modelling of the robot.

The transforms between $\mathcal{E}$ and $\mathcal{S}$, ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{E}}$ and ${}^{\mathcal{E}}\mathbf{T}_{\mathcal{S}}$, are the most useful. The exemplification surrounding Equations 1 and 2 indicates that, for most cases ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{E}}$ and ${}^{\mathcal{E}}\mathbf{T}_{\mathcal{S}}$ are the only ones necessary. They may be implemented by simply copying and sequencing the implementations for the transforms ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}}$ and ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{E}}$, and ${}^{\mathcal{E}}\mathbf{T}_{\mathcal{A}}$ and ${}^{\mathcal{A}}\mathbf{T}_{\mathcal{S}}$, respectively. However, depending on the structure of the mechanism, there might be further optimizations than just reducing the call overhead. This possibility for optimization has not been investigated, and for simple, serial mechanisms it is not expected that any such optimizations exist.

3.3.2 Robot Link and Joint Transforms

Section 3.4 explains how the kinematic system in PyMoCo is designed and which conventions are used. The kinematic structures and computations belong in the FrameComputer from the KINEMATICS package. However, it is the responsibility of the specific RobotDefinition classes to be able to create the correct joint and link objects for the kinematics; since the specific robot information is located with the ROBOT package.

Example: Nachi SC15F

As an example of the joint spaces transform for practical purposes the Nachi SC15F robot is considered here. It is the only robot yet to have been in real use with PyMoCo.

The coupling transform from actuator space to serial space, ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}}$, may be read off the native controller, and has the following shape

$$ {}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & {}^{3}\epsilon_4 & 1 & 0 \\ 0 & 0 & 0 & {}^{3}\epsilon_5 & 0 & 1 \end{bmatrix} \tag{4} $$

The parallel linkage is easily identified by the off-diagonal element ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}[2,1]} = -1$; enumeration starting at 0. This transforms such that

$$ {}^{\mathcal{S}}q_2 = {}^{\mathcal{A}}q_2 - {}^{\mathcal{A}}q_1 \tag{5} $$

The fact that ${}^{\mathcal{S}}\mathbf{T}_{\mathcal{A}[2,1]} = -1$ stems from the balance of the parallel linkage, forming a parallelogram. Had this parallel linkage not been a parallelogram, but of a more general trapezoid shape, the right hand side of Equation 5 would be a non-linear function of ${}^{\mathcal{A}}q_1$ and ${}^{\mathcal{A}}q_2$.

The other coupling mechanism, by which joint 3 has a slight impact on joints 4 and 5, due to rotation of the lower arm around the axles for the latter joints, is manifested by small off-diagonal elements ${}^{3}\epsilon_4$ and ${}^{3}\epsilon_5$, respectively. The equations for serial joints 4 and 5 thus become

$$ {}^{\mathcal{S}}q_4 = {}^{\mathcal{A}}q_4 + {}^{\mathcal{A}}q_3{}^{3}\epsilon_4 \tag{6} $$

$$ {}^{\mathcal{S}}q_5 = {}^{\mathcal{A}}q_5 + {}^{\mathcal{A}}q_3{}^{3}\epsilon_5 \tag{7} $$

The transform from actuator space to encoder space, ${}^{\mathcal{E}}\mathbf{T}_{\mathcal{A}}$, simply follow the affine transformation in Equation 3, where $\mathbf{g}$ and $\mathbf{o}$ may be directly read off the native controller according to the actual calibration.

3.4 Kinematics Utilities: The KINEMATICS Package

Though the underlying creation of objects from the kinematic modelling in PyMoCo resides with the specialized RobotDefinition classes in the ROBOT package, it is natural to describe the model and conventions used

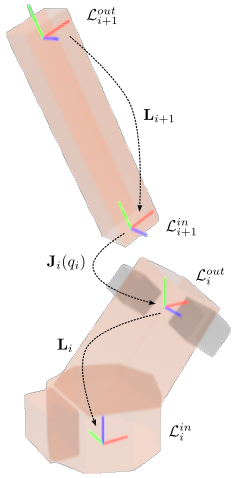


Fig. 9: The layout of link frames, and link and joint transforms for links 1 (labelled i) and 2 (labelled $i+1$) of the Nachi SC15F. For clarity, link 2 is shown slightly rotated from its home position, and offset upwards from its attachment to link 1. Link frames are shown by red, green, and blue sticks, symbolizing $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$, and $\hat{\mathbf{z}}$ directions, respectively.

for the kinematic implementation with the KINEMATICS package. The computations using the implemented and configured kinematics for a specific robot resides in the specializations of the Joint classes and the FrameComputer class, all residing in the KINEMATICS package.

Figure 9 shows two consecutive links, rendered transparent and separated for clarity, from a crude 3D model of the Nachi SC15F. The links are annotated with link frames, and joint and link transforms, which follow a convention to be described in the following.

The kinematic model underlying the current implementation, and the conventions used in the specialized robot definitions and kinematics are not computationally optimal; when compared to other representation schemes like Denavit-Hartenberg or Hayati-Roberts. The emphasis has been on flexibility and ease of modelling, description, and implementation. Though this may, in some cases, hamper even the soft real-time requirements of applications based on PyMoCo, it has not been the case so far.

For modelling and maintaining code and configuration, a clean separation between joint and link transforms is advantageous. The clean separation between joint and link transforms is expressed even in the implementation. In PyMoCo, efficiency is compromised for the cause of flexibility and pure mapping between conceptual modelling and implementation. The resulting model and implementation adheres to the conceptual description by Waldron and Schmiedeler (2008) [13], and all lower-pair joint types can be easily and directly implemented.

3.4.1 Kinematic Model and Formalism

The kinematics model is explained in terms of link frames, $\mathcal{L}$, joint transforms, $\mathbf{J}$, and link transforms, $\mathbf{L}$. Link frames are coordinate frames fixed in some link. Link transforms are homogeneous transforms between internal frames in a given link. Joint transforms are homogeneous transforms of one or more parameters, which are joint space coordinates. A joint transform defines the relation between link frames of two consecutive links.

The number of joints i symbolized by N, and joints transforms are described by an ordered set $\{\mathbf{J}_i\}_{i=0}^{N-1}$ and in a serial structure there are a set of $N+1$ links, with an ordered set of link transforms $\{\mathbf{L}_i\}_{i=0}^{N}$.

For link number i there will be two internal link frames for a serial kinematic structure. These are symbolized by $\mathcal{L}_i^{in}$ and $\mathcal{L}_i^{out}$. The superscripts in and out refers to whether the link frame is the *inward* or *outward* frame with respect to the kinematic chain, starting at the base. The base and tool of the kinematic frame are symbolized by $\mathcal{B}$ and $\mathcal{T}$, respectively. The base frame is taken to be equal to the inward frame of link 0, $\mathcal{B} = \mathcal{L}_0^{in}$, and the tool frame will be attached to the last outward frame by a tool transform, $\mathbf{T}_{tool}$, such that $\mathbf{T}_{tool} = {}^{\mathcal{L}_N^{out}}\mathcal{T}$. The total set of frames for the kinematics of a robot is described by an ordered set $\left\{\mathcal{L}_i^{in}, \mathcal{L}_i^{out}\right\}_{i=0}^{N} \cup \{\mathcal{T}\}$.

The link transform for link i, $\mathbf{L}_i$, is static and transforms $\mathcal{L}_i^{out}$ coordinates to $\mathcal{L}_i^{in}$ coordinates. It is defined by the relation between the link frames of the pertinent link as follows

$$\mathbf{L}_i = {}^{\mathcal{L}_i^{in}}\mathcal{L}_i^{out} \tag{8}$$

The definition of a joint transforms follow a rule similar to the definition of the link transforms. However, joint transforms are, not surprisingly, dynamic. Currently in PyMoCo, each joint is functionally dependent of a single parameter. The i'th joint transform is dependent of the i'th joint value, ${}^{\mathcal{S}}q_i$, from the serial kinematics space; i.e. $\mathbf{J}_i = \mathbf{J}_i({}^{\mathcal{S}}q_i)$. The joint transforms defines the kinematic chain connection between an immediately succeeding inward link frame to an immediately preceding outward link frame as follows

$${}^{\mathcal{L}_i^{out}}\mathcal{L}_{i+1}^{in} = \mathbf{J}_i({}^{\mathcal{S}}q_i) \tag{9}$$

A joint transform need not necessarily depend on only one joint configuration value. However, as it will become clear in Section 3.4.2, the current implementation depends on the ordering and numbering of the joint and link transforms, for performing the combined computation of the kinematic chain.

10

The precise definition of *forward kinematics*, or *direct kinematics*, vary a among the authors in the literature of the field of robotics. PyMoCo uses a variant, which may be dubbed *frame forward kinematics*, which honours a principle of separation of kinematic structure and geometric link models. In the frame forward kinematics, link frames and tool frame, i.e. the set $\left\{\mathcal{L}_i^{in}, \mathcal{L}_i^{out}\right\}_{i=0}^{N} \cup \{\mathcal{T}\}$, are computed as functions of the serial joint space configuration. This may be formulated as

$$\left\{\mathcal{L}_i^{in}({}^{\mathcal{S}}\mathbf{q}), \mathcal{L}_i^{out}({}^{\mathcal{S}}\mathbf{q})\right\}_{i=0}^{N} \cup \left\{\mathcal{T}({}^{\mathcal{S}}\mathbf{q})\right\} \tag{10}$$

The actual representation of the forward kinematics in Equation 10 must be expressed in some coordinate frame, which may conveniently be taken to be the robot base, $\mathcal{B}$, or the world frame, $\mathcal{W}$. The actual representation as computed by the forward kinematics may thus be expressed more precisely as

$$\left\{{}^{\mathcal{B}}\mathcal{L}_i^{in}({}^{\mathcal{S}}\mathbf{q}), {}^{\mathcal{B}}\mathcal{L}_i^{out}({}^{\mathcal{S}}\mathbf{q})\right\}_{i=0}^{N} \cup \left\{{}^{\mathcal{B}}\mathcal{T}({}^{\mathcal{S}}\mathbf{q})\right\} \tag{11}$$

The base frame has been chosen as reference for convenience, avoiding the extra transform to world coordinates, ${}^{\mathcal{W}}\mathcal{B}$, which can be considered irrelevant for the kinematics of the robot.

3.4.2 Computational Implementation

Based on the model and formalism presented in Section 3.4.1, this section presents design-aspects of kinematic computation and its corresponding implementation.

From the basic formulation and definition the internal link frames and the tool frame of the robot may be computed in base coordinates according to the following recursive formulae; with obvious limits on n

$$
\begin{aligned}
{}^{\mathcal{B}}\mathcal{L}_0^{in} &= \mathbf{I} && (12)\\
{}^{\mathcal{B}}\mathcal{L}_n^{out} &= {}^{\mathcal{B}}\mathcal{L}_n^{in}\mathbf{L}_n && (13)\\
{}^{\mathcal{B}}\mathcal{L}_n^{in} &= {}^{\mathcal{B}}\mathcal{L}_{n-1}^{out}\mathbf{J}_{n-1}({}^{\mathcal{S}}q_{n-1}) && (14)\\
{}^{\mathcal{B}}\mathcal{T} &= {}^{\mathcal{B}}\mathcal{L}_N^{out}\mathbf{T}_{tool} && (15)
\end{aligned}
$$

The recursions in Equations 12 to 15 may be trivially unravelled to the following explicit forms

$$
\begin{aligned}
{}^{\mathcal{B}}\mathcal{L}_0^{in} &= \mathbf{I} && (16)\\
{}^{\mathcal{B}}\mathcal{L}_n^{out} &= \left(\prod_{i=0}^{n-1}\mathbf{L}_i\mathbf{J}_i({}^{\mathcal{S}}q_i)\right)\mathbf{L}_n && (17)\\
{}^{\mathcal{B}}\mathcal{L}_n^{in} &= \prod_{i=0}^{n}\mathbf{L}_i\mathbf{J}_i({}^{\mathcal{S}}q_i) && (18)\\
{}^{\mathcal{B}}\mathcal{T} &= \left(\prod_{i=0}^{N-1}\mathbf{L}_i\mathbf{J}_i({}^{\mathcal{S}}q_i)\right)\mathbf{L}_N\mathbf{T}_{tool} && (19)
\end{aligned}
$$

For reasons of clarity, the dependence of joint space configurations have been omitted from Equations 12 to 19, except for the single components of the joint transforms. The joint transform dependencies are the elementary dependencies on the joint space configuration. From the indices in the products of the explicit forms in Equations 16 to 19, the dependencies of the link and tool frame transforms to base reference, on which joint space configuration components they depend.

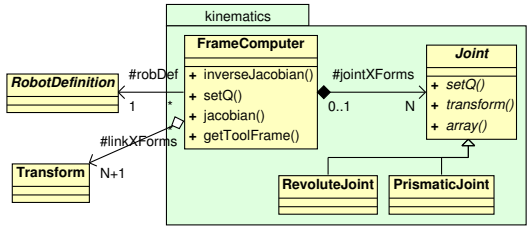


Fig. 10: Class diagram of the central classes in the KINEMATICS package.

Figure 10 illustrates the KINEMATICS package classes, and some of their most important relations. As is clear from the figure, the central class of the KINEMATICS package is the FrameComputer class.

The Joint class implements an abstract joint, with concrete joint classes inheriting from the Joint class. Implemented with PyMoCo are specialized joint classes for the most common two types of joints in industrial robotics, RevoluteJoint and PrismaticJoint. Most industrial robots use only revolute joints, but in applications they are often placed on a set of external, prismatic servo axes for reach, transport, or relocation.

The abstract Joint class imposes the implementation of a method, setQ, for setting the joint variable, and the methods transform() and array() for retrieving the computed joint transform. The operations of retrieving the joint transform may be given an optional joint value, implying a call to setQ(). A call to setQ() caches the computed joint transform, and subsequent calls to retrieval methods, without setting a joint value, or setting the same joint value, results in no new computation.

The methods transform() and array() return the same homogeneous transform, but for convenience in different form. transform() returns an object of class MATH3D.Transform, whereas array() returns a more efficient $(4, 4)$ NumPy array,

The FrameComputer class implements the kinematics relations between joint configuration space and the space of link and tool frames for the robot, with which it has been set up. The implementation is a realization of the formulae in Equations 16 to 19. An instance of a FrameComputer needs a specific RobotDefinition at creation time; stored as member robDef. From robDef the FrameComputer object acquires reference to a set of link transforms,

linkXForms, and a dedicated copy of a set of joint transform objects, jointXForms. The link transforms are constants, and hence several FrameComputer objects may share the same set of link frames. The joint transform objects are parametrized, state-full object, of the base class Joint, and hence, each FrameComputer object must have a dedicated copy.

A FrameComputer object caches its state induced by the caching joint transformations, and its computed values from the last joint space configuration given. Thus subsequent calls require little computation, provided they do not set a new joint space configuration.

The notable, operational methods of the FrameComputer class are

setQ
: Sets a given serial space configuration, and performs the full computation of the forward kinematics.

getToolFrame
: Retrieves the tool frame according to the computed forward kinematics.

jacobian
: Computes and returns the Jacobian, according to the computed forward kinematics.

inverseJacobian
: Computes and returns the inverse of the Jacobian, acquired by implicitly calling jacobian.

In addition to the listed methods of the FrameComputer class, a range of more specific methods exists for retrieving any computed link frame, and for dynamically configuring the tool transform. All retrieval methods take an optional new serial space configuration, by which an implicit computation of the forward kinematics is performed accordingly.

At the interface of the FrameComputer class, transforms from the MATH3D package are used per default. However, for efficiency, all computation and caching is internally performed on corresponding $(4, 4)$ arrays from the NumPy package. For all retrieval methods returning link frame poses, represented by transforms to world or base coordinates, these are also per default given by transforms from the MATH3D package. By an optional parameter to these retrieval methods, the returned object will be a $(4, 4)$ NumPy array. This may reduce conversion overhead for surrounding client code for the FrameComputer object that also operate with NumPy arrays.

3.5 Motion Controllers: The MOCO Package

The environment and facilities for implementing motion controllers are the essential features of PyMoCo; in contrast to supplying implemented motion controllers. When regarded apart from the facilities and the environment, basic motion controllers can be extremely easy to implement. Naturally, any number of specific control issues may arise from, for instance, complex, incomplete, or noisy sensor inputs. Other issues may arise from an application in need of a complex system of motion controller covering different aspects, to behave well, cooperate, and be coordinated.

The current design of PyMoCo distinguishes one specific motion controller in any given run-time situation, as the one that has the communication to the robot controller, through the low-level access to the operated robot; see Section 2.1. This may appear to be an artificially tough constraint, but it addresses the requirement that the low-level accessor must be fed control commands at a regular rate. Failing to meet the deadline for sending a control command, may, depending on the low-level controller or low-level accessor logic, imply shutting down the robot servo system. Thus, in PyMoCo, there must always be a dedicated motion controller, responsible for the obligations to the low-level controller. A more flexible mechanism for coordination and cooperation among several motion controllers, to share the responsibility of meeting the requirements from the low-level controller, or accessor, is possible to implement with PyMoCo, but none are included, at its current state.

The current design has two flaws, related to the lack of the aforementioned mechanism for sharing responsibility, which must be addressed in future analysis and design:

- No mechanism is supplied or devised for switching motion controller in case the low-level accessor is intolerant towards a period of absent control commands.
- No mechanism enforces access control, to ensure that only one motion controller is actually active at any time.

Neither of these flaws present any conceptual problems, but requires an elaborate addition to the design. For the current use, with the Nachi AX10 controller, modified with the SBC hosting the AX10Accessor, there is no real problem with switching controller, since it is tolerant to any length of disruption in the control command stream.

3.5.1 Controller Base Classes

The important relation of the motion controller base classes, Controller and KinematicsController, are shown in the class diagram in Figure 11.

The most general controller base is the Controller class. It subscribes to an associated object of the LLCPublisher class, for receiving status updates from the low-level controller, by invocation of its llcNotify() method. For specialized motion controllers, it provides protected methods for setting joint space configurations or adding an increment to the actual joint configuration, as reported by the associated

12

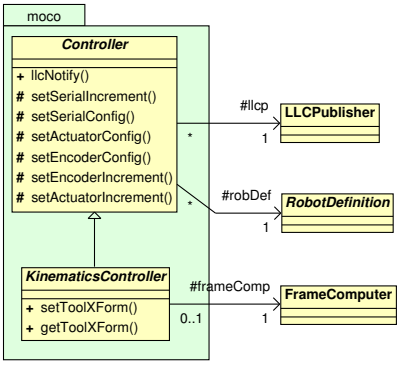


Fig. 11: Base classes for controllers, and their main relations.

LLCPublisher instance. The associated RobotDefinition instance provides kinematic information and conversion operations for the specific connected robot. The methods to set or increment the commanded joint space configuration uses joint speed limits from the associated RobotDefinition instance for scaling a commanded joint step into one tolerated by the operated robot, if necessary.

The KinematicsController class provides some further facilities for specialized motion controllers that relate to operational space; i.e. rigid body manipulation of the tool frame, $\mathcal{T}$. It gives access to a FrameComputer instance for inherited controllers to use, and specific methods, setToolXForm and getToolXForm, for accessing the applied tool transform, $\mathbf{T}_{tool}$. The tool transform is important for the application layer to manipulate through the motion control layer, since it is the application that has the knowledge of tool setups, tool changes, and tool transform calibration and corrections.

3.5.2 Specific, Canonical Controllers

As previously stated, the specific, canonical motion controllers are not a necessity for PyMoCo as a framework. But insofar as a selection of highly applicable and versatile motion controllers may be identified that are not specific to any application in particular, they may be hosted and distributed with PyMoCo; if for nothing else, just to serve as examples. The categorization "canonical motion controller" should be considered similar in nature to the set of general motion controllers offered by the application platform of a standard industrial robot controller.

Figure 12 illustrates the inheritance hierarchy, and prominent functional interface methods, of all the controller classes. For each controller there is a variety of administrative methods for synchronization, aborting, etc., which are not shown in the interfaces in Figure 12.

There are currently six canonical controllers implemented with PyMoCo which fall pair-wise into three categories.

- Two are standard, non-interactive, linear controllers in joint and operational space. Such linear controllers are expected to be found in any standard industrial robot controller.
- Two are specializations of the operational space linear controller, but open for real-time corrections of tool or path transforms. Such real-time interactiveness is typically provided to, or available for purchase by, users of robot controllers with advanced processing applications.
- The last two are real-time interactive velocity controllers in joint and operational spaces. They are useful in sensor-based control applications, but rare to come by in industrial robot controllers due to their demand for real-time interactivity.

The design of these controllers in PyMoCo will be explained in the following sections.

3.5.3 Linear Motion Controllers

JointLinearController and ToolLinearController represent what are probably the two most common motion controllers found in traditional industrial robot controllers. They lend themselves well towards playing back offline-generated paths from simple applications.

The functional methods are JointLinearController.setJointTarget() and ToolLinearController.setTargetPose(). The functional methods takes a specification of a target to be obtained by a linear motion in the addressed spaces. Both controllers take optional parameters, with reasonable default values, for specifying maximum accelerations for ramping the speed up and down at the start and end of the path. Another optional parameter gives the target speed to use on the main path. The controllers are stable towards unattainable combinations of acceleration limits, path lengths, and target speeds; preference is given to respect acceleration limit and total path length.

JointLinearController addresses the serial space, $\mathcal{S}$. The given target speed and maximum acceleration is used for limiting the motion on the joint that has the longest travel to the given target. These limits are applied not in the serial, but in the actuator joint configuration space, $\mathcal{A}$, since this is the space for physical speed limits on the joint motors.

ToolLinearController addresses the space of 3D poses, $\mathrm{SE}(3)$. Like the JointLinearController, maximum acceleration and target speed is given as optional parameters, but are interpreted as positional scalar Euclidean (L_2-norm) speed and acceleration values for the motion. This exposes a common problem in robotics. There are

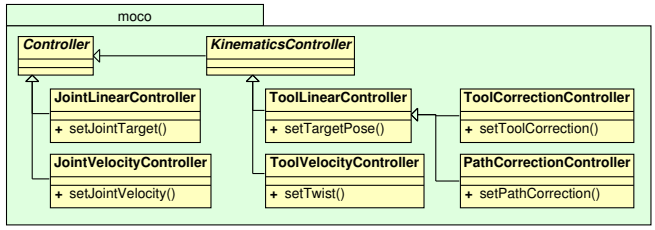


Fig. 12: Class diagram for the canonical motion controllers in PyMoCo.

no universal speed and scalar acceleration defined for $\mathsf{SE}(3)$, and PyMoCo simply restrict the speed and acceleration considerations to $\mathsf{SE}(3)/\mathsf{SO}(3)$; effectively $\mathbb{E}^3$. A resolution for this mixed rotation and displacement measure problem could be brought by introducing a set of suitable length scales from the tool transform, whereby a sensible measure may be introduced on $\mathfrak{se}(3)$; the Lie algebra of $\mathsf{SE}(3)$. The measure would, however, be subject to dynamic changes, if the tool is corrected under motion.

Both motion controllers support proper abortion of an active motion, by the method abort(). During abortion, the desired acceleration limit and the commanded path are respected, and the robot is brought to rest as fast as these allow. Such abortion or cancellation mechanisms are common features in contemporary robot controllers.

In the current implementation, there is no support for corner, path, or trajectory blending in PyMoCo. As a result, the robot arm will come to rest at the endpoints of every linear segment of a composed path. This is a severe lack of functionality for many applications, and it must be addressed in the near future, if PyMoCo is to be considered a viable replacement for the motion control layer of an industrial robot controller. Though slightly mathematically involved, blending algorithms allowing for advanced control of the motion in the blend, have been available for almost two decades. Lloyd and Hayward (1991) [14] introduced the use of blend functions, and describes how the shape of a blend can be flexibly controlled. The main strength of their scheme is that the blend is easy to compute, but the drawback is that it is only described for the positional trajectories; i.e. in $\mathbb{E}^3$. Extending this to blending also orientation was demonstrated later on by Volpe (1993) [15] and Lloyd and Hayward (1993) [16].

Another observation of the implementation is the absence of jerk limitation. The trajectories generated use simple acceleration pulses of the maximum allowed acceleration, implying infinite jerk at both ends of the ramp-up and the ramp-down pulses. Finite-jerk control, as described by Kröger and Wahl (2006) [17], or continuous-jerk control, as described by Kröger and Wahl (2010) [18], is considered a control task below the motion control level addressed by PyMoCo. The jerk control time scale is, for industrial robot control, too small, compared to the $10ms$-order timescale of motion control in PyMoCo. Jerk control is considered relevant at the $1ms$-order timescale *inside* the low-level controller, or even at the individual joint servo controllers.

3.5.4 Real-Time Correction Controllers

The ToolCorrectionController and PathCorrectionController classes are specializations of the ToolLinearController, which can effectuate asynchronous, real-time corrections of the tool transform and the specified linear path, respectively. This functionality is mostly aimed at processing, in which a process model have offline- or online-generated nominal trajectories and a process controller corrects the robot tool or operation path, based on real-time sensor feedback at execution-time. Though this functionality is probably quite application oriented, it is widely usable across many process application areas.

The basic operation of following a linear path, with given speed and acceleration limit, is directly used from the ToolLinearController base class. Superposed on the motion, a tool or path correction may be applied, by the method setToolCorrection or setPathCorrection, respectively. As the main parameter the controller methods take the correction transform, which may be left out to reset the correction to the identity. By another parameter, the dynamics of the correction is addressed in terms of the time of duration to obtain the correction. Since successive corrections are assumed to be small, no ramp-up and ramp-down of the joint speeds are performed on the superposed correction motion.

Any ongoing correction motion may be interrupted by setting a new correction target, or aborted by calling the abortCorrection() on either controller.

3.5.5 Velocity Controllers

The velocity controllers are the most fundamental and primitive. They are useful, if not necessary, in full-information, real-time sensor-based control applications. Due to their primitiveness, a typical application may

not directly use these motion controllers, in the same direct way as the linear controllers. Thus, an application is supposed to set up an *application motion controller*, for managing or controlling the primitive velocity controller.

Both velocity controllers support asynchronous update of the commanded velocity. For every control update the velocity controller sends to the low-level controller, the most recent velocity command is applied in the computation.

The functional method setTwist of the ToolVelocityController is addressed with a target in $\mathfrak{se}(3)$. The representation for the target is chosen conventionally as an array of six floating point numbers, the first three representing the components of linear and the last three representing the components of angular velocities of the tool frame.

The setTwist takes two further optional parameters, expressing in which reference frame the given twist is expressed and in which reference frame the given twist is attached. The valid reference frames for both parameters are "Tool" and "Base", with "Base" being the default for both. The reference frame for expression of the given twist coordinates is of little functional importance, since any external application may transform it to any suitable or required reference frame. In contrast, the selection of attachment reference frame gives highly different behaviour. Attaching the twist to the "Base" reference system gives a normal, intuitive behaviour with the tool centre following a linear path with constant speed, and the tool orientation rotating at constant angular speed around an axis fixed with the base reference frame. When attaching the twist instead in the "Tool" reference system, the twist will be transformed in every frame along with the tool motion, resulting in a helical motion of the tool centre with corresponding rotation.

4 Applications

PyMoCo has been used in a real-time simulation for a production control system application, and in two real sensor- and vision-based control applications. This section shortly described the three applications.

PyMoCo has been most extensively used in a real-time simulation and emulation system, aimed at development of flexible and intelligent production control systems. The simulation and emulation principles are described by Lind and Dubonnet (2010) [19] and in its more complete version will be described by Lind and Skavhaug (2011) [20]. The response time and tracking delay, see Section 2.1.2, of the low-level controller are not modelled in the emulator. Since the simulated application uses only the ToolLinearController class and does not involve any sensor-based control, the absence of modelled delays is irrelevant.

A real path-tracking application, based on PyMoCo and vision-based control, is described in detail by Schrimpf et al. (2010) [21] and is analysed further in Schrimpf et al. (2011) [22]. The application uses the ToolLinearController class for positioning the robot tool during setup and initialization for tasking, and the ToolVelocityController class for the vision-based control loop during tasking motion. The geometric task-setup in the application consists of a surface with a path drawn with a marker. The surface is bulging, shaped into a smooth hilly landscape. A tool-mounted camera observes the tool centre frame, positioned in free space in extension of the tool flange. Through the camera field of view pass the rays of four lasers pointers, directed along the observation direction and originating from the tool flange. Vision algorithms enable the observation of the marked path and, by the laser spots, the relative inclination in of the tool centre frame to the surface. The objectives for the application are to drive the tool centre at constant speed on the observed path while maintaining a fixed orientation of the tool frame to the surface at the tool centre.

A real 6D compliance-control application using a tool-mounted force-torque sensor was shortly described by Lind et al. (2010) [12]. The work presented there was based on an early, monolithic revision of the ComplianceController class, which basically contained a copy of the functionality of the canonical ToolVelocityController. In a later revision, the ComplianceController class was refactored to instead used an associated ToolVelocityController object. This naturally improved the overall code and design reuse; specifically it reduced the code complexity of the ComplianceController class.

A complete overview of the deployed components, their dependencies, and their node associations is seen in Figure 13. On the right is seen the three components of the Nachi AX10 controller, the main controller, the accessor, and the low-level controller, each deployed in a dedicated node. At the top is seen the force reader component deployed on a remote PC, serving dedicated data acquisition from the force-torque sensor. The main node is a PC with Debian GNU/Linux, hosting the PyMoCo and application components.

The sequence diagram in Figure 14 shows the important mechanisms of interactions around the ComplianceController and ToolVelocityController objects under operation. In the sequence diagram, the PyMoCo node is modelled as the system, with the ForceReader and LLCAccessor objects playing roles of external actors. The velocity loop is driven by the reception of status package from the low-level accessor, and the force loop is driven by the reception of force reader updates. The two loops run asynchronously, and the only interaction, the call to the setTwist() of the ToolVelocityController object, may happen at any state in the cycle of the ToolVelocityController object.

15

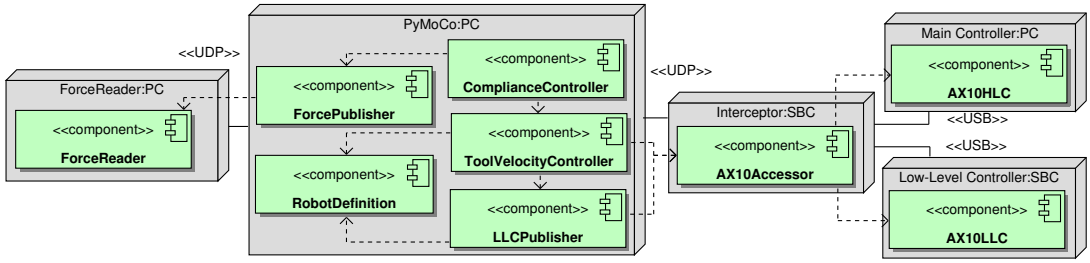


Fig. 13: Deployment diagram for the compliance control application.

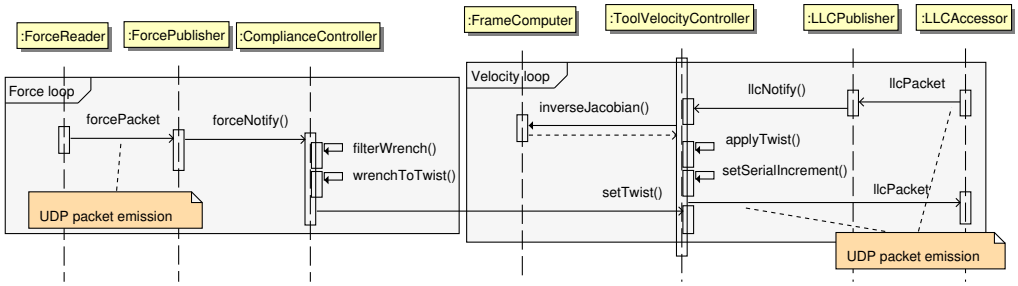


Fig. 14: The main loop sequence for the force-torque compliance control application.

5 Discussion and Future Work

The PyMoCo framework evolved out of experimenting with modifications to a Nachi AX10 controller. As PyMoCo has reached a sufficient level of sophistication and stability, it has become desirable to use it for production automation prototype setups in the laboratory. Thus, the current main focus of future development is to support production control applications with PyMoCo, which may be based on other industrial robot controllers than the Nachi AX10.

The current design suffers from leaving the detailed communication management with the generic classes Controller and LLCPublisher. A refactoring of the mechanism for communicating with different robot types is thus necessary for PyMoCo to become available with a wider range of robot controllers. The functionality of detailed communication must be moved to classes that in return support the interfaces of the two generic classes. This will enable differentiation for communicating with different robot controller, accessors, hardware, protocols, and setups. The analysis and design for such refactoring is work in progress. PyMoCo is currently being adapted for use with the Universal Robots[10] UR-6-85-5-A, and in the near future with KUKA robots over the Robot Sensor Interface.

The linear correction controllers, the ToolCorrectionController and PathCorrectionController classes described in Section 3.5.4, are of quite general use in simple process control. However, they are also of very rigid functionality, compared to what is generally required. A more flexible, though slightly less efficient PluginCorrectionController class, applying the plug-in architectural pattern, is under design. It will support the registration of specializations of a CorrectionPlugin class, which will be implemented by the domain experts for special correction control in a given application. An example of highly specialized correction control is weaving or grove tracking in arc-welding.

For many applications, especially in laboratory prototyping and experimentation, operation cycle times is not a matter of vital concern. However, if PyMoCo should contend to be a motion control framework for real applications, for instance in production control systems, there is an imperative need for facilities to support corner, path, and trajectory blending; as discussed in 3.5.3. The tolerated operation cycle time for a robot in a simple production automation setup is typically low, even though it is composed of many linear motion segments. Meeting such tolerances may be achievable only with efficient trajectory or path blending.

An important point that has not been investigated in the presented work is trajectory accuracy and tolerance of the tool controllers. Among notable conditions that have a negative impact on path or trajectory accuracy are high trajectory speed, low control frequency, and low

10. http://www.universal-robots.com

operation-point dexterity. No features in the canonical controllers support the specification of path or trajectory tolerance, and no strategies have been implemented for handling with path and trajectory deviations. In the future, basic strategies will be implemented for managing accuracy in real-time; such as lowering speed, reiteration of the inverse Jacobian solution in a control step, and optimal extrapolation of a given control step. Related facilities for online and offline pre-checking of a desired trajectory, or computing deviation statistics, will also be analysed and designed.

It is a known fact that Python, by its nature as a dynamically typed language running on an interpreter, is inferior to compiled implementations based on, say, C/C++. It is thus expected that PyMoCo will meet a limit of performance in setups with sufficiently short cycle times or narrow deadlines, whereby it will fail to meet the real-time requirements of the application or of the low-level controller. The contingency plan for dealing with deficient computational efficiency, is to use one or more of the following techniques and technologies:

- Use the Python KDL bindings from Orocos.
- Re-implement the Python code in C/C++ as Python extension modules, possibly with the aid of CXX.
- Annotate the real-time critical code in PyMoCo with type information for use with Cython.

6 Conclusion

Details and specifics of PyMoCo design has been presented. Technological hindrances toward general application to all industrial robots and all application requirements have been touched upon to the degree that it can be projected. The major objectives for future, and current development, have been stated to be redesign of robot communication mechanisms, real-time trajectory segment blending for linear controllers, and management of motion accuracy and tolerance. For tolerant, non-critical operations we conclude that PyMoCo and Python are an effective, maybe even efficient, pair for rapid prototype implementation. This has already been established by a force-torque compliance control application and a vision-based path-tracking application.

Acknowledgements

The authors are grateful to Trygve Thomessen at PPM AS[11] for supporting the development towards access to low-level control in the Nachi AX10 controller, and for valuable comments in the process or writing this paper.

Thanks goes to Thomas Ulleberg at Department for Production Technology, SINTEF Raufoss Manufacturing AS, for good support and involvement.

The Department of Production and Quality Control[12], Norwegian University of Science and Technology, has been hosting and supporting the project of PyMoCo development. Professor Terje Lien is the person at the department responsible for the continual support, and for this we owe our gratitude.

11. http://ppm.no

12. http://www.ntnu.edu/ipk

References

[1] S. Stramigioli and H. Bruyninckx, "Geometry and screw theory for robotics," 2001, tutorial T9 at ICRA 2001. [Online]. Available: http://students.sabanciuniv.edu/acsatici/public_files/GeometryandScrewTheoryforRobotics.pdf 1.1, 2.2

[2] E. B. Dam, M. Koch, and M. Lillholm, "Quaternions, interpolation and animation," Department of Computer Science, University of Copenhagen, Tech. Rep. DIKU-TR-98/5, 1998. [Online]. Available: http://www.diku.dk/DOWNLOAD/98-5.pdf 1.1, 2.2

[3] G. Sally, "Real time," in *Pro Linux Embedded Systems*. Apress, 2010, ch. 12, pp. 257–271. 1.2

[4] P. Leitão, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 979–991, Oct. 2009. 1.2

[5] T. Kröger and F. M. Wahl, "Low-level control of robot manipulators: A brief survey on sensor-guided control and online trajectory generation," in *ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, D. Kubus, K. Nilsson, and R. Johansson, Eds. Technical University of Braunschweig, 2010, pp. 46–53. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010 1.3

[6] D. Dallefrate, D. Colombo, and L. M. Tosatti, "Development of robot controllers based on PC hardware and open source software," in *Seventh Real-Time Linux Workshop*, Nov. 2005. [Online]. Available: https://www.osadl.org/Papers.rtlws-2005-papers.0.html#PAPER_DarioDallefrate 1.3

[7] P. Cederberg, M. Olsson, and G. Bolmsjö, "Remote control of a standard ABB robot system in real time using the Robot Application Protocol (RAP)," in *Proceedings of the 33rd International Symposium on Robotics*, Oct. 2002. 1.5

[8] A. Blomdell, G. Bolmsjö, T. Brogårdh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, A. Robertsson, and J. Wang, "Extending an Industrial Robot Controller: Implementation and Applications of a Fast Open Sensor Interface," *IEEE Robotics Automation Magazine*, vol. 12, no. 3, pp. 85–94, Sep. 2005. 1.5

[9] S. M. Jensen, "Open Modular Controller," in *Proceedings of the 29th International Symposium on Robotics*, 1998. 1.5

[10] H. Bruyninckx, "Open robot control software: the orocos project," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, vol. 3, 2001, pp. 2523–2528. 1.5

[11] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," in *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, vol. 2, Sep. 2003, pp. 2766–2771. 1.5

[12] M. Lind, J. Schrimpf, and T. Ulleberg, "Open Real-Time Robot Controller Framework," in *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, T. K. Lien, Ed. NO-7005, Trondheim, Norway: Tapir Academic Press, Jun. 2010, pp. 13–18. 2.1.2, 4

[13] K. Waldron and J. Schmiedeler, "Kinematics," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 9–33. [Online]. Available: http://www.springerlink.com/content/r6m219/ 3.4

[14] J. Lloyd and V. Hayward, "Real-time trajectory generation using blend functions," in *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, vol. 1, Apr. 1991, pp. 784–789. 3.5.3

[15] R. Volpe, "Task space velocity blending for real-time trajectory generation," in *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, vol. 2, May 1993, pp. 680–687. 3.5.3

[16] J. Lloyd and V. Hayward, "Trajectory generation for sensor-driven and time-varying tasks," *International Journal of Robotics Research*, vol. 12, no. 4, p. 380, Aug. 1993. 3.5.3

[17] T. Kröger, A. Tomiczek, and F. M. Wahl, "Towards on-line trajectory computation," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2006, pp. 736 –741. 3.5.3

[18] T. Kröger and F. M. Wahl, "Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 94 –111, Feb. 2010. 3.5.3

[19] M. Lind and O. Roulet-Dubonnet, "Emulation of Manufacturing Devices for Simulation of Distributed Real-Time Control," in *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, T. K. Lien, Ed. NO-7005, Trondheim, Norway: Tapir Academic Press, Jun. 2010, pp. 67–72. 4

[20] M. Lind and A. Skavhaug, "Using the blender game engine for real-time emulation of production devices," *International Journal of Production Research*, vol. 0, no. 0, pp. 1–17, 2011, online available, iFirst. 4

[21] J. Schrimpf, M. Lind, T. Ulleberg, C. Zhang, and G. Mathisen, "Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control," in *Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems*, T. K. Lien, Ed. NO-7005, Trondheim, Norway: Tapir Academic Press, Jun. 2010, pp. 19–23. 4

[22] J. Schrimpf, M. Lind, and G. Mathisen, "Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking," in *Proceedings of the IEEE/RJS 2011 International Conference on Intelligent Robots and Systems*, Sep. 2011, accepted. 4

Morten Lind received his M.Sc. in computer science and physics from The Maersk McKinney Moller Institute at The University of Southern Denmark in 2000. The ensuing years were used for practical research with robot systems integration and control at the same institute, followed by a period of research and development at the Department of Production Technology, SINTEF Raufoss Manufacturing AS. He is currently finishing his Ph.D.-thesis at the Department of Production and Quality Engineering, The Norwegian University of Science and Technology. His professional interests are widely in the area of soft real-time control in production automation; in particular distributed cooperation and coordination of production devices, and control of industrial robots and AGVs.

Johannes Schrimpf received his M.Sc. in engineering cybernetics from Darmstadt University of Technology and from the Norwegian University of Science and Technology as part of a double degree program in 2009. Since 2010 he attends a Ph.D. programme at the Norwegian University of Science and Technology at the Department of Engineering Cybernetics. His studies focus on real-time sensor-based multi-robot control in manufacturing.

Chapter 5

Conclusions

This chapter summarizes conclusions and contributions of the PhD work presented with this thesis. Finally some future directions regarding the presented work are presented.

5.1 Main Conclusions

There are two main conclusions to be drawn within the presented PhD work, which both are substantiated by implementations and experiments:

Real-Time Production Emulation. For the sake of developing, experimenting, and validating distributed, notably agent-based, control systems in production control, it was demonstrated by a full implementation that the Blender Game Engine is suitable as platform for hosting a device-level real-time emulator for an entire shop-floor. The ZeroC Ice communication middleware has been found very stable as the foundation of a developed middleware for agent-based control systems; called IceHMS. The real-time emulator in the Blender Game Engine is, through implementation in the embedded Python Interpreter, able to present real-time control interfaces through the IceHMS middleware. An Ethernet-distributed control system, using the device interfaces, was developed to demonstrate the satisfactory operation of the emulated shop-floor system. The principles for developing a real-time emulator in the Blender Game Engine are described by Lind and Skavhaug [2011] and Lind and Roulet-Dubonnet [2010].

Robot Motion Control. It is possible to control the motion of an industrial robot arm in soft real-time at $\sim 100Hz$ from a controller implemented on a Python Interpreter running on a standard GNU/Linux system on a standard PC or laptop. The native robot controller must allow a high-frequency access to the low-level joint control of the robot over some fast communication hardware such as Ethernet. The motion control framework PyMoCo, presented by Lind and Schrimpf [2011] and Lind et al.

[2010], provides a set of "canonical" motion controllers, as are commonly found in industrial robot controllers, and the framework have been put to laboratory use as a facility in several production-oriented applications.

A third conclusion is drawn by a literature study conducted as part of this PhD thesis:

Real-Time Linux for Production Control. It is concluded that Real-Time Linux with a GNU system on standard, industrial, or embedded PCs are suitable for taking part in the direct control in production systems. It is further strongly suggested that it may be necessary to bring operating systems as powerful as GNU/Linux and computers as powerful as PCs into such direct control of production devices and systems, for realizing highly agile and intelligent production control.

5.2 Summary of Contributions

The most substantial and tangible contributions of this PhD work reside in produced software, the development of which was entirely or mainly driven and undertaken by the author. In addition, during the PhD scholarship the author contributed to several related software development projects led by collaborators.

The number of lines of code produced by software development projects is often a poor measure for estimating the relative effort and, especially, usefulness of the final software products. However, if disregarding variations in code complexity, it may give a good indication of the comprehensiveness of the code base for a given software product. I.e. the number of code lines may provide a first indication of the estimated effort for a new developer or user to take the software to use. Therefore, the overview of produced software within this PhD work, given in the following, mentions the approximate number of code lines for each project[1]. In the following is given a short summary of each of these software development projects.

The Math3D library comprises about 1000 lines of Python code for basic 3D mathematics operations on the $SE(3)$. It is of central importance to most of the other software projects developed in the PhD work, in providing fundamental and general computational utilities. The Math3D library is notably valuable for

- being of very general nature;
- an efficient implementation based on NumPy [NumPy Website] matrix and array representations and operations;
- and its implementation of interpolation on $SE(3)$ by the Slerp method in the orientation subspace $SO(3)$; confer e.g. Dam et al. [1998] for a good overview of Quaternions and interpolation of orientation.

[1] Generated using "SLOCCount" by David A. Wheeler. Confer [SLOCCount Website].

It is distributed freely under the GPL and available from [PyMath3D Launchpad Website].

The PyMoCo framework for real-time motion control of industrial robots comprises about 3000 lines of Python code. It is adequate for use with any industrial robot arm to which the controller provides low-level access to joint position control at $\sim 100Hz$. The aim with PyMoCo is to establish a framework for fast and flexible development of robotics applications that need advanced real-time interactive control from other systems; such as real-time sensor and vision systems. It provides a set of canonical motion controllers for direct configuration and use, as well as fundamental resources for the development of specialized motion controllers for integration with external, real-time information or sensor systems. PyMoCo is in use in laboratory prototype systems and under continual development for more efficiency and versatility. It is released under the GPL and available at [PyMoCo Launchpad Website].

A real-time emulator for an extended version of the IntelliFeed demonstrator was designed and implemented using the Blender Game Engine. Its embedded Python Interpreter was used extensively as execution platform. The code is split in a general resource library, "emulib", comprising about 600 lines of Python code, and a case specific code base comprising about 1500 lines of Python code. The emulator design emphasizes the realistic and real-time nature of all devices in the demonstrator, and provides an implementation of all emulated devices for supporting the network-exposed interfaces of the corresponding real devices.

The original main motivation for the development of an emulator for the IntelliFeed demonstrator was to make a platform for implementation of real-time shop-floor control available, prior to the completion of the installation of the physical demonstrator in the laboratory. Using the emulator, such a shop-floor control system was implemented for the control of the extended IntelliFeed demonstrator. This experimental control system comprises about 1500 lines of Python code.

5.3 Future Work

The immediately most useful contribution of the presented PhD work is the PyMoCo framework for motion control of industrial robot arms. Its conception evolved out of experiments with a Nachi AX10 controller, extended with a single-board computer acting as an "accessor" to the low-level joint position control. The communication mechanism in PyMoCo still bears some remnant design from this single scenario, and a back-end design for more general and flexible communication mechanisms is in progress.

Advanced future laboratory projects for production automation, regarding such subjects as redundant manipulators and multi-robot manipulation control, are expected to lead to large extensions in functionality and facilities in the PyMoCo framework, as they progress. PyMoCo is hoped to evolve to become a comprehensive framework for prototyping the robot motion control and interaction in advanced production applications.

The well described extended IntelliFeed demonstrator is modelled as the single case for the principles of using the Blender Game Engine as a platform for realistic real-time shop-floor emulation. As this development evolved, more and more code was moved from the application library of modules and packages to the separate *emulib*, a library for generic functionality of real-time device modelling and emulation. This process of separating the general facilities into the emulib library is not entirely completed, and will proceed in the future. When complete, the emulib library will hold a code base and facilities for easily modelling and emulating many kinds of production devices, and it will enable fast development and setup of an emulator for a production shop-floor. Along with this process of separation of general functionality and facilities, and with more prototype cases, a clearer and more detailed documentation of the principles of using the Blender Game Engine for real-time emulation of production systems is expected to emerge.

If it is proven easy to set up a realistic real-time shop-floor emulator, it is expected to have a major impact on the development of highly agile and complex control systems for prototype and pilot installations in production automation. This is a goal for future work with the principles and the code base for production system emulation with the Blender Game Engine.

It has been indicated in this PhD thesis that there are natural performance limits associated with the centralized architecture of the Blender Game Engine for production system emulation. This impacts the feasibility of the described principle of using the Blender Game Engine for emulating large-scale, factory-wide production systems. An objective of future research will be to investigate how a run-time management system may be designed for distributing a real-time emulation of a large scale production system over several Blender Game Engine nodes. This may enable the necessary scalability in the technology for emulating production systems of any scale and size.

References

S. Arthur, C. Emde, and N. McGuire. Assessment of the Realtime Preemption Patches (RT-Preempt) and their impact on the general purpose performance of the system. In *Ninth Real-Time Linux Workshop*, Nov. 2007. URL http://www.realtimelinuxfoundation.org/events/rtlws-2007/ws.html.

F. Auinger, R. Brennan, J. Christensen, J. L. M. Lastra, and V. Vyatkin. Requirements and solutions to software encapsulation and engineering in next generation manufacturing systems: OOONEIDA approach. *International Journal of Computer Integrated Manufacturing*, 18(7):572–585, 2005. doi: 10.1080/09511920500069507.

J. Baumgartner and S. Schoenegger. POWERLINK and Real-Time Linux: A Perfect Match for Highest Performance in Real Applications. In *Twelfth Real-Time Linux Workshop*, Oct. 2010. URL https://www.osadl.org/fileadmin/dam/rtlws/12/Baumgartner.pdf.

P. Blanc, I. Demongodin, and P. Castagna. A holonic approach for manufacturing execution system design: An industrial application. *Engineering Applications of Artificial Intelligence*, 21(3):315–330, 2008. ISSN 0952-1976. doi: 10.1016/j.engappai.2008.01.007.

H. Bruyninckx, P. Soetens, and B. Koninckx. The Real-Time Motion Control Core of The Orocos Project. In *International Conference on Robotics and Automation*, volume 2, pages 2766–2771. IEEE, Sept. 2003. doi: 10.1109/ROBOT.2003.1242011.

G. Bruzzone, M. Caccia, G. Ravera, and A. Bertone. Standard Linux for embedded real-time robotics and manufacturing control systems. *Robotics and Computer-Integrated Manufacturing*, 25(1):178–190, 2009. ISSN 0736-5845. doi: 10.1016/j.rcim.2007.07.016.

J. J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, Upper Saddle River, New Jersey, USA, 3rd edition, 2004. ISBN 0-13-123629-6.

D. Dallefrate, D. Colombo, and L. M. Tosatti. Development of robot controllers based on PC hardware and open source software. In *Seventh Real-Time Linux Workshop*, Nov. 2005. URL https://www.osadl.org/Papers.rtlws-2005-papers.0.html#PAPER_DarioDallefrate.

E. B. Dam, M. Koch, and M. Lillholm. Quaternions, interpolation and animation. Technical Report DIKU-TR-98/5, Department of Computer Science, University of Copenhagen, 1998. URL http://www.diku.dk/DOWNLOAD/98-5.pdf.

S.-T. Dietrich and D. Walker. The Evolution of Real-Time Linux. In *Seventh Real-Time Linux Workshop*, Nov. 2005. URL https://www.osadl.org/fileadmin/events/rtlws-2005/SvenThorstenDietrich.pdf.

D. M. Dilts, N. P. Boyd, and H. H. Whorms. The Evolution of Control Architectures for Automated Manufacturing Systems. *Journal of Manufacturing Systems*, 10(1):79–93, 1991. ISSN 0278-6125. doi: 10.1016/0278-6125(91)90049-8.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, Jan. 1995. ISBN 0201633612.

F. Haghighirad, A. Makui, and B. Ashtiani. Chaos in Production Planning. *Journal of Applied Mathematics and Informatics*, 26:739–750, 2008.

J. Hatvany. Intelligence and cooperation in heterarchic manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 2(2):101–104, 1985. ISSN 0736-5845. doi: 10.1016/0736-5845(85)90065-1.

I. Hegny, O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan. Integrating Software Agents and IEC 61499 Realtime Control for Reconfigurable Distributed Manufacturing Systems. *International Symposium on Industrial Embedded Systems*, pages 249–252, June 2008. doi: 10.1109/SIES.2008.4577710.

S. Josifovska. What next for the PLC? *Manufacturing Engineer*, 83(4):10–11, Aug. 2004. ISSN 0956-9944.

W. Kastner, C. Csebits, and M. Mayer. Linux in factory automation? internet controlling of fieldbus systems! In *International Conference on Emerging Technologies and Factory Automation*, volume 1, pages 27–31. IEEE, IEEE, 1999.

T. D. Khanh, P. Smolík, and P. Píša. An Open Implementation of Profibus DP. In *Eleventh Real-Time Linux Workshop*, Sept. 2009. URL http://www.realtimelinuxfoundation.org/events/rtlws-2009/ws.html.

P. Leitão. Agent-based distributed manufacturing control: A state-of-the-art survey. *Engineering Applications of Artificial Intelligence*, 22(7):979–991, Oct. 2009. ISSN 0952-1976. doi: 10.1016/j.engappai.2008.09.005.

OMG. OMG Unified Modeling Language™ (OMG UML), Superstructure, Version 2.4 , 2010. URL http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF. Online.

P. B. Petersen. The misplaced origin of just-in-time production methods. *Management Decision*, 40(1):82–88, 2002.

E. S. Raymond. The cathedral and the bazaar. *First Monday*, 3(3), Mar. 1998. URL http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/578. Online.

R. Rio. Moving Through the Barriers of Chaotic Manufacturing. ARC Brief on behalf of nMetric, Nov. 2007. URL http://www.nmetric.com/pdfs/ARC_Chaotic_Maufacturing.pdf.

S. Rostedt and D. V. Hart. Internals of the RT Patch. In A. J. Hutton and C. C. Ross, editors, *Proceedings of the Linux Symposium*, volume 2, pages 161–172, June 2007.

A. Rullán. Programmable Logic Controllers versus Personal Computers for Process Control. *Computers and Industrial Engineering*, 33(1-2):421–424, 1997. ISSN 0360-8352. doi: 10.1016/S0360-8352(97)00127-7. Proceedings of the 21st International Conference on Computers and Industrial Engineering.

G. Sally. *Pro Linux Embedded Systems*. Apress, 2010. ISBN 978-1-4302-7226-7. doi: 10.1007/978-1-4302-7226-7.

R. J. Schilling. *Fundamentals of Robotics: Analysis and Control*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1st edition, 1990. ISBN 0-13-344433-3.

S. Shingō. *A Study of the Toyota Production System From an Industrial Engineering Viewpoint*. Productivity Press, Cambridge, Massachusetts, USA, 1989. ISBN 0-915299-17-8.

J. S. Smith. Survey on the Use of Simulation for Manufacturing System Design and Operation. *Journal of Manufacturing Systems*, 22(2):157–171, 2003. ISSN 0278-6125. doi: 10.1016/S0278-6125(03)90013-6.

F. W. Taylor. *The Principles of Scientific Management*. Harper & Brothers, New York, New York, USA, 1913.

K. Thramboulidis. The Function Block Model in Embedded Control and Automation From IEC61131 to IEC61499 . *WSEAS Transactions on Computers*, 8(9), Sept. 2009. ISSN 1109-2750.

H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters. AReference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37 (3):255–274, 1998.

P. Vrba and V. Marík. From Holonic Control to Virtual Enterprises: The Multi-Agent Approach. In R. Zurawski, editor, *The Industrial Information Technology Handbook*. CRC Press, 2005. ISBN 0-8493-1985-4.

Web References

4DIAC Website. URL http://www.fordiac.org/. Home of 4DIAC, open source for distributed industrial automation.

Blender Website. URL http://www.blender.org/. The Blender 3D modelling, animation, and game engine studio.

Bullet Physics Website. URL http://bulletphysics.org/. The home of "Bullet Physics Library", for game physics simulation.

IBM 1984 Archive Website. URL http://www-03.ibm.com/ibm/history/history/year_1984.html. The IBM history archive, 1984.

IMS Website. URL http://www.ims.org/. Home of Intelligent Manufacturing Systems, an industry-led, International business innovation and research and development program.

Linux for Devices Website. URL http://www.linuxfordevices.com/. All About Linuxpowered Devices.

NumPy Website. URL http://numpy.org/. NumPy is the fundamental package needed for scientific computing with Python.

OSADL Website. URL https://www.osadl.org/. The "Open Source Automation Development Lab", open source software for automation and other industries.

O^{3}neida Website. URL http://www.oooneida.org/. Home of O^{3}neida, a network of networks focused on fostering distributed industrial automation based upon open standards.

Pengutronix Website. URL http://pengutronix.de/. Pengutronix supports customers within industrial embedded Linux projects.

PyMath3D Launchpad Website. URL https://launchpad.net/pymath3d. Python Math3D.

PyMoCo Launchpad Website. URL https://launchpad.net/pymoco. Python Robot Motion Control.

Python Website. URL http://www.python.org/. The Python Programming Language.

Qt Website. URL http://qt.nokia.com/. Qt is a cross-platform application and UI framework.

Real-Time Linux Wiki Website. URL https://rt.wiki.kernel.org/. The home of real-time Linux.

SLOCCount Website. URL http://www.dwheeler.com/sloccount/. The home of "SLOCCount", a set of tools for counting physical Source Lines of Code (SLOC) in a large number of languages of a potentially large set of programs.

Steinhoff Website. URL http://www.steinhoff.de/. Steinhoff Automation & Fieldbus-Systems, software components and fieldbus controllers for system application developers.

Tordivel Website. URL http://www.tordivel.com/. Home of the Scorpion Vision Software®.

ZeroC Inc. Website. URL http://www.zeroc.com/. Home of the Internet Communication Engine, Ice™.

Appendix A

Production Control System Code

Selected, central interface to the production emulation system are included and code excerpts from the prototype production control system are included and briefly described in this appendix.

A.1 Interfaces to Emulated Devices

In this section, selected interfaces to controllable devices in the emulated production system are presented. These interfaces are not particularly application specific. The interfaces are well described by in-lined comments.

A.1.1 Workpiece Producer Interface

```
interface WorkpieceProducer extends Holon {
    /**
        Interface to a producer of workpiece. A workpiece is
        indistinguishable from a pickable part; but there may
        exist part types which can not be produced.
    **/

    /// Get the workpiece type that this producer can produce.
    idempotent string getWpType();

    /// Wait for an active production order to complete. If
    /// not completed withing a time of "timeout" seconds have
    /// passed, "False" is returned; otherwise "True"
    idempotent bool waitForIdle(float timeout);

    /// Order the production of "number" workpieces, to be
    /// produced at the given "frequency" in Hertz. If a
    /// produce command is already proceeding, "False" will be
    /// returned.
    bool produce(int number, float frequency);
};
```

A.1.2 Turntable Control Interface

```
interface TurnTableController extends hms::DeviceAdapter{
    /**
        Interface for controlling a simple turntable. The
        turntable has a number of stopping positions, in which
        the controller may stop the table rotation. There is no
        absolute encoding of the positions, and only relative
        motion in terms of a number of stops to pass through
        from the current position is possible.
    **/

    /// Command a rotation through "nStops" stop
    /// positions. Asynchronous.
    bool rotate(int nStops);

    /// Blocks the calling thread until a commanded rotation
    /// completes. Unblocks immediately if there is no
    /// commanded rotation. Wait a maximum time of
    /// timeout. Returns true if the rotation operation ended
    /// withing the timeout limit.
    idempotent bool waitForCompletion(float timeout);

    /// Sets the angular velocity, "alpha" in m/s, to be
    /// maintained during a rotation operation.
    bool setAngularVelocity(float alpha);

    /// Get the currently configured angular velocity.
    idempotent float getAngularVelocity();
};
```

A.1.3 Tool Control Interfaces

ToolController Interface

```
interface ToolController extends hms::DeviceAdapter{
    /**
        Base interface for all tool control interfaces. Only
        supports the static transformation internal to the
        tool; between tool center frame and tool attachment
        frame.
    **/

    /// Retrieve the transformation between the tool attachment
    /// frame and the tool center frame.
    idempotent hms::math3di::Transform getToolTransform();
};
```

BinaryTwoFingerGripper Interface

```
interface BinaryTwoFingerGripper extends ToolController {
    /**
        Interface to the controller of a standard two-finger
        gripper with binary positions for the fingers;
        i.e. either open or closed.
    **/

    /// Command opening of the gripper. Asynchronous.
    bool openGripper();

    /// Command closing of the gripper. Asynchronous.
    bool closeGripper();

    /// Wait for a closing operation to complete; ensuring a
    /// secure grip. If the close operation ends after a time
    /// of "timeout", in seconds, has passed, "False" is
    /// returned.
    idempotent bool waitForClosed(float timeout);

    /// Query if the gripper is in open.
    idempotent bool isOpen();

    /// Query if the gripper is closed.
    idempotent bool isClosed();
};
```

A.1.4 PnF-Conveyor Control Interfaces

PnFDrive Interface

```
interface PnFDrive extends hms::DeviceAdapter {
    /**
        Interface for the control of the drive chain of a
        PnF-conveyor. Speed of the drive is the only
        implemented control.
    **/

    /// Get the current speed, in m/s, of the drive chain.
    idempotent float getSpeed();

    /// Set the speed of the drive chain; in m/s.
    void setSpeed(float speed);
};
```

PnFStop Interface

```
interface PnFStop extends hms::DeviceAdapter {
    /**
        Inteface for controlling a PnF-stop on a
        PnF-conveyor. A stop has an integrated proximity
        sensor, which can determine if a trolley is in position
        at the PnF-stop. All methods with a "timeout"
        parameter will return "True" if the given call succeeds
        within the given time in seconds.
    **/

    /// Get the named location on the conveyor.
    idempotent string location();

    /// Query if the stop is in blocking state.
    idempotent bool isBlocking();

    /// Query the proximity sensor, to determine if a trolley
    /// is at the stop.
    idempotent bool hasTrolleyContact();

    /// Query the controller if the PnF-stop is blocking and
    /// there is a trolley at the proximity sensor.
    idempotent bool hasBlockedTrolley();

    /// Set the stop in blocking mode. Trolleys will be
    /// stopped against the PnF-stop. Returns whether a change
    /// was made to the blocking state.
    bool block();

    /// Unblock the stop. Trolleys will pass freely. Returns
    /// whether a change was made to the blocking state.
    bool unBlock();

    /// Block the PnF-stop and wait for a trolley to arrive.
    /// If "timeout" second passed reached before any trolley
    /// was blocked, False is returned; True otherwise. Option
    /// for asynchronous invokation.
    ["ami"] bool waitForBlockedTrolley(float timeout);

    /// Unblock the stop, and wait for a trolley to leave the
    /// proximity sensor. If a trolley has not left within
    /// "timeout" seconds, False is returned; True
    /// otherwise. Leaves the stop in unblocking
    /// state. Optionally asynchronous method invocation by
    /// callback.
    ["ami"] bool waitForTrolleyLeave(float timeout);

    /// Release a single trolley from the stop. If "timeout"
    /// seconds passes while waiting for a blocked trolley to be
    /// release, False is returned; True otherwise. The stop
    /// is left in blocking state. Option for asynchronous invokation.
```

```
    ["ami"] bool releaseTrolley (float timeout);
};
```

PnFStop Interface

A.1.5 Transport Control Interfaces

Dock Interface

```
/// States of a Dock
enum DockState {Free, Booked, Full};

interface Dock extends ResourceHolon {
    /**
       A dock is a registered target pose of an AGV on the
       shop-floor. The dock manages reservation for allowing
       or rejecting entrance of an AGV. Before entering, an
       AGV must have obtained reservation.
     **/

    /// Get the dock's associated pose on the shop-floor in
    /// world coordinates.
    idempotent agvsys::Pose getPose();

    /// Get the operational cell name that the dock is
    /// associated with; if any. Simple string originating
    /// from the fundamental configuration.
    idempotent string getCellName();

    /// Get the type of dock. Simple string originaging from
    /// the fundamental configuration.
    idempotent string getDockType();

    /// Get access to the fundamental configuration
    /// information for the dock,
    idempotent DockInfo* getDockInfo();

    /// Get the current state of the dock.
    idempotent DockState getDockState();

    /// Query if a given AGV has reservation of the dock.
    idempotent bool haveReservation(AGV* reserver);

    /// Notify that the given AGV is entering the dock. If the
    /// given AGV is not holding reservation, False is
    /// returned; otherwise True is returned. In success, the
    /// dock switches to the full state.
    bool enter(AGV *reserver);

    /// Request reservation of the dock. If the given AGV is
    /// already the reserver, or if the dock is currently
    /// free, the dock is reserved to the AGV and True is
    /// returned; otherwise False is returned.
    bool reserve(AGV* reserver);

    /// Release the dock from reservation to the given AGV. If
    /// the AGV is holding reservations, True is returned and
```

```
    /// the dock changes state to free. Otherwise False is
    /// returned.
    bool release(AGV* reserver);
};
```

Dock Interface

AGV Interface

```
/// Externally controlled load states of an AGV.
enum AGVLoadState {Loaded, Loading, UnLoaded, UnLoading};

/// Internally controlled operational states of an AGV.
enum AGVOperationState {Idle, Buffered, Buffering, Tasking};

/// Internally controlled reservation states of an AGV.
enum AGVReservationState {Reserved, Available};

/// Internally controlled target motion states of an AGV.
enum AGVTargetState {NoTarget, DockTarget, FreeTarget};

interface AGV extends ResourceHolon {
    /**
       Administrative and task-control interface for an AGV as
       a transport unit. Task command methods take two
       specific parameters, "buffer" and "reserver", with the
       following semantics: The buffer argument should be set
       to true if the motion is not a transport-related task,
       wherby the motion task has low priority and may be
       interrupted. The reserver argument must be the
       reservation ID of the current reserver of the AGV;
       otherwise the task will not be acknowledged. If the AGV
       is unreserved, a reserver ID of 0 may be given and the
       AGV acknowledges the task but remain unreserved.
    **/

    /// Set the target pose for the AGV, for free motion
    /// without a specified task.
    bool goToTarget(agvsys::Pose target, bool buffer, int reserver);

    /// Set a dock as a target for the AGV.
    bool goToDock(Dock *targetDock, bool buffer, int reserver);

    /// Get the localizer pose for the AGV.
    idempotent agvsys::Pose getPose();

    /// Wait for completion of the currently active motion
    /// command. Returns "true" if the AGV went idle before
    /// the given "timeout" time; in seconcs.
    idempotent bool waitForIdle(float timeout);

    /// Set and get the load state of the AGV.
    bool setLoadState(AGVLoadState loadState);
    idempotent AGVLoadState getLoadState();

    /// Get and set the payload specification.
    string getPayloadSpec();
    void setPayloadSpec(string payloadSpec);

    /// Get the operation state of the AGV.
```

```
    idempotent AGVOperationState getOperationState();

    /// Get the reservation state of the AGV.
    idempotent AGVReservationState getReservationState();

    /// Get the target motion state of the AGV.
    idempotent AGVTargetState getTargetState();

    /// Query the AGV for a specific reserver.
    idempotent bool haveReservation(int reserver);

    /// Release a reservation of the AGV. Only the current
    /// reserver may release the reservation.
    bool release(int reserver);

    /// Attemt to reserve an AGV with a desired "reserver"
    /// ID. If "reserver" matches the current reserver ID, it
    /// is a no-op. Otherwise, success is only achieved if the
    /// AGV is currently unreserved.
    bool reserve(int reserver);
};
```

TransportManager Interface

```
interface TransportManager extends SupervisorHolon {
    /**
        Interface to a transport manager, which takes partial
        control of all AGVs. Any number of buffer docks may be
        registered with the transport manager. An idle AGV,
        matching the load state associated with a buffer dock,
        may be sent to the dock on expectation of a future
        request. Ordinary docks for tasking are registered with
        their associated production system managers.
    **/

    /// Register a dock, "dck", as a buffer dock associated
    /// with the load state given in "loadState".
    bool registerBufferDock(Dock* dck, AGVLoadState loadState);

    /// Unregister the bufferdock "dck" from the given load state.
    bool unregisterBufferDock(Dock* dck, AGVLoadState loadState);

    /// Request reservation on an AGV in the given "loadState"
    /// and with given "payloadSpec", to be reserved to the
    /// specified "reserver" ID. The caller is responsible for
    /// releasing its reservation of the AGV and updating its
    /// load state and payload specification. The
    /// "payloadSpec" argument is an arbitrary string for
    /// encoding the payload inventory of an AGV. If
    /// "payloadSpec" is "*", any payload specification of an
    /// AGV will match. If an unreserved AGV can not be found,
    /// None is returned. Otherwise an interface to an AGV
    /// with reservation ID as requested is returned.
    AGV* requestAGV(int reserver, AGVLoadState loadState, string payloadSpec);
};
```

A.1.6 Robot Control Interfaces

RobotInfo Interface

```
interface RobotInfo extends hms::DeviceAdapter {
    /**
       Lowest level, administrative interface to a robot. An
       information service for a network connected robot with
       in- and out-ports, robot type, pose, tool methods, etc.
    **/

    /// Get the in- and out-ports to communicated with the
    /// robot over UDP/Ethernet.
    idempotent int llcOutPort();
    idempotent int llcInPort();

    /// Get the robot type.
    idempotent string robotType();

    /// Get the base pose in world coordinates.
    idempotent hms::math3di::Transform baseWorldPose();

    /// Set a given tool, specified by a proxy to its
    /// controller in "toolController", with a given
    /// attachment, "attachXForm".
    void setTool(hms::toolcon::BinaryTwoFingerGripper toolController,
                 hms::math3di::Transform attachXForm);

    /// Query if the robot already has an attached tool.
    idempotent bool hasTool();

    /// Get the attahced tool. Returns None if no tool is
    /// attached.
    idempotent hms::toolcon::BinaryTwoFingerGripper* tool();

    /// Get the attachment transform for the tool. I.e.\ the
    /// transform from the tool base frame to the robot
    /// end-effector frame.
    idempotent hms::math3di::Transform attachXForm();
};
```

ControllerManager Interface

```
interface ControllerManager extends hms::DeviceAdapter {
    /**
       Administrative interface for a robot controller. Basic
       information about robot base location. Control
       interface to, and configuration of the attached tool is
       exposed. Currently only the binary two-finger gripper
       is supported, and for simplification it is given
       directly by its type. Access to interfaces on
       configured motion controllers. No reservation system is
       currently implemented, so care must be taken for
       avoiding race conditions.
    **/

    /// Get the base pose of the robot in world coordinates.
    idempotent hms::math3di::Transform getBaseWorldPose();

    /// Get a tool-linar controller for the robot.
    hms::robcon::LinearController* getLinearController();

    /// Attach a specific tool, given by a proxy for its
    /// controller in "toolController", with a given
    /// attachment transform, "attachXForm", between robot
    /// end-effector and tool base frames.
    void useTool(hms::toolcon::BinaryTwoFingerGripper toolController,
                 hms::math3di::Transform attachXForm);

    /// Get the attached tool, if any. If no tool is attached,
    /// None is returned.
    idempotent hms::toolcon::BinaryTwoFingerGripper* getTool();
};
```

LinearController Interface

```
interface LinearController extends KinematicsController {
    /**
        Interface to a tool-linear controller for a robot. The
        controller may be commanded to go linearly to a target
        in the tool space, and gives access to the configured
        speed on the linear part of the motion. The motion
        operation is asynchronous and have associated methods
        for querying for or synchronizing to completion.
    **/

    /// Go to the operational space home-pose for the robot.*/
    void goHome();

    /// Set and get the linear target speed for motion
    /// commands.
    void setLinSpeed(double linSpeed);
    double getLinSpeed();

    /// Move the tool linearly with the configured speed to
    /// the pose given in "tool", with coordinate reference
    /// given in "reference". "Base" and "World" are supported
    /// as references. If an active motion is proceeding, the
    /// request is rejected and "False" is returned, otherwise
    /// it is acknowledged and "True" is returned.
    bool goTo(hms::math3di::Transform tool, string reference);

    /// Wait for completion of an active motion. If an active
    /// motion does not complete within the given "timeout",
    /// in seconds, "False" is returned; otherwise "True" is
    /// returned.
    idempotent bool waitForIdle(float timeout);

    /// Query methods for an active, executing motion and for
    /// the inverse.
    idempotent bool isExecuting();
    idempotent bool isIdle();

    /// Get the instantaneous pose of the tool. "reference"
    /// must be "Base" or "World".
    idempotent hms::math3di::Transform getPose(string reference);

    /// Get the home pose of the tool, in "Base" reference,
    /// and the home joint-configuration for the robot.
    idempotent hms::math3di::Transform homePose();
    idempotent hms::math3di::Vector homeConfig();
};
```

A.2 Control System Code Excerpts

This sub-appendix includes selected controller classes from the higher level production control system.

A.2.1 SupplyCell Class

```
import time
import threading

import numpy
import math3d

import icehms
from hms.mlagvsys import AGVLoadState, AGVOperationState, DockState

from picker import Picker
from utils import options, transportManager, ti2t, t2ti, pose2transform

class SupplyCell(threading.Thread):
    """ A supply cell object is an orchestrator for the producing and
    providing of workpieces from a CNC machine, or other controllable
    outlet such as a storage system, and bin-picking the workpieces
    from a box on a turntable. The workpieces are placed in a layout
    on a docked AGV, reserved for the supply cell."""

    def __init__(self, vision, controllerManager, dock,
                 partType, partRegistry, partSupplier, turnTable):
        self._logLevel = 2
        self._dock = dock
        self._vision = vision
        self._turnTable = turnTable
        self._partType = partType
        self._ps = partSupplier
        self._cm = controllerManager
        threading.Thread.__init__(self, name='SupplyCell_' \
                        + self._cm.getDescriptor().get('Cell','<NoCell>'))
        self.daemon = True
        self._stop = False
        self._agv = None
        self._agvPose = None
        self._lc = self._cm.getLinearController()
        self._tool = self._cm.getTool()
        self._via = ti2t(self._lc.homePose())
        self._via.pos.y += .1
        self._via.orient.rotateZ(numpy.pi/2)
        self._via = t2ti(self._via)
        self._dObjDrop = 0.05
        self._picker = Picker(self._cm, self._vision, self._partType, partRegistry)
        self._nFillStorage = options.getint('SupplyCell','nFillStorage')
        self._nFillAGV = options.getint('SupplyCell','nFillAGV')
        self._supplyFrequency = options.getfloat('SupplyCell','supplyFrequency')
        self._liftOffFromDrop = options.getfloat('SupplyCell','liftOffFromDrop')
        ## Inclination limit on parts to pick.
        #self._inclLim = numpy.pi / 3.0
        self._inclLim = numpy.pi / 2.0 # Grab anything!

    def _log(self, msg, level=2):
```

```
        if level <= self._logLevel:
            print self.name+' : '+msg

    def _acquireAGV(self, waitForArrival=True):
        """ Acquire an unloaded AGV from the transport manager,
        until one is reserved."""
        ## Only request an AGV, if there is not already one reserved and
        ## associated.
        if not self._agv is None:
            self._log('Warning: Requesting AGV while holding one reserved!.', 2)
            return False
        ## Continue the requesting of an AGV from the transport
        ## manager, until the cell is stopping or an AGV is reserved.
        self._log('Requesting unloaded AGV.', 4)
        while (not self._stop) and self._agv is None:
            self._log('Iterating request of unloaded AGV.', 5)
            #self._agv=transportManager.requestUnLoadedAGV(id(self))
            self._agv=transportManager.requestAGV(id(self), AGVLoadState.UnLoaded, '')
            ## Give som time for an AGV to be released.
            time.sleep(2.0)
        ## Return if the cell is stopping.
        if self._stop:
            return False
        else:
            ## Command the AGV to the dock, and set appropriate load
            ## state.
            self._log('Got reservation on an AGV.', 4)
            self._agv.goToDock(self._dock, buffer=False, reserver=id(self))
            self._agv.setLoadState(AGVLoadState.Loading)
            self._agv.setPayloadSpec(self._partType)
            ## If requested, block until the AGV has arrived.
            if waitForArrival:
                self._agv.waitForIdle(1000.0)
                self._log('Requested AGV arrived', 4)
            return True

    def _releaseAGV(self):
        """ Release the currently reserved AGV."""
        ## Only release, if there is a reserved AGV.
        if not self._agv is None:
            ## Set the correct load-state in which the AGV is
            ## dismissed and release it.
            self._log('Releasing the reserved AGV.', 4)
            self._agv.setLoadState(AGVLoadState.Loaded)
            self._agv.release(id(self))
            self._agv = None
        else:
            self._log('Warning: Called without having an AGV!', 2)

    def _getGrabableParts(self):
        """ Return all grabable parts. The strategy is to filter away
        all visible parts having a 'too horizontal' z-direction of its
        refrence frame. Such parts will have horizontal approach of
```

```
        the tool, and hence high probability for a collision."""
        ## Get all visible workpiece poses.
        partTs = []
        partTs=self._vision.getWorldPoses('WpType:'+self._partType)
        ## Determine the parts that are accessible for grasping by a
        ## limit on the inclination of their z-direction relative to
        ## the world up or down directions.
        grabPartTs = []
        for pT in partTs:
            zp = math3d.Vector(pT.orient[2])
            azp = zp.angle(math3d.Vector.e2)
            ## Check the inclination limit for determining
            ## grabability.
            if azp < self._inclLim or numpy.pi - azp < self._inclLim:
                grabPartTs.append(pT)
        return grabPartTs

    def _selectPart(self):
        """ Select and return the transform for a grabable part for
        picking."""
        ## Get all grabable parts.
        grabPartTs = self._getGrabableParts()
        ## If any parts, return the one with highes reference
        ## position. Otherwise return None.
        if len(grabPartTs) > 0:
            return ti2t(max(grabPartTs, key = lambda part: part.pos[2]))
        else:
            return None

    def _supplyWorkpieces(self):
        """ The turntable is rotated for getting an empty box under
        the producer. Then the producer is ordered to produce. It is
        thus not the ordered products which are supplied for picking,
        but an earlier batch in the box which is immediately rotated
        to the picker."""
        ## Wait for possibly ongoing rotation and produce operations.
        self._turnTable.waitForCompletion(100.0)
        self._ps.waitForIdle(100.)
        ## Rotate a new box to the producer and await arraival.
        self._turnTable.rotate(1)
        self._turnTable.waitForCompletion(100.0)
        ## Command the producing of parts
        self._ps.produce(self._nFillStorage, self._supplyFrequency)

    def _dropOnAGV(self, partSlot=0):
        """ Dispose of a grasped workpiece by dropping it into the
        commanded slot in the pattern to fill on the AGV. Return the
        drop pose."""
        ## Get the AGV pose and compute the (elevated) drop position
        ## for the part slot.
        tagv = pose2transform(self._agv.getPose())
        tdrop = math3d.Transform(tagv)
        tdrop.pos.z = 0.50
```

SupplyCell Class

```
        tdrop.pos += (partSlot * self._dObjDrop - 0) * tdrop.orient.vecX
        tdrop.orient.rotateZ(numpy.pi/2)
        tdrop.orient.rotateY(numpy.pi)
        tdrop.orient.rotateX(.01)
        ## Perform the motion to the drop pose.
        self._lc.setLinSpeed(.5)
        self._lc.goTo(t2ti(tdrop), 'World')
        self._lc.waitForIdle(100.)
        ## Drop the workpiece.
        self._tool.openGripper()
        ## Go to the the lift-off position above the drop pose.
        tdrop.pos.z += self._liftOffFromDrop
        self._lc.goTo(t2ti(tdrop), 'World')
        self._lc.waitForIdle(100.)

    def _fillAGV(self):
        """ Orchestrate the filling of an AGV by selecting, picking,
        and dropping workpieces."""
        ## Reset the slot and number counters.
        agvPosCnt = 0
        N = self._nFillAGV
        while N>0 and not self._stop:
            ## Select a part to pick, refill workpiece storage if necessary
            selPartT = self._selectPart()
            if selPartT is None:
                self._supplyWorkpieces()
            selPartT = self._selectPart()
            if selPartT is None:
                self._log('No grabable part after resupply.',1)
                return False
            ## Command picking of the part, and go to home position.
            self._picker.pick(selPartT)
            self._lc.goTo(self._via, '')
            self._lc.waitForIdle(100.)
            ## Command dropping of the part in the next part slot and go to home.
            self._dropOnAGV(agvPosCnt)
            self._lc.goTo(self._via, '')
            self._lc.waitForIdle(100.)
            ## Update counters and re-iterate.
            agvPosCnt = (agvPosCnt + 1)%9
            N-=1
        return True

    def stop(self):
        """ Command for ending the supply cell operation."""
        self._stop = True

    def run(self):
        self._lc.goTo(self._via, '')
        while not self._stop:
            ## Acquire a new AGV
            self._acquireAGV(waitForArrival=False)
            ## Resupply the storage, while waiting for the AGV.
```

SupplyCell Class

```
while len(self._getGrabableParts()) < 2:
    self._supplyWorkpieces()
## Await the arrival of the AGV.
self._agv.waitForIdle(1000.0)
## Perform the filling of the AGV and release it.
self._fillAGV()
self._releaseAGV()
```

A.2.2 UploadCell Class

```
import time
import threading

import numpy
import math3d

import icehms
from hms.mlagvsys import AGVLoadState, AGVOperationState, DockState

from picker import Picker
from utils import options, transportManager, ti2t, t2ti

class UploadCell(threading.Thread):

    def __init__(self, controllerManager, carrierManager,
                 dockVision, partType, partRegistry, dock, logLevel=2):
        self._logLevel = logLevel
        self._stop = False
        self._conMan, self._carMan, self._dockVision, self._partType, self._dock \
                    = controllerManager, carrierManager, dockVision, partType, dock
        threading.Thread.__init__(self, name='UploadCell_'+self._conMan.getName())
        self.daemon = True
        ## Find out what side of the carrier this upload cell belongs.
        if self.name.find('Left')>0:
            self._side = 'left'
        elif self.name.find('Right')>0:
            self._side = 'right'
        ## Register with the carrier manager on the correct side.
        self._carMan.registerUploadCell(self, self._side)
        ## Variable for holding the interface to a reserved AGV.
        self._agv = None
        ## Get a handle on the tool and robot linear controllers.
        self._tool =self._conMan.getTool()
        self._linCon = self._conMan.getLinearController()
        ## Cpmpute a via pose between AGV and the carrier.
        via = ti2t(self._linCon.homePose())
        via.orient.rotateYB(numpy.pi/2)
        if self._side == 'left':
            via.pos += math3d.Vector(-1.0,0.6,-0.75)
        elif self._side == 'right':
            via.pos += math3d.Vector(-1.0,-0.6,-0.75)
        self._via = t2ti(via)
        ## Setup the picker.
        self._picker = Picker(self._conMan,
                              self._dockVision, self._partType,
                              partRegistry, via=self._via)
        ## Get the retract length for the approach for uploading a
        ## workpiece to a site on the carrier.
        self._approachRetract = options.getfloat('UploadCell','approachRetract')
        ## Running variable for counting the number of uploads on a
        ## carrier.
```

```
        self._nUploaded = 0

    def _log(self, msg, level=2):
        if level <= self._logLevel:
            print self.name+' : '+msg

    def _acquireAGV(self, waitForArrival=True):
        """ Acquire a loaded AGV from the transport manager.
        Keep requesting until one is reserved."""
        ## Only request an AGV, if there is not already one reserved and
        ## associated.
        if not self._agv is None:
            self._log('Warning: Requesting AGV while holding one reserved!.', 2)
            return False
        ## Continue the requesting of an AGV from the transport
        ## manager, until the cell is stopping or an AGV is reserved.
        self._log('Requesting unloaded AGV.', 4)
        while (not self._stop) and self._agv is None:
            self._log('Iterating request of unloaded AGV.', 5)
            #self._agv=transportManager.requestLoadedAGV(id(self))
            self._agv=transportManager.requestAGV(id(self),
                                                  AGVLoadState.Loaded, self._partType)
            ## Give som time for an AGV to be released.
            time.sleep(2.0)
        ## Return if the cell is stopping.
        if self._stop:
            return False
        else:
            ## Command the AGV to the dock, and set appropriate load
            ## state.
            self._log('Got reservation on an AGV.', 4)
            self._agv.goToDock(self._dock, buffer=False, reserver=id(self))
            self._agv.setLoadState(AGVLoadState.UnLoading)
            ## If requested, block until the AGV has arrived.
            if waitForArrival:
                self._agv.waitForIdle(1000.0)
                self._log('Requested AGV arrived', 4)
            return True

    def _releaseAGV(self):
        """ Release the currently reserved AGV."""
        if not self._agv is None:
            self._agv.setLoadState(AGVLoadState.UnLoaded)
            self._agv.release(id(self))
            self._agv = None

    def _agvWorkpieceCount(self):
        """ Return a count of remaining identifiable workpieces on the
        AGV."""
        return len(self._dockVision.getWorldPoses('WpType:'+self._partType))

    def _uploadWorkpiece(self):
        """ Upload the currently picked workpiece to the carrier."""
```

UploadCell Class

```
        ## Get a site. The call blocks until a new carrier arrives, if
        ## there are no remaining sites for this uploader's side.
        wpT= self._carMan.getSite( self._side )
        ## Compute the pose to put the workpiece onto the site.
        putT = wpT.copy()
        if putT.orient.vecY.y < 0.0:
            putT.orient.rotateXT(-numpy.pi/2)
            putT.orient.rotateZT(-numpy.pi/3.9)
        else:
            putT.orient.rotateXT(numpy.pi/2)
            putT.orient.rotateZT(-2.2*numpy.pi/3)
        ## Compute the approach pose for attaching the workpiece.
        apprT = putT.copy()
        apprT.pos += self._approachRetract * wpT.orient.vecZ
        ## Perform the motions for attaching the workpiece.
        self._linCon.setLinSpeed(1.0)
        self._linCon.goTo(t2ti(apprT),'World')
        self._linCon.goTo(t2ti(putT),'World')
        self._linCon.waitForIdle(100.)
        self._linCon.setLinSpeed(0.1)
        self._linCon.goTo(t2ti(putT),'World')
        self._linCon.waitForIdle(100.)
        self._tool.openGripper()
        self._nUploaded += 1
        self._linCon.setLinSpeed(1.0)
        self._linCon.goTo(self._via,'')
        self._linCon.waitForIdle(100.)
        ## Notify the carrier manager of the filled site.
        self._carMan.siteFilled(wpT)

    def stop( self ):
        self._stop = True

    def run( self ):
        ## Initially command the robot to the via point for operation.
        self._linCon.goTo(self._via,'')
        while not self._stop:
            ## Ensure that an ordered AGV has docked.
            if not self._agv is None:
                self._agv.waitForIdle(1000.0)
            ## Try picking a workpiece.
            if not self._picker.pick():
                ## If unsuccessful pick, get a new AGV
                self._releaseAGV()
                self._acquireAGV(waitForArrival=False)
            else:
                ## If no further workpieces, wait for a new AGV while
                ## uploading.
                if self._agvWorkpieceCount() == 0:
                    self._releaseAGV()
                    self._acquireAGV(waitForArrival=False)
                ## Perform the uploading.
                self._uploadWorkpiece()
```

UploadCell Class

A.2.3 Picker Class

```
import copy

import icehms
from hms import math3di

from utils import ti2t, t2ti

class Picker(object):
    """ A picker is an association of a robot, a vision system, and a
    part type. The part type must be registered with a part registry,
    for retrieving grasping information. An optional via point can be
    given, for the robot to move to after a pick."""

    def __init__(self, controllerManager, vision,
                 partType, partRegistry, via=None):
        self._via = via
        self._cm = controllerManager
        self._v = vision
        self._lc = self._cm.getLinearController()
        self._t = self._cm.getTool()
        self._partType = partType
        self._partHolon = partRegistry.getPart(self._partType)
        self._grasps = [ti2t(g) for g in self._partHolon.getGrasps()]
        self._tb2w = ti2t(self._cm.getBaseWorldPose())
        self._tw2b = self._tb2w.inverse()

    def _selectGrasp(self, tp):
        """ Strategy for selecting grasp, given the part pose."""
        ztipz = [(tp*g.orient.vecZ).z for g in self._grasps]
        i = min(enumerate(ztipz), key=lambda x:x[1])[0]
        return self._grasps[i]

    def pick(self, partT=None):
        """ The method to command a pick of a part of the given
        type. If a part transform, 'partT' in world coordinates, is
        not given, select a part found with the vision system."""
        ## Prepare.
        self._t.openGripper()
        self._lc.waitForIdle(20.)
        ## If a part transform is not given, chose one.
        if partT is None:
            partTs = self._v.getWorldPoses('WpType:'+self._partType)
            if len(partTs) > 0:
                partT = partTs[0]
            else:
                return False
        if type(partT) == math3di.Transform:
            tPartT = ti2t(partT)
        else:
            tPartT = partT
        ## Transform to robot base reference
```

```
tPartB = self._tw2b*tPartT
## Select a grasp and transfrom to robot base coordinates.
self._g =  self._selectGrasp(tPartT)
tGraspB = tPartB * self._g
## Set a high via speed and go to approach.
self._lc.setLinSpeed(.7)
tAppr = copy.copy(tGraspB)
tAppr.pos -= 0.05*tAppr.orient.vecZ
self._lc.goTo(t2ti(tAppr),'')
self._lc.waitForIdle(20.)
## Set a low speed, move to the grasp, and pick.
self._lc.setLinSpeed(0.3)
self._lc.goTo(t2ti(tGraspB),'')
self._lc.waitForIdle(20.)
self._t.closeGripper()
self._t.waitForClosed(10000.)
## Lift out to approach via.
self._lc.goTo(t2ti(tAppr),'')
self._lc.waitForIdle(20.)
## Move away to via, if configured.
self._lc.setLinSpeed(.7)
if not self._via is None:
    self._lc.goTo(self._via,'')
    self._lc.waitForIdle(20.)
return True
```

A.2.4 TransportManager Class

```
import threading
import time
import random

import icehms
import hms
from hms import mlagvsys, math3di
from hms.mlagvsys import AGVOperationState, AGVReservationState, AGVLoadState

class TransportManager(mlagvsys.TransportManager, icehms.Holon):

    def __init__(self):
        icehms.Holon.__init__(self,name='TransportManager')
        ## Separate task thread for updating the list of known AGVs.
        self._logLevel = 2
        self._stop = False
        ## Lock for any resource access.
        self._resourceLock = threading.Lock()
        ## The register of buffer docks.
        self._bufferDocks={}
        ## The list of known AGVs.
        self._agvs = []

    def _log(self, msg, level=2):
        if level <= self._logLevel:
            print self.name+' : '+msg

    def _updateResources(self):
        """ Refresh the list of known AGVs in the system. Some may
        have shut down, and new may have been deployed."""
        with self._resourceLock:
            ## Refresh the list of AGVs available.
            self._agvs = [mlagvsys.AGVPrx.checkedCast(x)
                          for x in self._icemgr.findHolons(' :: hms::mlagvsys::AGV')]

    def registerBufferDock(self, dck, loadState, current=None):
        """ Register 'dck' as a buffer dock for idle AGVs of the given
        'loadState'."""
        if not loadState in self._bufferDocks:
            self._bufferDocks[loadState] = []
        if not dck in self._bufferDocks[loadState]:
            self._bufferDocks[loadState].append(dck)
            return True
        else:
            return False

    def unregisterBufferDock(self, dck, loadState, current=None):
        """ Unregister 'dck' as a buffer dock for idle AGVs of the given
        'loadState'."""
        if not loadState in self._bufferDocks:
            return False
```

```
        if dock in self._bufferDocks[loadState]:
            self._bufferDocks[loadState].remove(dck)
            if len(self._bufferDocks[loadState]) == 0:
                del self._bufferDocks[loadState]
            return True
        else:
            return False

    def requestAGV(self, reserver, loadState, payloadSpec='*', current=None):
        """ Request for reservation of an AGV. This includes no
        ordering of motion, but only reservation. One attempt is made
        to find an AGV which is available and in the requested
        'loadState'. Upon success, the AGV is reserved to the
        'reserver' id and an inteface to the AGV is
        returned. Otherwise None is returned."""
        reserved = None
        random.shuffle(self._agvs)
        with self._resourceLock:
            for a in self._agvs:
                if (a.getOperationState() != AGVOperationState.Tasking
                    and a.getReservationState() == AGVReservationState.Available
                    and a.getLoadState() == loadState):
                    if (loadState != AGVLoadState.UnLoaded and
                        payloadSpec != '*' and
                        a.getPayloadSpec() != payloadSpec):
                        ## The requested "payloadSpec" was not set to
                        ## any payload ("*") and it did not match the
                        ## payloadSpec of the AGV. So continue
                        ## searching.
                        continue
                    else:
                        reserved = a
                        a.reserve(reserver)
                        break
        return reserved

    def _getBufferables(self):
        """ Return a list of AGVs that are idle and available;
        i.e. suitable for buffering."""
        return [a for a in self._agvs if
                (a.getOperationState() == AGVOperationState.Idle and
                 a.getReservationState() == AGVReservationState.Available)]

    def _bufferAGVs(self):
        """ Send idle, available AGVs to appropriate buffers."""
        with self._resourceLock:
            iaAGVs = self._getBufferables()
            for a in iaAGVs:
                self._log('TransportManager: Buffering available, idle AGV', 5)
                ## Check the AGV for its load state, and send to a random,
                ## appropriate buffer dock.
                if (a.getLoadState() == AGVLoadState.Loaded
                    and AGVLoadState.Loaded in self._bufferDocks):
```

```
                    a.goToDock(random.choice(
                        self._bufferDocks[AGVLoadState.Loaded]),
                            buffer=True, reserver=0)
                elif (a.getLoadState() == AGVLoadState.UnLoaded
                      and AGVLoadState.UnLoaded in self._bufferDocks):
                    a.goToDock(
                        random.choice(
                            self._bufferDocks[AGVLoadState.UnLoaded]),
                        buffer=True, reserver=0)

    def run(self):
        ## Do an initial search for AGVs.
        self._log('Initial update of resources', 4)
        self._updateResources()
        ## Timer for the the resource updater.
        _tResUpd = time.time()
        while not self._stop:
            ## Update AGV list at very low frequency
            if time.time() - _tResUpd > 10.0:
                self._updateResources()
                _tResUpd = time.time()
            ## Continually buffer idle, available AGVs.
            self._bufferAGVs()
            ## Run tasks at low frequency.
            time.sleep(1.0)
```

A.2.5 AGV Class

```
import threading
import time

import numpy
import Ice

import icehms
from hms import mlagvsys, agvsys
from hms.mlagvsys import \
     AGVLoadState, AGVOperationState, AGVReservationState, AGVTargetState

from utils import pose2transform

class _DockMotion(threading.Thread):
    def __init__(self, agv, dock):
        self._agv, self._dock = agv, dock
        self._target = pose2transform(self._dock.getPose())
        self._dockReserved = False
        self._entered = False
        threading.Thread.__init__(self, name='DockMotion'+agv.name)
        self.daemon = True
        self._stop = False
        self._vlin = 0.75 # m/s
        self._standoffstart = 1.5 # m
        self._standoffdist = 1.0 # m
        self._enterdist = 0.7 # m
        self._vrot = 1.5 # rad/s
        self._ltol = 0.05 # m
        self._rtol = 0.02 # m
        self.start()

    def dockReserved(self):
        self._agv._log('DockMotion: received dock reservation notification',4)
        self._dockReserved = True

    def stop(self):
        self._stop = True

    def run(self):
        while not self._stop:
            pose=pose2transform(self._agv._loc.getPose())
            tvec = self._target.pos - pose.pos
            tdist = tvec.length()
            if (not self._entered) and self._dockReserved and tdist < self._enterdist:
                self._agv._log('DockMotion: target dist (%.2f) within enter dist (%.2f)'
                               % (tdist, self._enterdist),4)
                self._dock.enter(self._agv.proxy)
                self._entered = True
            heading = pose.orient.vecX
            vcut = min(self._vlin, tvec*heading)
            vlin = max(0.0,vcut)
```

```
            if tdist <0.05:
                ang = heading.sangle( self . _target . orient .vecX)
            else :
                ang = heading.sangle(tvec)
            vrot = min(self. _vrot , 2*ang)
            if tdist < self. _ltol and abs(ang) < self. _rtol :
                self ._agv. _vc. setPolarVelocity (agvsys. PolarVelocity (0.0,0.0))
                break
            else :
                if (not self ._dockReserved) and tdist < self. _standoffstart :
                    self ._agv. _log ('DockMotion: target dist (%.2f) within standoff '
                                      + 'start (%.2f) and dock not reserved'
                                      % (tdist , self . _standoffstart ), 4)
                    vlin *= ((tdist - self. _standoffdist )
                             /( self . _standoffstart - self. _standoffdist ))
                self ._agv. _vc. setPolarVelocity (agvsys. PolarVelocity ( vlin , vrot ))
            time. sleep (0.2)
        if not self . _stop :
            # Notify AGV
            self ._agv. _tcDoneFlag.set ()


class _FreeMotion(threading. Thread):
    """ A motion task for obtaining a freely specified pose in world
    coordinates . 'target' must be given as a agvsys. Pose object ."""
    def __init__ ( self , agv, target ):
        threading. Thread. __init__ ( self , name='FreeMotion'+agv.name)
        self .daemon = True
        self ._agv = agv
        ## Store target as a math3d.Transform
        self . _target = pose2transform(target)
        self . _stop = False
        self . _vlin = 0.75
        self . _vrot = 1.5
        self . _ltol = 0.05
        self . _rtol = 0.02
        self . start ()

    def stop( self ):
        self . _stop = False

    def run( self ):
        while not self . _stop :
            pose=pose2transform(self. _agv. _loc .getPose())
            tvec = self . _target .pos - pose.pos
            tdist = tvec. length ()
            heading = pose. orient .vecX
            vcut = min(self. _vlin , tvec*heading)
            vlin = max(0.0,vcut)
            if tdist <0.05:
                ang = heading.sangle( self . _target . orient .vecX)
            else :
                ang = heading.sangle(tvec)
```

```
            vrot = min(self._vrot, 2*ang)
            if tdist < self._ltol and abs(ang) < self._rtol:
                self._agv._vc.setPolarVelocity(agvsys.PolarVelocity(0.0,0.0))
                break
            else:
                self._agv._vc.setPolarVelocity(agvsys.PolarVelocity(vlin,vrot))
            time.sleep(0.2)
        if not self._stop:
            # Notify AGV
            self._agv._tcDoneFlag.set()


class AGV(mlagvsys.AGV, icehms.Holon):

    def __init__(self, vc, loc, logLevel=3):
        icehms.Holon.__init__(self, name='AGV_'+vc.getName(), logLevel=logLevel)
        self._vc = vc
        self._loc = loc
        self._target = None
        self._reserver = 0
        self._payloadSpec = ''
        self._stop = False
        self._idle = threading.Event()
        self._loadState = AGVLoadState.UnLoaded
        self._opState = AGVOperationState.Idle
        self._resvState = AGVReservationState.Available
        self._targetState = AGVTargetState.NoTarget
        self._targetController = None
        self._targetDock = None
        self._targetLock = threading.RLock()
        self._tcDoneFlag = threading.Event()

    def __getattr__(self, name):
        if name == 'idle':
            return self._idle.isSet()
        elif name == 'reserved':
            return self._reserver!=0
        else:
            return self.__dict__[name]

    def __setattr__(self, name, val):
        if name == 'idle':
            if bool(val):
                self._idle.set()
            else:
                self._idle.clear()
        else:
            object.__setattr__(self, name, val)

    def _targetControllerDone(self):
        """Handler for state updates on accomplished target control."""
        with self._targetLock:
            ## Flag for now idle.
```

```
            self._idle.set()
            ## If the operation was buffering, then switch to
            ## buffered.
            if self._opState == AGVOperationState.Buffering:
                self._opState = AGVOperationState.Buffered
            else:
                self._opState = AGVOperationState.Idle

    def goToTarget(self, target, buffer=True, reserver=0, current=None):
        """ Setup a task for going to a freely specified target."""
        self._log('Request for target pose',4 )
        ## Reserver must have reservation and the AGV must not be tasking.
        if reserver != self._reserver or self._opState == AGVOperationState.Tasking:
            ## Log the reason for rejection.
            if reserver != self._reserver:
                self._log('Request rejected due to mismatch in reservation.', 3)
            if self._opState == AGVOperationState.Tasking:
                self._log('Request rejected due to tasking.', 3)
            ## Reject the task.
            return False
        with self._targetLock:
            ## Release a current target dock if reserved.
            if ((not self._targetDock is None)
                and self._targetDock.haveReservation(self.proxy)):
                self._log('Releasing current target dock: "%s"'%str(self._targetDock), 4)
                self._targetDock.release(self.proxy)
                self._targetDock = None
            ## Stop the current target control, if existing.
            if not self._targetController is None:
                self._log('Stopping current target controller.', 4)
                self._targetController.stop()
                self._targetController.join()
            ## Save the new target.
            self._target = target
            ## Update target and operation states
            self._targetState = AGVTargetState.FreeTarget
            if buffer:
                self._opState = AGVOperationState.Buffering
            else:
                self._opState = AGVOperationState.Tasking
            ## Start operating.
            self._idle.clear()
            ## Start target controller.
            self._targetController = _FreeMotion(self, target)
            self._log('New target controller: "%s"'%str(self._targetController), 4)
            return True

    def goToDock(self, targetDock, buffer=True, reserver=0, current=None):
        """ Setup of task for going to the specified dock."""
        self._log('Request for target dock', 4)
        ## Reserver must have reservation and the AGV must not be tasking.
        if reserver != self._reserver or self._opState == AGVOperationState.Tasking:
            ## Log the reason for rejection.
```

```
            if reserver != self._reserver:
                self._log('Request rejected due to mismatch in reservation.', 3)
            if self._opState == AGVOperationState.Tasking:
                self._log('Request rejected due to tasking.', 3)
            ## Reject the task.
            return False
        with self._targetLock:
            ## Release a current target dock if reserved.
            if ((not self._targetDock is None)
                and self._targetDock.haveReservation(self.proxy)):
                self._log('Releasing current target dock: "%s"'%str(self._targetDock), 4)
                self._targetDock.release(self.proxy)
                self._targetDock = None
            ## Stop the current target control, if existing.
            if not self._targetController is None:
                self._log('Stopping current target controller.', 4)
                self._targetController.stop()
                self._targetController.join()
            ## Save the new target.
            self._targetDock = targetDock
            ## Update target and operation states
            self._targetState = AGVTargetState.DockTarget
            if buffer:
                self._opState = AGVOperationState.Buffering
            else:
                self._opState = AGVOperationState.Tasking
            ## Start operating.
            self._idle.clear()
            ## Start target controller.
            self._targetController = _DockMotion(self, self._targetDock)
            self._log('New target controller: "%s"'%str(self._targetController), 4)
            return True

    def getPose(self, current=None):
        """ Return the instantaneous pose of the AGV in world
        coordinates, according to the localizer."""
        return self._loc.getPose()

    def stop(self):
        """ Shutdown method for the AGV."""
        ## Stop a current target controller.
        if not self._targetController is None:
            self._targetController.stop()
            self._targetController.join()
        ## Flag for stopping to the thread.
        self._stop = True

    def waitForIdle(self, timeout=None, current=None):
        """ Utility for clients to be blocked until the AGV is
        idle. Wait a maximum of 'timeout' seconds for the current
        operation to complete. Returns the idle status."""
        self._idle.wait(timeout)
        return self._idle.isSet()
```

```
    def getLoadState(self, current=None):
        """ Get the current load state."""
        return self._loadState

    def setLoadState(self, loadState, current=None):
        """ Set the load state. (Only used in external logic.)"""
        self._loadState = loadState
        if loadState == AGVLoadState.UnLoaded:
            self._payloadSpec = ''
        return True

    def getPayloadSpec(self, current=None):
        """ Get the current payload specification."""
        return self._payloadSpec

    def setPayloadSpec(self, payloadSpec, current=None):
        """ Set the payload specification. (Only used in external
        logic.)"""
        self._payloadSpec = payloadSpec

    def getOperationState(self, current=None):
        """ Get the current operation state."""
        return self._opState

    def getReservationState(self, current=None):
        """ Get the current reservation status."""
        return self._resvState

    def getTargetState(self, current=None):
        """ Return the current target state. This specifies if there
        is no target, the target is a dock, or the target is a general
        pose."""
        return self._targetState

    def release(self, reserver, current=None):
        """ Release a reservation of the AGV. If the addressing
        'reserver' does not match the current reserver, False is
        returned."""
        if self._reserver == reserver:
            self._reserver = 0
            self._resvState = AGVReservationState.Available
            return True
        else:
            return False

    def reserve(self, reserver, current=None):
        """ Request reservation of the AGV. If the AGV is already
        reserved, or if the 'reserver' is not the current reserver,
        False is returned."""
        if self._reserver == 0 or self._reserver == reserver:
            self._reserver = reserver
            self._resvState = AGVReservationState.Reserved
```

```
            return True
        else:
            return False

    def haveReservation(self, reserver, current=None):
        """ Query if a given 'reserver' is the current reserver of the
        AGV."""
        return self._reserver == reserver

    def run(self):
        while not self._stop:
            if self._tcDoneFlag.isSet():
                ## The current target controller is done. Handle the
                ## arrival to target.
                self._targetControllerDone()
                self._tcDoneFlag.clear()
            if not self.idle:
                ## Under operation. Handle running tasks.
                if not self._targetController is None:
                    self._log('Have target controller',4)
                    if self._targetController.isAlive():
                        self._log('Target controller alive',4)
                        ## Check if the target is an unreserved dock.
                        if type(self._targetController) == _DockMotion \
                                and not self._targetController._dockReserved:
                            with self._targetLock:
                                ## Try to obtain reservation
                                self._log('Trying to reserve dock: "%s"'
                                          % str(self._targetDock), 4)
                                if (not self._targetDock is None
                                    and self._targetDock.reserve(self.proxy)):
                                    ## Reservation has been obtained.
                                    ## Notify the target dock controller.
                                    self._targetController.dockReserved()
                ## Moderate frequency activity when not idle.
                time.sleep(1.0)
            else:
                ## Low frequency when idle.
                time.sleep(1.0)
```

A.2.6 CarrierManager Class

```
import threading
import time

import numpy

from utils import options, ti2t, t2ti

class CarrierManager(threading.Thread):
    """ Class for the manager of the carrier at the upload
    PnF-stop. Upload cells may register to cover the sites of a
     particular side of a carrier. An upload cell is given, on request,
    the 3D pose of any side requested. If no sites are free for the
     pertinent side, the call is blocked until notified about new
     sites. It is part of the carrier managers life cycle to recognize
    the situation where all sites with potential to be filled, with
     respect to the registered uploaders, have been filled, and react
    by acquiring a new carrier."""

    def __init__(self, carrierVision, conveyorManager):
        threading.Thread.__init__(self, name='CarrierManager')
        self.daemon = True
        self._carVis = carrierVision
        self._convMan = conveyorManager
        self._sites = {'left':[], 'right':[]}
        self._reservedSites = []
        self._sitesCond = threading.Condition()
        self._filledEvent = threading.Event()
        self._stop = False
        self._uploaders = {}
        #self._uploaderWait = {}
        self._nUploadsPerCarrier = options.getint('CarrierManager', 'nUploadsPerCarrier')

    def _partitionSites(self):
        """ Separate all identifiable sites of a carrier into two sides."""
        with self._sitesCond:
            siteTs = [ti2t(s) for s in self._carVis.getWorldPoses('')]
            ## Separate sites for left and right by center
            centerX = numpy.average([s.pos.x for s in siteTs])
            leftTs = [s for s in siteTs if s.pos.x > centerX]
            rightTs = [s for s in siteTs if s.pos.x < centerX]
            ## Sort and trim site lists to fit number of desired uloads.
            rightTs.sort(key=lambda s:s.pos.z)
            halfNUploads = (self._nUploadsPerCarrier + 1) // 2
            if halfNUploads < len(rightTs):
                rightTs = rightTs[:halfNUploads]
            leftTs.sort(key=lambda s:s.pos.z)
            if halfNUploads < len(leftTs):
                leftTs = leftTs[:halfNUploads]
            self._sites['left'] = leftTs
            self._sites['right'] = rightTs
```

```
def _siteCount(self):
    """ Count the remaining sites for the carrier to be full. This
    comprises all reserved sites and free sites for which an
    uploader is registered."""
    count = len(self._reservedSites)
    for side in self._uploaders:
        count += len(self._sites[side])
    return count

def registerUploadCell(self, uploader, side):
    """ Register an upload cell for a given side of a carrier."""
    self._uploaders[side] = uploader
    #self._uploaderWait[uploader] = False

def getSite(self, side):
    """ Return a site for the given 'side' of the carrier. If
    there are no sites available, block and wait for a
    notification about new sites. If then still no sites are found
    for the 'side', None is returned. Otherwise, the site pose is
    returned. The site is moved to the reserved sites list."""
    with self._sitesCond:
        if len(self._sites[side]) == 0:
            ## No available sites, wait for notification about new
            ## sites on a new carrier.
            self._sitesCond.wait()
        if len(self._sites[side]) > 0:
            ## Pop a site from available sites for the 'side', and
            ## add it to the reserved list.
            site = self._sites[side].pop(0)
            self._reservedSites.append(site)
            return site
        else:
            ## If still no new sites, return None.
            return None

def siteFilled(self, site):
    """ Notification from a client, that a reserved site have been
    filled. The site is removed from the reserved sites list."""
    self._reservedSites.remove(site)

def stop(self):
    self._stop = True

def run(self):
    while not self._stop:
        with self._sitesCond:
            ## Check if all potential sites are filled.
            if self._siteCount() == 0:
                ## Acquire a new carrier.
                self._convMan.shiftCarrier()
                ## Separate sites for the upload clients.
                self._partitionSites()
                ## Notify upload clients that the new carrier is
```

```
            ## ready for upload.
            self._sitesCond.notifyAll()
        time.sleep(1.0)
```

A.2.7 ConveyorManager Class

```
import threading
import time

from utils import iceMgr

import icehms
from hms import blenderpnf

class ConveyorManager(threading.Thread):
    """ Class for managing the trolleys on the conveyor by
     orchestrating the PnF-stops. The main purpose is to support the
     request for a new carrier at the upload PnF-stop, while ensuring
     that the upload buffer PnF-stop is replenished."""

    def __init__(self):
        threading.Thread.__init__(self, name='CarrierManager')
        self.daemon = True
        ## Get all PnF-stops.
        self._pnfStops = [
            blenderpnf.PnFStopPrx.checkedCast(x).ice_timeout(60000)
            for x in iceMgr.findHolons('::hms::blenderpnf::PnFStop')]
        ## Get the PnF-drive units. Should only be one.
        self._pnfDrives = [
            blenderpnf.PnFDrivePrx.checkedCast(x)
            for x in iceMgr.findHolons('::hms::blenderpnf::PnFDrive')]
        ## Setup named PnF-stops as object attributes.
        for s in self._pnfStops:
            self.__dict__['_%s'%s.getDeviceName()] = s
        ## The paint buffer stop should always let trolleys through,
        ## to recycle the carriers without transfer.
        self._PaintBuffer.unBlock()
        self._stop = False

    def shiftCarrier(self, wait=True):
        """ The method to shift the carriers, ensuring that a new
         carrier is posed at the upload PnF-stop and that the upload
         buffer PnF-stop is replenished with a carrier."""
        ## Free a current upload carrier.
        if self._Upload.hasBlockedTrolley():
            self._Upload.releaseTrolley(60.)
            ## Interruption point
            if self._stop:
                return False
        ## Make sure there is a trolley at the upload buffer.
        if not self._UploadBuffer.hasBlockedTrolley():
            ## Release a trolley from storage.
            self._CarrierStorage.releaseTrolley(60.)
            ## Interruption point
            if self._stop:
                return False
        ## Check that no trolley has arrived to upload. This may have
```

```
        ## happened, if the previously released trolley had another
        ## trolley blocked on it. If no trolley at upload, send one
        ## from the upload buffer.
        if not self._Upload.hasBlockedTrolley():
            ## Release a trolley from upload buffer.
            self._UploadBuffer.releaseTrolley(60.)
            ## Interruption point
            if self._stop:
                return False
            ## Now replenish the upload buffer with a carrier from storage.
            self._CarrierStorage.releaseTrolley(60.)
        ## Optionally wait until the new carrier has arrived at upload.
        if wait:
            ## Await a carrier at arriving at Upload.
            self._Upload.waitForBlockedTrolley(60.)
        return True

    def stop(self):
        self._stop = True

    def run(self):
        while not self._stop:
            time.sleep(1)
```