Johannes Schrimpf

Sensor-based Real-time Control of Industrial Robots

Thesis for the degree of Philosophiae Doctor

Trondheim, August 2013

Norwegian University of Science and Technology
Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology,
Mathematics and Electrical Engineering
Department of Engineering Cybernetics

ISBN 978-82-471-4568-5 (printed ver.)
ISBN 978-82-471-4569-2 (electronic ver.)
ISSN 1503-8181

ITK Report 2013-8-W

Doctoral theses at NTNU, 2013:225

Printed by NTNU-trykk

To my parents.

Summary

This PhD thesis presents topics related to sensor-based real-time robot control in applied automation.

While in the past industrial challenges often could be solved with traditional robot programming methods, the industry looks more and more towards the automation of complex tasks that need real-time sensor-feedback in the control loop of the robot. This is especially important in applications where the workpiece shape or characteristics are uncertain or unknown. When working on such workpieces, the movement has to be adjusted in real-time, ie. while the robot is moving. To achieve this, sensors are added in the control loop of the robot.

One challenge for the implementation of industrial solutions including real-time control is the lack of appropriate interfaces in most commercially available robots. When such interfaces are not present, in-house developed solutions can be used to access the low-level system of the robots, at least for research applications. These usually require modifications of the hardware, the software platform or both. Often, interfaces are implemented that allows for parts of the control loop being moved to an external PC. In this thesis, real-time interfaces for two different industrial manipulators are presented that are used for lab experiments and demonstrators. The real-time interfaces work position-based and have update frequencies of about 100 Hz. Ethernet UDP is used to communicate with the external controller platform. Experiments are presented that measure the delays in the low-level systems of the robots.

To externally control the movement of the real-time controlled robots from an application controller, a trajectory generator is needed. This thesis gives an overview of three different on-line trajectory generators that were in-house developed. The real-time capabilities of the presented trajectory generators were analyzed in order to ensure that the requirement for response times of the real-time interfaces are met.

Based on a presented real-time interface and trajectory generator, a

test platform was built. The test platform demonstrates tracking of a line that is sketched on the workpiece. The robot tool is controlled to keep a fixed distance between tool and workpiece, while maintaining an orientation perpendicular to the workpiece surface. The line tracking is done using a line-of-sight based control method. Experiments are presented, measuring the delays in the robot-sensor system.

The main part of this thesis is a presentation of an automated robotic sewing cell that demonstrates a case of sewing for the furniture industry. The system is able to sew together two parts with slightly different shapes. A two-robot solution is presented that controls the work pieces independently during the sewing operation. A force sensor is integrated in the control system to keep a constant sewing force. The seam allowance is controlled by an edge control system based on optical sensors that are mounted on the sewing machine. The real-time capabilities of the system are analyzed. Experiments are presented showing the feasibility of the presented control methods. The seam quality was evaluated by manual inspection and was found to be adequate. Further steps were identified that are necessary to include the demonstrator in an industrial setup, mainly corner matching, sewing of the last few centimeters and the material handling before and after the sewing operation.

Acknowledgements

I want to thank my supervisor Adjunct Associate Professor Geir Mathisen for close cooperation and guidance during the whole period of my PhD work. I also want to thank my co-supervisors Professor Kristin Y. Pettersen and Professor Terje K. Lien.

Furthermore, I want to thank all persons that made this work possible and helped me with advice, proofreading, discussions and comments, especially Kristjan Dempwolf and Line Holien. I want to thank my colleagues from SINTEF Raufoss Manufacturing and from NTNU ITK, first of all Morten Lind, Lars Erik Wetterwald and Magnus Bjerkeng, for close collaboration and for the good working environment. I also want to thank my PhD colleagues for interesting discussions, presentations and good cake. A special thanks goes to Magnus Bjerkeng and Christian Holden for spicing up the work day with intellectual training and discussions.

Many thanks to the administrative staff at the department, especially Tove K. Johnsen for help in administrative matters.

I want to thank my friends for their support and friendship. I also would like to thank my fellow Lindy Hoppers for keeping my mood up during my time as a PhD student.

Finally, and most important, I want to thank my family who always encouraged me and supported me in my decisions. Thank you for always being there for me.

The presented work is part of the SFI Norman research program (Centre for Research-based Innovation - Norwegian Manufacturing Future), established by the Research Council of Norway. I want to thank Ekornes ASA as well as the projects Automated 3D Sewing and Robust Industrial Sewing Automation for close cooperation.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

Term	Description
C	A programming language
C++	A programming language
CC BY-SA	A Creative Commons license
CPU	Central Processing Unit
API	Application Programming Interface
ATI Net F/T	A network force/torque sensor system from ATI
IP	Internet Protocol
KUKA RSI	Robot Sensor Interface for KUKA robots
KR60L30	A KUKA robot
OLS	Optical Line Sensor
OpenCV	Open Source Computer Vision, a library
Orocos	Open Robot Control Software
Orocos BFL	Bayesian Filtering Library of Orocos
Orocos KDL	The Kinematics and Dynamics Library of Orocos
Orocos PyKDL	The Kinematics and Dynamics Library of Orocos with Python bindings
Orocos RTT	The Real-time Toolkit of Orocos
PC	Personal Computer
PCI	Peripheral Component Interconnect
PyMoCo	A Python-based on-line trajectory generator
RAM	Random-access Memory
RCCL	The Robot Control C Library
ROS	Robot Operating System, a robot library
RT	Real-time
RX	A Stäubli robot type
SC15F	A Nachi robot

SSH	Secure Shell, a network protocol
TCP	Transmission Control Protocol, a network protocol
TX	A Stäubli robot type
UDP	User Datagram Protocol, a network protocol
UR5	A robot from Universal Robots

Chapter 1

Introduction

1.1 Motivation

The automation of complex industrial processes is an important topic in current robotics research. Especially in high-cost countries like Norway, a large effort is put into automation in order to keep the production in the country rather than outsourcing to low-cost countries.

Robots can be found everywhere in manufacturing companies. In 2012, the worldwide stock of operative robots was estimated to be about 1 400 000 units[rob]. The largest part of these installations solve assembly, painting, welding, or pick-and-place tasks that are pre-programmed and repeatable for mass production.

Challenges arise when tasks that are complex are to be automated and the robot system has to interact with an unknown environment or has to handle unpredictable behavior of the work pieces. In the recent years, there has been a new trend in robotics in the direction of real-time sensor integration, increasing the possibility to develop adaptive real-time controlled robot applications. In addition, the number and types of sensors are increasing while the cost is decreasing. At the same time, robot manufacturers are opening their control systems for research. This enables development of new real-time controlled applications.

1.2 Real-time Robot Control

The advance in the area of real-time robotics during the past decades has large benefits for the industry. This advance opens the possibility to automate complex industrial tasks that cannot be automated with traditional robot platforms due to the lack of possibilities for sensor integration in the

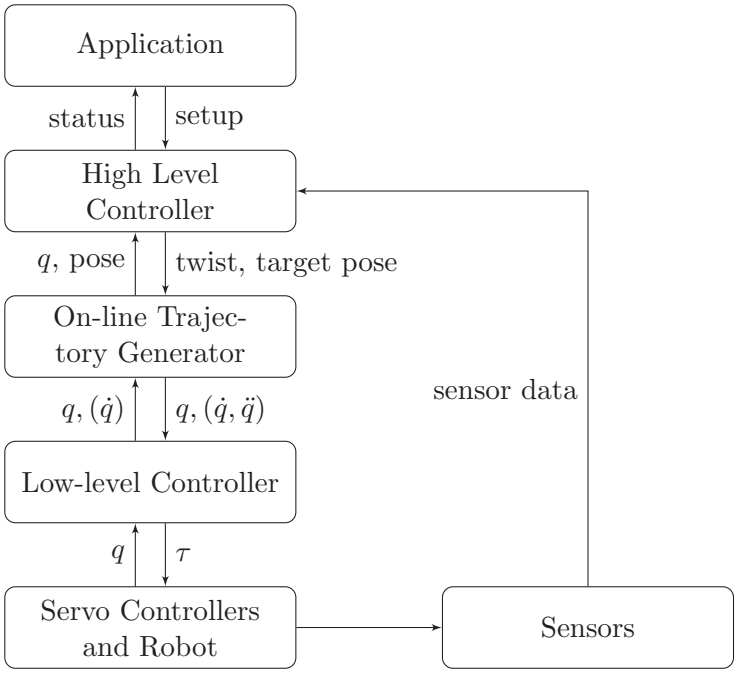


Figure 1.1: The model of a real-time robot controller. q is the joint position vector, τ the joint torques. Details of low-level controller interfaces are discussed in Chapter 2, implementations of trajectory generators in Chapter 3. Examples of applications using the whole control loop are presented in Chapter 4 and 5.

real-time control loop. By including sensors in the control loop, it is possible to build systems that interact with their surroundings while the robot is moving. Especially when working with workpieces with unknown shape or material characteristics, it is important to measure the current state of the system in order to generate robot motions that solve the specified task. This flexibility makes it also possible to use the robot system not only for operations that are pre-programmed, but also use the robot in small scale production lines where the products are changing rapidly or the products cannot be modeled satisfactory, for example due to non-rigid materials.

When controlling a robotic system, different control layers can be defined. The control problem is typically decomposed into three tasks: path planning, trajectory generation and trajectory tracking [SHV06]. Path planning is the problem to find a collision free path that connects an initial configuration with a final configuration of the robot. Trajectory generation is the process of computing a time dependent function of the desired path

which specifies the motion of the robot. The trajectory tracking is the task to control the robot to move on the desired trajectory.

When controlling a robot in real-time and including sensor input into the control loop, the trajectory cannot be specified in advance. Figure 1.1 shows the author's view on a robot system in real-time control mode. On the top layer an application sets up a high-level controller that gets input values from an external sensor system and calculates movement commands for the robot. These commands are communicated to the next layer, consisting of the trajectory generator that also provides the high-level controller with status updates from the robot such as the robot tool pose or the joint positions. The trajectory generator periodically receives the actual joint positions and calculates new joint position set points. These are sent to the low-level controller that calculates the torque set points that are sent to the servo controllers.

In traditional robot controllers, all the control layers are included in the controller cabinet and a high-level language is used to program the robot. The possibility to include sensor inputs in the real-time control loop of commercially available robots is often very limited, if present at all. Furthermore, there is usually no interface to the lower control layers. On the other hand, sensor-based robot control has become a large field in the research community in the past decades and there have been developed a large number of in-house solutions for accessing the low-level controllers of commercially available robots. But also the industry begins to work with the automation of complex processes that require real-time control of the robots which leads to the robot manufacturers slowly opening up the access to the low-level controllers by providing real-time interfaces.

Two types of interfaces to the low-level system can be found for commercially available robots that offer possibility for real-time control: joint-position-based interfaces and torque-based interfaces:

Joint-position-based interfaces allow the control of the robot by commanding joint positions to the low-level controller of the robot. Some interfaces also require velocity and acceleration set points. They usually have a cycle frequency of about 100 Hz and the low-level controller in the robot is responsible for calculating the motor torques. An external trajectory generator has to be used to calculate the trajectory of the robot.

Torque-based interfaces have a higher update frequency, usually above 1000 Hz. These interfaces require an external motion controller to calculate the torques from the desired motion or sensor input. The

torques are then sent to the low-level system of the robot. Torque-based interfaces are mainly used for research activities.

The presented PhD work focuses on the use of joint-position-based interfaces for industrial robots together with external trajectory generation in order to work on the automation of complex industrial applications. A special focus is on sensor-integration in the real-time control loop of industrial robots. The presented work suggests solutions for challenges in the automated sewing of two non-rigid parts to subassemblies that are to be used in the furniture industry.

1.3 Contributions and Scope of this Thesis

The presented work can be classified as applied research, driven by demands from the industry to integrate real-time controlled robots in industrial applications. Therefore the focus is on experimentation and integration of methods that are well known in research into industrial demonstrator cases rather than generation of theory.

This thesis covers topics in different levels of real-time control of industrial robots, with the following core areas:

- Real-time interfaces for industrial robots
- On-line trajectory generation
- Line tracking with tool orientation control
- Automated sewing

The work in this thesis started with the development of real-time interfaces for industrial manipulators in order to build a platform for experimentation with real-time robot control in industrial cases. After the initial development, the focus moved away from pure low-level development towards applications utilizing the real-time interfaces for sensor-based robot control. Working areas included development of on-line trajectory generators, implementation of demonstrators and experiment setups. A special emphasis was on flexible and reconfigurable implementations that are based on widely available hardware and preferable on open-source software. Another design choice was on using software platforms that are well suited for fast prototyping. An important part was the analysis of the real-time capabilities of the systems built, especially in respect to the decision to use flexible systems rather than customized systems that would result in a

better performance. A test platform was built, demonstrating an application where the robot follows a line drawn on a workpiece while keeping the tool perpendicularly orientated to the workpiece surface in a fixed distance. This case was chosen to demonstrate the system's ability to interact with the surroundings based on sensor input, inspired by painting and welding applications.

When the experiments showed satisfactory results, the way was paved for using the developed systems in real industrial applications. Among various challenges presented by the different project partners, the topic of automated sewing was chosen to demonstrate the benefits of real-time robot control in manufacturing.Recent research projects emphasized the need of sensor-based real-time control to solve the challenge of sewing automation. Even though the sewing application is quite specific, the applied methods are kept as general as possible and have a significant value to other projects that benefit of real-time robot control.

While many parts of the whole sensor-based robot systems were discussed, the following list gives an overview of topics that are not discussed in the presented work due to the limited time and more specific focus on other parts:

- Torque-based joint control (not accessible in the used robots)
- Mathematical details in the trajectory generators (handled by the used libraries)
- Optimal control solutions and control parameter adjustment (due to limited time and due to desired simplicity and flexibility)

A more detailed overview of the contributions is presented in the following subsections.

1.3.1 Real-time Interfaces to Industrial Robots

An overview is given of the robot platforms available in the laboratory that were used in the PhD project. The presented robots can be controlled in real-time either with a native interface or using software and/or hardware manipulations on the robot platform. One emphasis of the presented work is on real-time interfaces that allow joint position control of the robot.

A special focus is on real-time control of the *Nachi SC15F* and the *Univeral Robots UR5* robot since the used interfaces were enhanced (SC15F) or developed (UR5), and tested as a part of this work. These interfaces are

evaluated and the behavior is discussed with respect to available interfaces for other robots.

The author's contributions are:

- The real-time interface for the Nachi SC15F was developed by the author before starting the PhD project. It was enhanced and tested as part of the presented work.
- The real-time interface for the Universal Robots UR5 was designed, implemented and tested in close cooperation between the author and Morten Lind as part of the presented work.

1.3.2 On-line Trajectory Generation

To achieve sensor-based robot control using a joint-position-based real-time interface, a trajectory generator has to be used that can generate a robot motion based on sensor input. Different trajectory generators are presented that are either based on in-house developed software or already existing frameworks. The behavior and performance of the different implementations is evaluated.

A special emphasis is on trajectory generators and control applications developed in Python. Python is a flexible high-level language with a large number of available libraries and with the possibility to include C code and C libraries for computation intensive parts of the programs. Examples for supported libraries and framework related to robotics are ROS[1], OpenCV[2] and Orocos[3]. Python on Ubuntu was chosen as main programming platform, rather than a hard real-time language, to build a flexible system that allows for rapid prototyping. Throughout the different papers the matter of delays in the system are discussed, also with respect to the choices of software and hardware.

The author's contributions are:

- The author was involved in design, testing and enhancement of the presented trajectory generator PyMoCo.
- The author single-handedly designed, implemented and tested the presented trajectory generator based on Orocos PyKDL.
- The author single-handedly designed, implemented and tested the presented trajectory generator based on Orocos KDL.

[1] `http://www.ros.org`
[2] `http://opencv.willowgarage.com`
[3] `http://www.orocos.org`

- The author designed and conducted experiments regarding the real-time capabilities of the trajectory generators in close cooperation with Morten Lind.

1.3.3 Line Tracking with Tool Orientation Control

An experimentation platform was analyzed that demonstrates line-following with tool orientation control as used in applications like painting, welding or sewing. The robot tool follows a line sketched on the workpiece in a fixed distance while maintaining an orientation perpendicular to the workpiece surface. The platform was designed by the author prior to the PhD work. The author's contributions during the PhD work were:

- Mathematical formulation of the control algorithms.
- Detailed real-time analysis of the control system.
- Modelling and simulation of the case of approaching a straight line.
- Experiments were designed and conducted in close cooperation with Morten Lind.

1.3.4 Automated Sewing

This thesis includes work on an automated sewing cell as application for real-time robot control. The research setting is a sewing cell installation consisting of an industrial sewing machine and two robots controlling the sewing operation in real-time. Widely available software and hardware technologies were used to build a highly flexible system for operational experimentation and control prototyping. The software architecture and the communication structure are presented and different control mechanisms and methods are discussed.

Included topics are:

- Strategies to control the sewing operation of non-rigid materials
- Strategies for position control and force control of two different parts
- Synchronization between the two robots
- Real-time analysis of the control system

Experiments are included that show the feasibility of the proposed platform and control mechanisms.

The authors contributions are:

- Design of the different control methods for the sewing operation.
- Design and implementation of the overall system in close cooperations with colleagues.
- Experiments and analysis regarding the control strategies were conducted in close cooperation with colleagues.
- Experiments and analysis regarding the delays in the system.

1.4 Overview of the Included Publications

This section presents a short overview of the included publications.

1.4.1 Open Real-Time Robot Controller Framework

M. Lind, J. Schrimpf, and T. Ulleberg, *Open real-time robot controller framework*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010

This paper describes topics in external real-time control of industrial robots. Experiments are presented, analyzing the behavior of three different real-time controlled robots with respect to delays in the low-level system.

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

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*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010

This paper describes a demonstrator setup for line-of-sight path tracking with tool orientation control. In the demonstrator setup the robot tool has to follow a line on the workpiece while keeping the tool orientated perpendicularly to the workpiece distance in a constant distance.

1.4.3 Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking

J. Schrimpf, M. Lind, and G. Mathisen, *Time-analysis of a real-time sensor-servoing system using line-of-sight path tracking*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 2861 –2866

This paper contains a time analysis for the line-of-sight path tracking demonstrator. The different delays in the system are described. A model of the system is presented for the case of approaching a straight line. The influence of the line-of-sight parameters is discussed. Experiments are presented comparing the estimated delay with experimental data.

1.4.4 Experiments towards Automated Sewing with a Multi-Robot System

J. Schrimpf and L. E. Wetterwald, *Experiments towards automated sewing with a multi-robot system*, International Conference on Robotics and Automation (ICRA), 2012

This paper describes a concept for an automated sewing cell. The sewing cell consists of a sewing machine, two robots, and sensors for force and edge measurements. Different control strategies are described and experiments are presented to examine the feasibility of the different control methods.

1.4.5 Implementation Details of External Trajectory Generation for Industrial Robots

J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, *Implementation details of external trajectory generation for industrial robots*, Proceedings of IWAMA 2012 - The Second International Workshop of Advanced Manufacturing and Automation, 2012

This paper describes implementation details for three different on-line trajectory generators. Experiments are presented that analyze the response times of the different implementations. General topics in robot control and trajectory generation are discussed with a focus of real-time trajectory generation using the Python programming language.

1.4.6 Real-Time System Integration in a Multi-Robot Sewing Cell

J. Schrimpf, L. E. Wetterwald, and M. Lind, *Real-time system integration in a multi-robot sewing cell*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012

This paper describes the implementation of an automated sewing demonstrator. The used control strategies for force and edge control are presented. Experiments are presented showing the performance of the proposed control strategies.

1.4.7 Python-Based Robot Motion Control

M. Lind, L. Tingelstad, and J. Schrimpf, *Real-time robot trajectory generation with python*, IROS2012 Workshop on Robot Motion Planning: Online, Reactive, and in Real-time, 2012

This paper describes the design of PyMoCo, a framework for external on-line trajectory generation. Performance measurements are presented for different canonical trajectory generators. General topics of using Python on a mainstream Linux platform are discussed with respect to the performance of the presented framework.

1.4.8 Real-Time Analysis of a Multi-Robot Sewing Cell

J. Schrimpf, M. Lind, and G. Mathisen, *Real-time analysis of a multi-robot sewing cell*, IEEE International Conference on Industrial Technology (ICIT), 2013

This paper presents a real-time analysis of an automated sewing cell. The communication and computation delays in the different parts of the system are measured and discussed. The real-time capability of the used software is discussed on the basis of the measurements.

1.5 PhD Thesis Outline

This thesis is designed as a collection of papers with a summarizing introduction.

Chapter 2 presents real-time interfaces that were used or developed during the presented work, including experiments measuring the delays in the robot system.

Chapter 3 gives an overview of the trajectory generators that were developed or used. This chapter also includes timing experiments to determine computation and communication delays in the presented implementations.

Chapter 4 presents a demonstrator system for sensor-based tracking of a visual line using a real-time controlled robot. The delays in the control system are estimated by analyzing the different system parts. Experiments are presented to compare the estimated delay with the delay in the real system.

Chapter 5 summarizes the work that was done in the field of automated sewing. Based on an industrial case, a sewing cell installation was designed and experiments were conducted, both concerning the seam quality and the real-time characteristics of the sewing cell and the control loops.

Chapter 6 contains the attached publications produced during the PhD work, including the bibliographic information for the contained papers and short declarations of contributions.

Chapters 2 to 5 include a motivation, related work and a summary of presented designs, implementations and experiments. These chapters also include references to the attached publications. When referring to attached publications, the complete title will be used as notation. Each chapter includes an introduction and an overview of related work in the specific work area as well as a conclusion of the presented part.

Chapter 2

Real-time Interfaces to Industrial Robots

This chapter describes real-time interfaces that were used or developed during the presented work. These interfaces are used throughout the work that is presented in the attached publications. The experiments at the end of the chapter were presented in *"Open Real-Time Robot Controller Framework"* [LSU10].

2.1 Motivation

Sensor-based robot control plays an important role in the automation of complex industrial processes. These processes often require the use of sensors in the control loop of the robot in order to react in real-time to unforeseen events or to work with unpredictable materials. Most industrial robots have a very limited access for real-time control in the high-level robot controller. Some robot manufacturers offer special sensor packages to include for example force sensors into the control loop or to alter the trajectory while moving the robot based on sensor input. A more flexible alternative is to use a direct interface to the low-level controller of the robot and thereby replace the high-level controller and typically also the trajectory generator with a custom implementation of these components. These interfaces often use Ethernet to communicate the actual joint states to the external controller and to receive joint update data. Only few robot manufacturers offer such interfaces. If the applied robot lacks this feature, modifications have to be made to the robot controller in order to allow for real-time low-level control of the joints.

2.2 Related Work

A comprehensive overview of commercially available real-time interfaces for industrial robots is presented in [KW10].

In Kubus et al. [KSK+10], two different real-time control architectures for real-time control of Stäubli manipulators are presented. One architecture is developed for RX manipulators. Modifications are made on the control hardware and the controller board is replaced by a standard PC running QNX Neutrino. More PCs are added for sensor data processing execution of manipulation primitives, and user interface software. Joint position control is done with a frequency of more than 10 kHz. Internal communication is handled via the QNX QNet protocol. The other presented architecture is built to control TX manipulators. A low-level interface can be accessed using a C library. This library allows access to the control loop at rates up to 1 kHz, but is limited by the used hardware to a rate of 250 Hz. The low-level system is connected to the external controller via an Ethernet TCP/IP connection to exchange set points and feedbacks. It is noted that TCP/IP is not real-time capable, especially in bigger networks, but in the point-to-point connection between the two controllers proves to meet the real-time demands.

Another modification of an industrial robot manipulator is presented by Blomdell et al. [BBB+05]. The design and implementation of a real-time interface to the ABB S4CPlus control system is described. The implementation uses shared memory via the PCI bus. Three different sampling times in the controller platform are described. The high level language ABB RAPID has a sampling time of 0.1 s, the interface to the arm servo control 4 ms and the internal sampling time for the JR3 force/torque sensor 0.125 ms. Experiments were included that demonstrate force control via the interface. It is concluded that the 4 ms sampling time is a good trade-off for many force control applications in respect to the needed computational power. It is also stated that applications using force control in an extremely stiff environment, higher update frequencies may be required.

2.3 NACHI SC15F

This section presents an overview of an in-house developed real-time interface to a NACHI SC15F robot. The interface is used in several setups, for example the test platform for sensor-based robot control that is presented in Chapter 4. Figure 2.1 shows the robot during an experiment with a compliance controller using the sensor input from a force sensor mounted on the

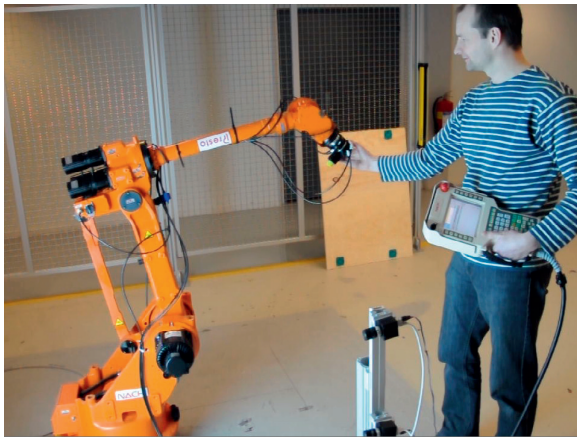


Figure 2.1: The NACHI SC15F in compliance control mode.

tool flange.

The SC15F is a 6-axis industrial manipulator. It ships with a NACHI AX controller box that does not natively offer an interface for external real-time control of the robot. The controller internally consists of the main controller and a servo controller board that does the dynamics calculations.

A low-level interface system has been built that intercepts the connection between the main controller and the servo controller. This interface offers observation and manipulation of the joint angles in the low-level system. By forwarding the current joint states to an external computer system and returning correction values, it is possible to control the joint angles from external applications. The communication runs with a cycle time of 10 ms. This cycle time is comparable to commercial robots with low-level interfaces as for example KUKA robots with RSI (Robot Sensor Interface).

Figure 2.2 shows the concept of the real-time interface. It is connected to the external controller PC via Ethernet/UDP. The commands from the external PC are stored in the interface system until a new command is sent. This makes the real-time interface robust for packet loss or other interruption of the connection and ensures that the communication cycles in the original connection are kept and are not depending on the immediate answer from the external controller.

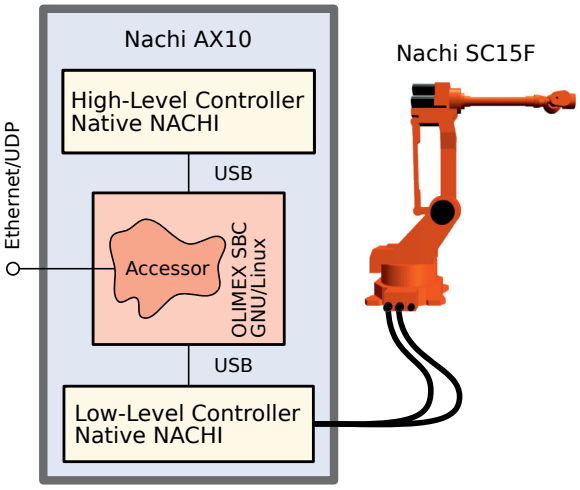


Figure 2.2: The NACHI robot with controller. License: CC BY-SA Morten Lind.

2.4 Universal Robots UR5

A other research activity in the lab is gathered around a Universal Robots UR5[1]. The UR5 is a relatively inexpensive 6-axis industrial manipulator with a reach of 85 cm and a maximal payload of 5 kg. This section presents an overview of different control methods and a more detailed description of an in-house developed real-time communication interface to the robot based on a C-API provided by the robot manufacturer. This API offers access to the low-level controller of the robot.

2.4.1 Control Methods

The UR5 includes a controller platform with a teach pendant that allows the costumer to program the robot using a graphical user interface. Alternatively, it is possible to write programs in a scripting language and either save the programs on the robot controller or send commands via a TCP interface to the robot. These programs are processed in the native high-level controller.

A more experimental feature for developers is the possibility to run user-written C-programs that interact with the controller with a cycle time of 8 ms. The UR5 controller is based on a Debian GNU/Linux system, which can be accessed by the user either via SSH (Secure Shell) or directly

[1] former UR-6-85-5-A

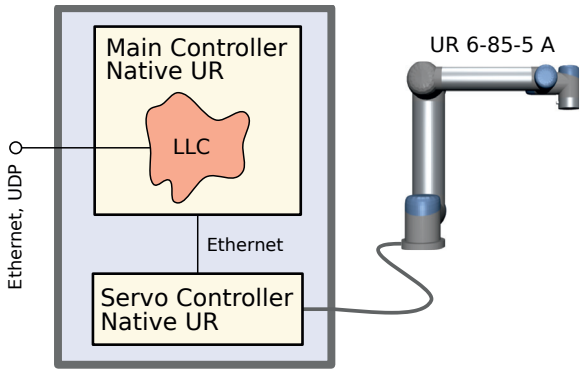


Figure 2.3: The Universal Robot with controller. The low-level controller (LLC) is a user space program which uses an API to access the servo controller. License: CC BY-SA Morten Lind.

by attaching a keyboard to the controller. The native high-level controller program can be stopped by a script and the user-written controller program can be started to access the low-level functions of the robot. This is done by programming a custom application and linking it to a C library file, distributed by the robot manufacturer on request.

2.4.2 Router Implementation

A router program was implemented that allows an external control PC to control the robot via Ethernet, see Figure 2.3.

The program runs on the UR5 controller PC and uses the C-API to access functions in the low-level controller. The program sends the actual joint values received through the API to the external control PC and waits for a respond packet containing joint updates. The Universal Robots servo controller can be controlled by either communicating joint velocities or a combination of joint position, joint velocity and joint acceleration. Since most trajectory generators give a trajectory consisting of position set points, an extrapolator was included into the router program in order to calculate the expected joint velocities and accelerations when only joint positions are given. The interface runs with an update frequency of 125 Hz which result in a cycle time of 8 ms. When in real-time control mode, the low-level API expects new joint updates within a time window of 4 ms. Exceeding this limit results in neglecting the new data or in a disconnect, depending on an option in the implementation.

2.5 Timing Experiments

High update frequencies of the real-time interfaces are a crucial requirement for a robotic system which has to react in real-time to sensor-inputs. On the other side this does not guarantee that the robot's reaction time and the delay when following a trajectory are in the same order of magnitude. Due to the proprietary nature of most industrial robots, the low-level controller has to be treated as a black box and may contain unknown filters, controller parameters and communication delays. Also the mechanical characteristics and motor parameters of industrial manipulators are often unknown.

This section presents a summary of experiments using the presented real-time interfaces in order to examine the response characteristics of the low-level interfaces as presented in the included publication *"Open Real-Time Robot Controller Framework"* [LSU10]. To do this, two different input signals were commanded to the robots, a step and a cosine trajectory. The step response was used to measure the delay from sending a command to the robot until a significant movement can be seen in the output values of the robot. The cosine trajectory was chosen to examine the tracking delay of the robot. The step size for the step response was experimentally adapted to the tolerances of the low-level interface regarding maximum step sizes. It is important to mention that no tweaking or optimization in the low-level controller was performed and that different control parameters may lead to different results.

The response time is defined as the time between a command is sent until half a step before a significant response is visible in the actual robot position. The tracking delay is defined as the delay between the commanded and the actual cosine trajectory at the steepest point.

The robots that were used in the experiments are a Nachi SC15F, a Universal Robots UR5 and a KUKA KR60L30 HA with RSI. The KUKA robot was included in the experiment in order to compare the responses to a robot that offers a native interface to the low-level controller.

Figure 2.4 shows the plots for the response of the Nachi SC15F robot. From the step response it can be seen that the response time of the SC15F robot is about 45 ms, while the tracking delay is around 120 ms. Figure 2.5 shows the corresponding plots for the Universal Robots UR5. Here, the response time is observed to be around 12 ms and the tracking delay is about 9 ms. The results for the KUKA KR60L30 are shown in Figure 2.6. The response time was recorded to be about 42 ms, whereas the tracking delay was around 115 ms.

It has to be mentioned that the results for the different robots are not

completely comparable due to different real-time interfaces, specifications, and robot size. While the SC15F and the KR60L30 real-time interfaces are joint-encoder-based, the UR5 interface takes commanded velocity and acceleration set points into account. Another difference may be in the internal processing and filtering of data in order to be tolerant to trajectories that are not directly feasible for the robot. These implementation details are not known due to the proprietary nature of the robot controllers.

2.6 Conclusions

Two implementations for position-based real-time interfaces for commercially available industrial robots have been presented. In the case of the Nachi SC15F manipulator, the controller hardware was modified in order to gain access to the low-level controller. The update frequency of the interface is 100 Hz. In the case of the Universal Robots UR5, a router application was installed on the native robot controller hardware in order to gain access to the low-level functionality of the robot from an external PC. The application was programmed in C using an API to access the low-level controller. The update frequency is 125 Hz. The interfaces can be accessed using Ethernet UDP. In both cases the low-level controllers are proprietary and the control parameters as well as the internal data flow are not known.

To gain a better understanding of the low-level systems of the two robots, the movement responses to a step input and to a cosine input were recorded. The response time was defined as the time between sending a command to the robot until a movement is recorded. The tracking delay was defined as the delay between the commanded and the actual cosine trajectory at the steepest point. This was also done for a KUKA KR60L30 robot using the RSI interface. The responses of the SC15F and the KR60L30 robot were quite similar, about 40 ms for the response time and about 115 ms for the tracking delay. The response time for the UR5 was observed to be 12 ms, whereas the tracking delay was 9 ms. The large discrepancy can be explained by different low-level controllers, specifications, robot size and filtering. Furthermore, the interface of the UR5 has an interface that expects position, velocity and acceleration set points, while the two other interfaces only expect joint position set points.

It is important to notice that the experiments are not meant to compare the performance of the robots, but rather to give an understanding of the real-time abilities of the low-level controller in order to be able to carry out timing analyses of control systems that include the examined robots.

Both presented low-level interfaces are frequently used in research projects

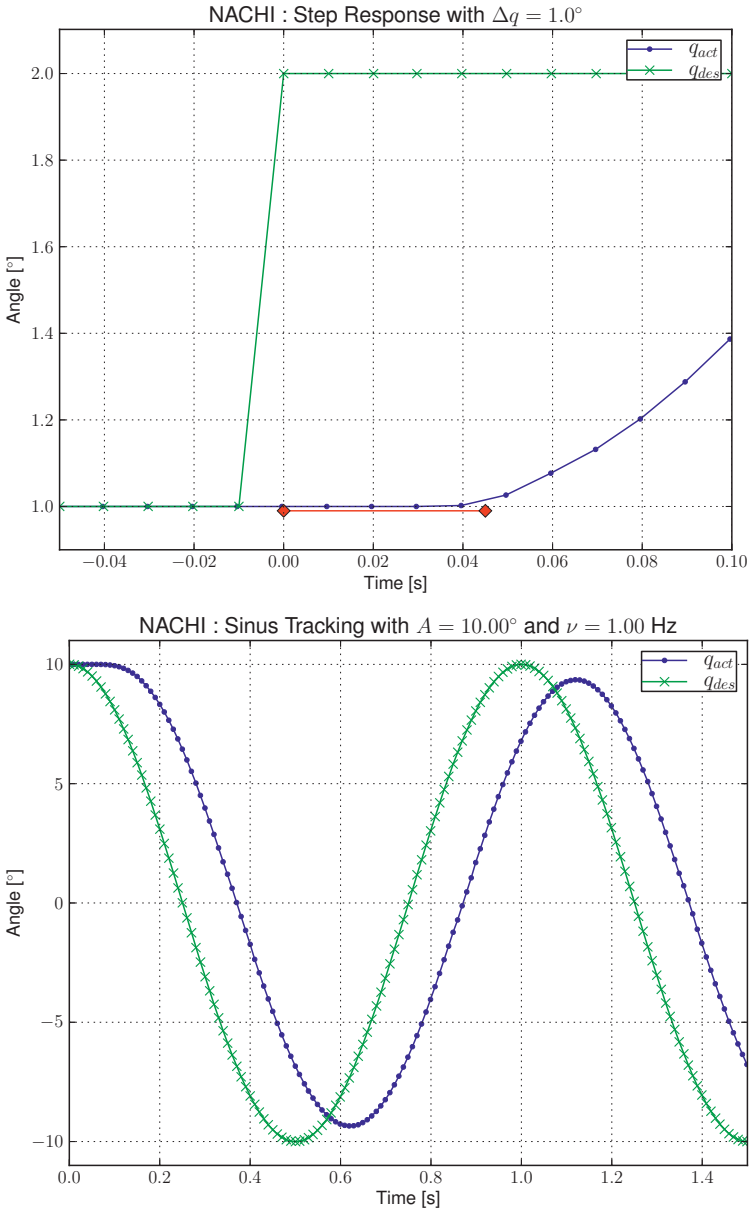


Figure 2.4: Step and cosine response for the Nachi SC15F, based on [LSU10]. Δq is the size of the step, A the amplitude of the cosine, and ν the frequency of the cosine. The response time is measured in the step response from the position change at time 0.00 s *until half a step before the first significant movement, shown as the red mark. The tracking delay is measured at the zero-crossing of the cosine curves.*

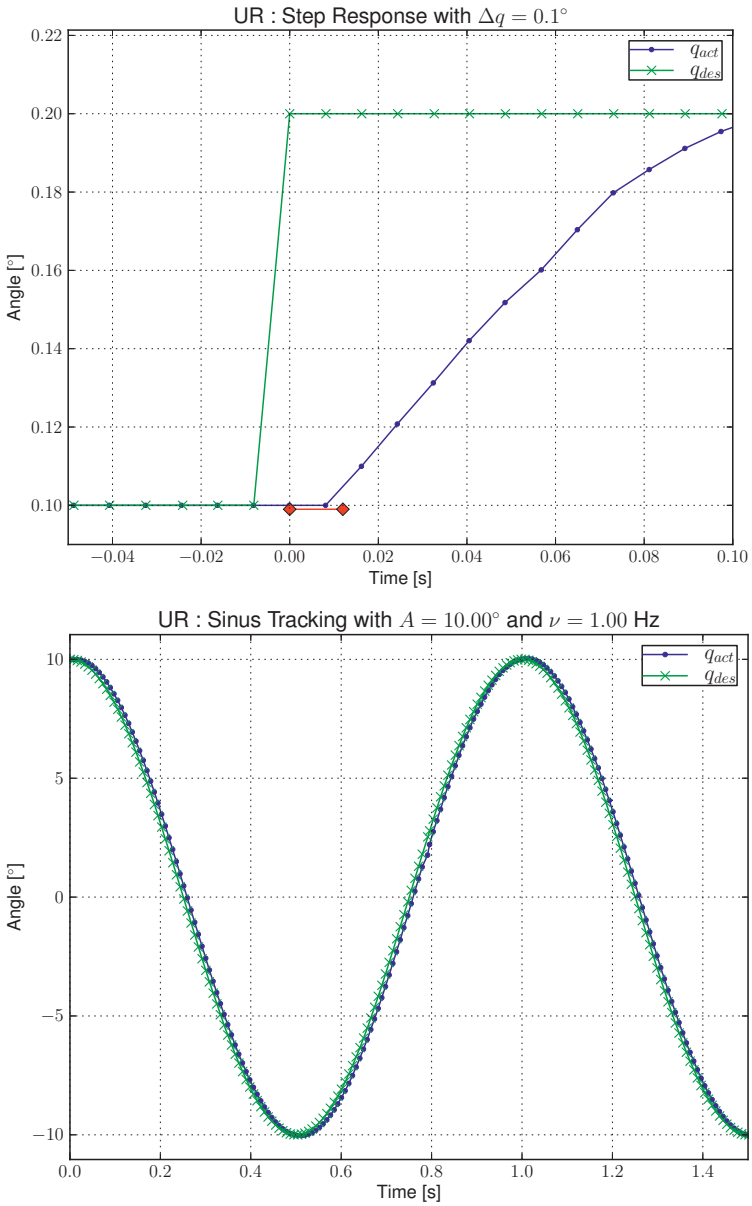


Figure 2.5: Step and cosine response for the Universal Robots UR5, based on [LSU10]. Δq is the size of the step, A the amplitude of the cosine, and ν the frequency of the cosine. The response time is measured in the step response from the position change at time $0.00\,\mathrm{s}$ until half a step before the first significant movement, shown as the red mark. The tracking delay is measured at the zero-crossing of the cosine curves.

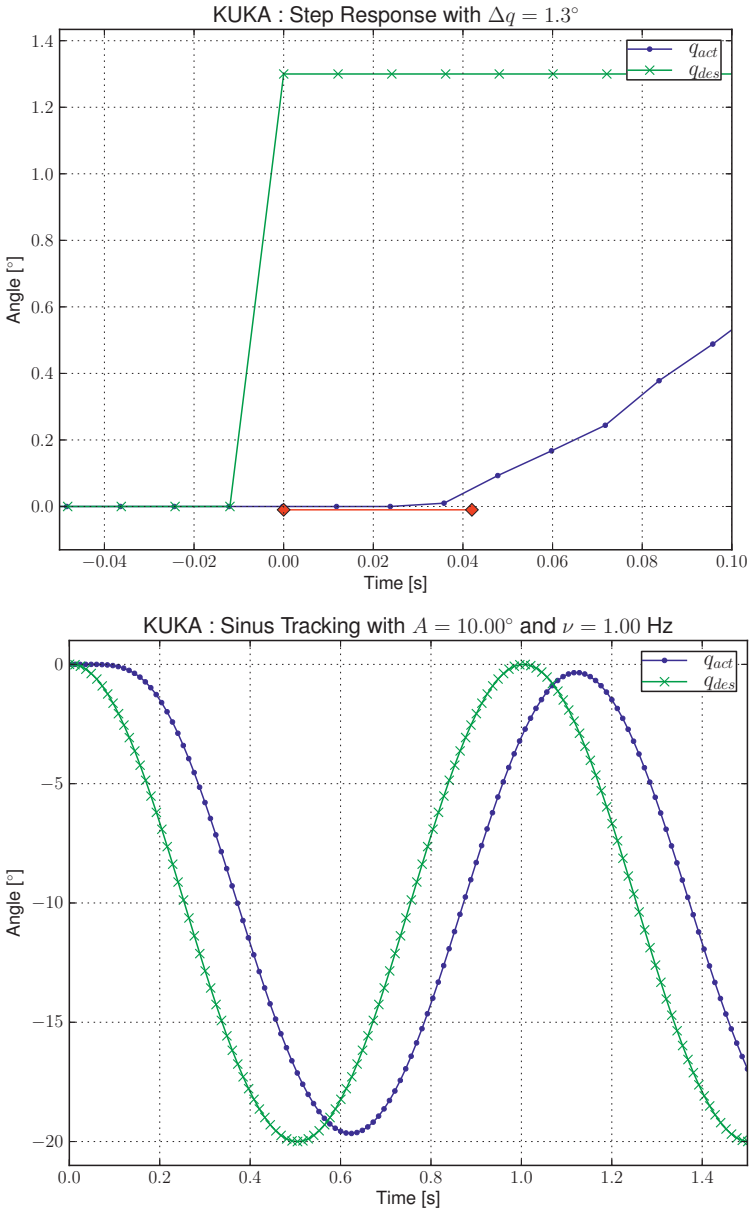


Figure 2.6: Step and cosine response for the KUKA KR60L30, based on [LSU10]. Δq is the size of the step, A the amplitude of the cosine, and ν the frequency of the cosine. The response time is measured in the step response from the position change at time 0.00 s *until half a step before the first significant movement, shown as the red mark. The tracking delay is measured at the zero-crossing of the cosine curves.*

and demonstrator setups at the laboratory, which demonstrated the usability of the interfaces.

Chapter 3

On-line Trajectory Generation

This chapter gives an overview of on-line trajectory generation of industrial robots. It will start with a short description of the topic followed by descriptions and an evaluation of different in-house programmed trajectory generator software. The work is presented in the included publications *"Open Real-Time Robot Controller Framework"* [LSU10], *"Implementation Details of External Trajectory Generation for Industrial Robots"* [SLSM12] and *"Real-Time Robot Trajectory Generation with Python"* [LTS12].

3.1 Motivation

As described in Chapter 1.1, a robot controller consists of different control layers. One layer is the trajectory generator. A trajectory is a time-dependent function $\mathbf{q}(t)$ that specifies the motion of the robot, where $\mathbf{q}$ contains the joint positions of the robot. Kröger [Krö10] distinguishes between the following two concepts:

Off-line trajectory generation is when the trajectory is calculated in advance and cannot be influenced during its execution.

On-line trajectory generation is when the motion of the robot can be (re-)calculated and/or adapted while the robot is moving.

When using on-line trajectory generation, the new state for all joints is calculated in each control cycle. This is typically done with an update frequency in the order of 100 Hz for position-based low-level interfaces. These interfaces typically have joint angles as input, often combined with joint

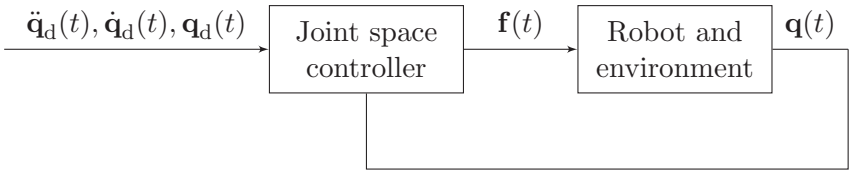


Figure 3.1: Scheme for trajectory-following control, common in industrial robots, as presented in Kröger [Krö10]. The desired joint states $\ddot{\mathbf{q}}_d(t), \dot{\mathbf{q}}_d(t), \mathbf{q}_d(t)$ are sent to the joint controller, which minimizes the error between the desired position $\mathbf{q}_d(t)$ and the actual position $\mathbf{q}(t)$. The output $\mathbf{f}(t)$ of the joint controller represents the forces/torques commanded to the robot joints.

velocities and accelerations. Figure 3.1 shows the scheme for a trajectory-following controller as often found in industrial robots.

Not all real-time interfaces for industrial robots have this kind of interface. Some robots, especially those that are used in research, offer interfaces that have higher update frequencies and other control methods. These usually work on torque control of the joint motors. The control loop has an update frequency of about 1 kHz or more. When using on-line trajectory generators with these interfaces, an additional layer in the external controller platform is needed to calculate the torques from the desired joint states.

In applications that use sensor input to control the robot trajectory, the next desired joint state is calculated on-line. This means that a new joint state is calculated in each interpolation cycle. Different strategies can be used to generate the robot movement. Often, a natural way is to command velocities in tool space based on a position or velocity error, but also movement towards a target pose that can be updated in real-time is common. To achieve this, most trajectory generator frameworks offer a variation of different canonical trajectory generators.

3.2 Related Work

Since this chapter focuses on implementations, this section mainly presents frameworks and libraries that can be used to implement on-line trajectory generators. Examples for the use of such on-line trajectory generation frameworks in robot applications can be found in countless publications. Most of these publications focus mainly on the robot applications and not the underlying principles for trajectory generation. While there are many

publications on traditional off-line trajectory generation, only a few publications focus on the concept of on-line trajectory generation. A comprehensive overview of these publications and the concept of on-line trajectory generation is presented in Kröger [Krö10].

Lloyd and Hayward [LH93] presented a framework called RCCL (Robot Control C Library) for programming of real-time robot controllers. It is a package of C routines for UNIX environments. Primitives are defined for target points in joint or Cartesian coordinates. The target points can be modified on-line and the motion can be canceled in response to sensor or control inputs. Smooth paths are provided which can be adjusted in real-time. Details are presented for path blending.

Orocos[1] (Open Robot Control Software), presented in Bruyninckx [Bru01], is a general-purpose and open robot control software package. It consists of C++ libraries for advanced machine and robot control. The Orocos libraries can be used in different levels of the control structure of the robot, for example the KDL (Kinematics and Dynamics Library) component can be used for kinematics and dynamics calculations based on kinematic chains and kinematic solvers. The RTT (Orocos Real-Time Toolkit) allows for real-time integration and communication, while the BFL (Bayesian Filtering Library) provides algorithms for data filtering and processing.

A commercially available library for trajectory generation is the Reflexxes[2] library, presented in Kröger [Krö11]. It focuses on high performance and provides smooth trajectories with response times below 1 ms. Based on the current state of motion, the target state of motion and motion constraints, a new state of motion and trajectory is generated. Due to the low response time the library is able to react to unforeseen events. It switches smoothly between different trajectory generation methods.

3.3 Implementations

This section presents a short overview of different implementations of trajectory generators that have been used during the PhD work. All trajectory generators were developed in-house, based on open source software. All presented trajectory generators are programmed to use joint-position-based low-level interfaces for industrial robots with an update frequency of about 100 Hz. The author contributed to the development of the presented PyMoCo framework by intensive discussions and testing. The two Orocos-based trajectory generators were designed and programmed single handedly

[1] `http://www.orocos.org`
[2] `http://www.reflexxes.com`

by the author as part of this PhD work.

3.3.1 PyMoCo

PyMoCo[3] is an in-house developed on-line trajectory generation framework entirely programmed in Python. The kinematics calculations are implemented using the python-math3d library[4] and the NumPy package[5]. The objective was to develop a flexible and general framework with functionality to include own trajectory generators and robot interfaces. It includes several canonical trajectory generators, for example a tool velocity trajectory generator and trajectory generator to move the tool linearly towards a target pose. Details are presented in the attached publications *"Open Real-Time Robot Controller Framework"* [LSU10] and *"Real-Time Robot Trajectory Generation with Python"* [LTS12].

3.3.2 Orocos-PyKDL-based Trajectory Generator

While PyMoCo was developed as a flexible and general framework, an alternative implementation was programmed that focuses more on performance than PyMoCo. It was implemented in Python and uses the Orocos KDL library[6] with the PyKDL Python wrapper for kinematics calculations. By doing this, the computation-intensive parts are computed in C instead of Python. Interpolation in tool space and 3D mathematics are implemented using the python-math3d library. Even though it can be adapted to be used for different robot models, the focus was on control of the Universal Robots UR5. It includes trajectory generators to move the tool linearly in tool space and in joint space, as well as a tool velocity trajectory generator and a joint velocity trajectory generator.

3.3.3 Orocos-C++-based Trajectory Generator

In order to compare the Python-based implementations with a pure C++ implementation, a minimal implementation of a trajectory generator was programmed in C++ using the Orocos KDL library for both kinematics calculations and trajectory interpolation. It includes trajectory generators to move the tool linearly to a target position, as well as a tool-velocity trajectory generator.

[3] `https://launchpad.net/pymoco`
[4] `https://launchpad.net/pymath3d`
[5] `http://www.numpy.org`
[6] `http://www.orocos.org/kdl`

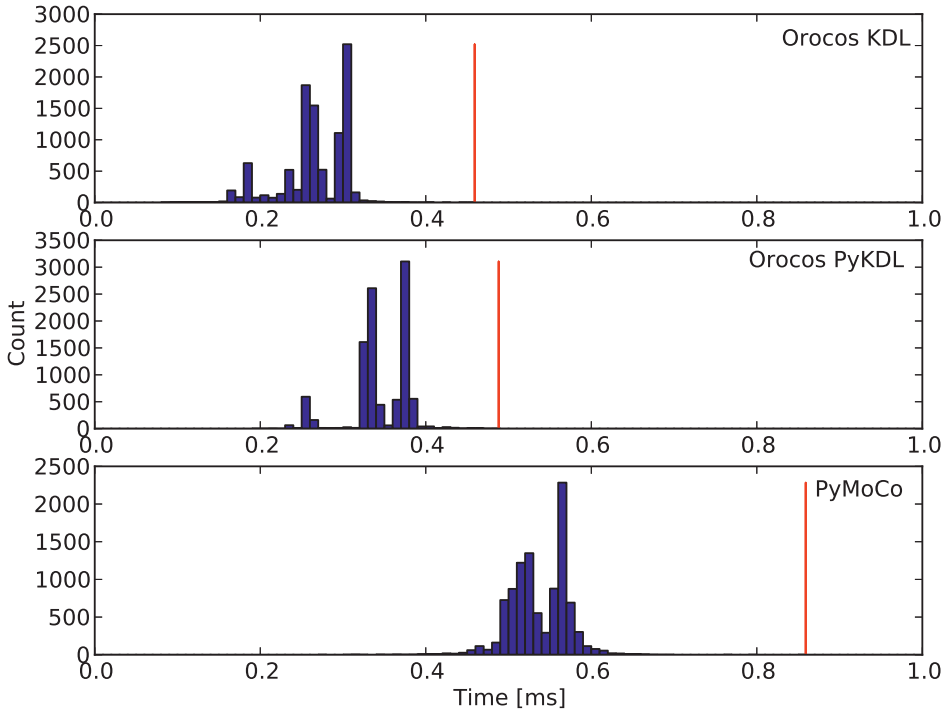


Figure 3.2: Response times for the different implementations for 10 000 *samples: Orocos KDL, Orocos PyKDL, and PyMoCo. The red marker is at the position of the maximum response time.*

3.4 Real-time Characteristics

The response times of the presented implementations were measured in an experiment. The goal with this experiment was to compare the different implementations to each other rather than running an absolute benchmark. A test platform was set up, using a PC with an 8 core 2.8 GHz CPU and 8 GB RAM. The operating system was Ubuntu Linux with the CPU frequency scheduler set to "performance". This disables frequency scaling, which was found to have a large impact on the results of this experiment. The low-level interface of the robot was emulated by a C++ program on the same PC as the trajectory generators. Each implementation was tested by recording the response times for 10 000 steps, while commanding a linear tool motion with a constant low speed. The response time is the time from a request for a new joint update is sent to the trajectory generator until the joint update is received in the low-level interface. The results are shown in Figure 3.2.

The measured response times for the C++ implementation have a mean value of 0.265 ms and a maximum value of 0.459 ms. The Python implementation using Orocos PyKDL has a slightly slower response with a mean value of 0.345 ms and a maximum value of 0.488 ms. Finally, PyMoCo has a mean response time of 0.537 ms and a maximum response time of 0.859 ms.

All trajectory generators manage to respond within 1 ms. Using the trajectory generators via Ethernet, connected to a real robot, the response time will increase slightly. Industrial robots with real-time interfaces, like KUKA robots with RSI or the Universal Robots UR5 have typically a deadline of a few milliseconds. For example, the UR5 has a deadline of 4 ms. When this deadline is exceeded, the robot skips one interpolation cycle, keeping the last position command. Depending on strategies in the implementation of the real-time interface on the robot, a strategy can be implemented that allows for larger response times, for example by extrapolating. Another strategy could be to set the robot in a security stop modus. Depending on how critical the outcome of one or more missed cycles are on the application, a stricter or more lenient strategy can be chosen.

3.5 Conclusions

Three different trajectory generator frameworks were presented. One framework, PyMoCo, was entirely programmed in Python. The development focus was on creating a general framework that makes it possible to easily implement custom trajectory generators for robots that offer a position-based real-time interface. PyMoCo includes a set of canonical trajectory generators. A second implementation, also implemented in Python, was based on Orocos PyKDL. By using the Orocos library, the computational-intensive parts are computed in C code. Also this trajectory generator comes with a set of canonical trajectory generators. A third implementation was programmed entirely in C++ using the Orocos KDL library. In comparison to the other trajectory generator, it only includes minimal functionality. It was implemented to compare the response times of the Python implementations with a C/C++ implementation.

A timing experiment was conducted to measure the response times of the three trajectory generators. All trajectory generators responded within 1 ms, which is more than satisfactory for the use with the robots in the lab. As expected, the C++ implementation had the fastest response time while PyMoCo had the largest computation time of the three presented frameworks. It is important to notice that not only the response time is important for the choice of the framework but also the functionality that is

required for the desired installation.

The PyMoCo framework and the trajectory generator that is based on Orocos PyKDL are actively used in several laboratory setups, which demonstrates the usability of the developed frameworks.

Chapter 4

Line Tracking with Tool Orientation Control

This chapter presents a system for sensor-based robot control that was presented in the enclosed publications *"Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control*" [SLU+10] and *Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking* [SLM11]. The setup is carried out as an eye-in-hand robot system, where the robot has to follow a line sketched on a surface in a constant distance as shown in Figure 4.1. The tool orientation is controlled such that it is aligned perpendicularly to the workpiece surface.

4.1 Motivation

Many robotic movements can be preprogrammed without any further adjustments while the robot is moving. On the other hand, there are industrial applications where the movements of the working tools have to be adjusted according to changes in the environment or workpieces with uncertain shapes. Some of these industrial applications require that the working tool has the ability to follow one or another form of a line. Examples are the tracking of a line for welding, following the contour of a fabric during a sewing operation or painting applications. These applications can be designed to work fully automated or in cooperations with humans. One example is an application where the human is marking a path on the workpiece where it has to be machined, painted, welded or cut, either in advance or while the robot is moving. When automating such processes, a sensor system is required that is capable of detecting these lines or contours in order to generate an appropriate trajectory of the robot. Especially if the

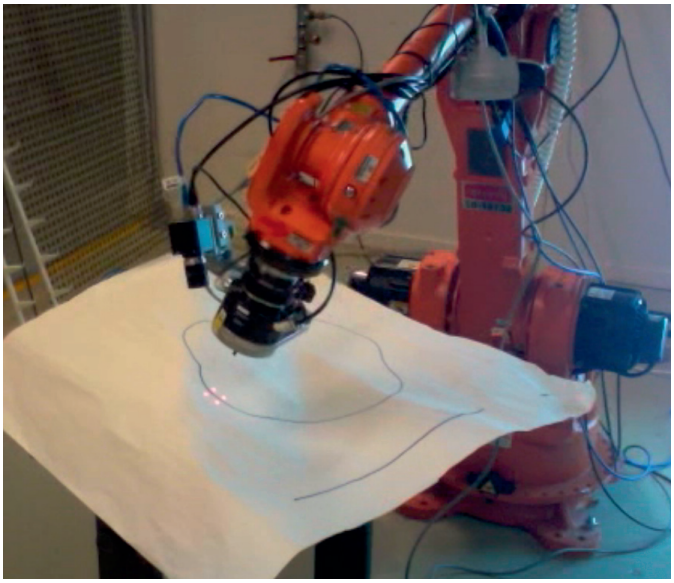


Figure 4.1: The demonstrator system. The robot is following a line with a constant distance between tool and surface.

path cannot be predicted before starting the operation, a tight integration of the sensors in the real-time control loop is needed.

The demonstrator presented in this chapter addresses such applications by introducing a control system where the robot follows a marked line on the workpiece while keeping the tool in a constant distance from the workpiece and the tool orientated perpendicularly to the surface.

4.2 Related Work

The demonstrated work is related to several topics in automation and control. The proposed line-tracking algorithm is closely related to challenges in marine systems where a vessel has to follow a given path. Fossen [Fos02] and Børhaug [Bør08] present line-of-sight-based path-following methods for vessels, that have been the starting point for the algorithm used in the demonstrator system.

Different sensor systems have been presented to follow a desired path on a workpiece surface. A combination of a fixed camera and a force sensor to measure the surface orientation are presented in Hosada et al. [HIA98]. The paper presents a hybrid visual/force servoing to move a robot on an unknown workpiece. The fixed camera is used to estimate the robot's position.

A desired path is marked at the camera image and the robot follows this path using visual servoing, while a force controller is used to hold the tool in contact with the workpiece. The surface normal at the tool is estimated using the force data. This is done by decomposing the measured forces into frictional forces in the direction of the end-effector motion and the normal force.

A system with a camera attached to the robot tool is presented in Zhang et al. [ZCX06]. A hybrid vision/force controller is used for programming the desired path for a painting task. The vision controller is used to move the tool along a painted line on the workpiece. A force controller is used to hold the tool in contact with the workpiece and to determine the surface normal. This is done by following a zigzag path and calculating the orientation from the force data. Experiments are presented that show that it is possible to achieve high accuracy and that the system is suitable for automatically generating tool paths for manufacturing operations.

De Graaf et al. [GAMJ05] presents a robot-based system for real-time 3D seam-tracking for laser welding, using a mechanism to synchronize a seam-tracking sensor with an industrial robot. The authors point out that measurement near the focal spot is necessary to meet the accuracy demands. A triangulation-based seam-tracking sensor that uses a laser-line is mounted on the robot tool. The authors distinguish between two types of tracking strategies. The first type consisted of two steps, first sensor-based teaching of the seam locations, and then laser welding. The second type are strategies that include real-time measurements to track the seam during the welding process. While the first method can lead to higher accuracy due to the absence of constraints for the robot movement, the second method may be preferred due to time saving. The paper presents a synchronization method to synchronize the data from the sensor to the location of the real-time controlled robot. A method for sensor-guided robotic laser welding is proposed. The path is generated from the vision sensor in some distance ahead from the welding spot. The measured locations are stored in a buffer.

Another approach for seam-tracking in a welding process is presented in Zhou et al. [ZLC06]. The seam path is tracked before the welding operation by using visual servoing. A camera is used to capture images of the seam. The tracking is done in 2D. Experiments are presented that show reliable tracking without special light sources for the camera. The accuracy was measured to be adequate for high quality welding processes.

Another 3D real-time seam-tracking system is presented by Kim et al. [KCL+08]. The method is based on the previous mentioned methods presented in de Graaf et al. [GAMJ05] and Zhou et al. [ZLC06]. A

triangulation-based laser sensor is mounted on the tool, with a servo motor that makes it possible to change the working distance by tilting the laser stripe. Experiments are presented that verify that the tracking is working with good accuracy.

The mentioned systems have in common that they are closely connected to industrial cases. The system presented in this PhD work intends to aim at industrial application as well, but the focus is on general methods of sensor-based robot control rather than on one specific case. The testing and development of the real-time interface to the robot, and the real-time analysis of the control loop including the sensors are seen as the main contribution of this work.

4.3 Demonstrator

This section presents design and implementation of the demonstrator setup.

Different cases were defined for the system. In all cases, the tool has to keep a constant distance from the surface.

- Line following on a 2D surface.
- Line following on a 3D surface without tool orientation control, cf. Figure 4.2.
- Line following on a 3D surface with tool orientation control, cf. Figure 4.3.

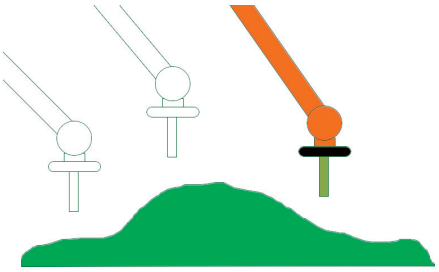


Figure 4.2: 3D line tracking without tool orientation control.

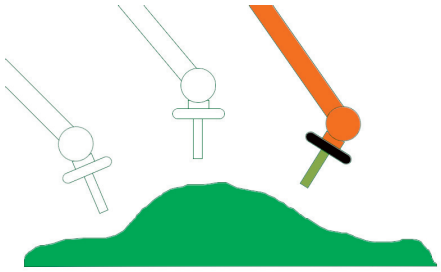


Figure 4.3: 3D line tracking with tool orientation control.

The algorithms presented in this chapter were designed for the latter case. The other cases were tested by deactivating the unused controller components.

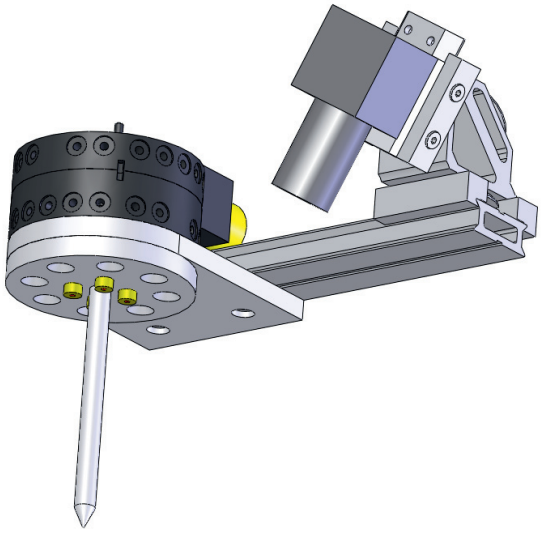


Figure 4.4: The tool mounted on the robot. It includes four lasers and a camera for distance measurements. The tip was designed to protect the sensor system and was not used in the experiments.

4.3.1 System Description

This section describes the demonstrator system. An overview is given of the hardware as well as the control strategies that are applied. Mathematical details can be found in the enclosed publication *"Real-Time Sensor Servoing using Line-of-Sight Path Generation and Tool Orientation Control"* [SLU+10].

Hardware

The demonstrator system is based on a Nachi SC15F 6-axis industrial robot with the real-time interface presented in Section 2.3. The robot is connected to a controller PC running Ubuntu linux with a controller program implemented in C++.

A tool is mounted on the robot's tool flange including four lasers and a Prosilica GC1350 Gigabit Ethernet Camera, see Figure 4.4. Using this tool, the distance between the laser diodes and the workpiece surface can be calculated by triangulation using the laser positions on the camera image, cf. Figure 4.5. The camera is connected to the controller PC.

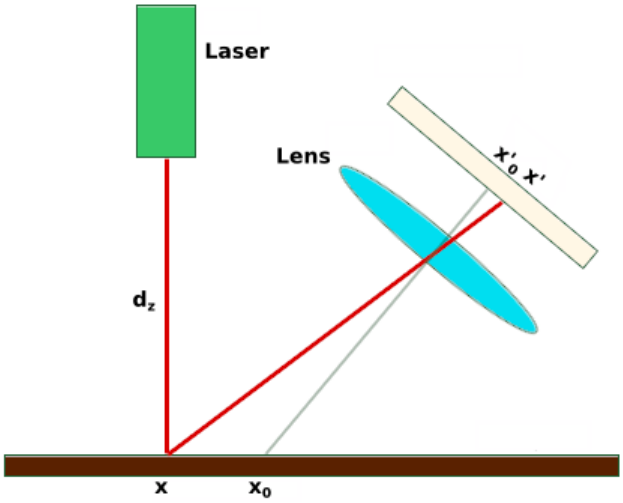


Figure 4.5: Concept of triangulation-based distance measurement.

Tool Distance and Orientation Control

The four triangulation-based distance measurements are used to control the distance between the tool and the workpiece surface as well as the tool orientation. The distance d_z between the tool and the workpiece is calculated by taking the average of the measurements,

$$d_\mathrm{z} = \frac{1}{4}\sum_{i=1}^{4} d_{zi} \ , \tag{4.1}$$

where d_{zi} are the separate distance measurements.

A controller is used, in this example a P-controller, to calculate a velocity v_z that moves the robot tool center point to the desired height,

$$v_\mathrm{z} = -k_\mathrm{z}(d_\mathrm{z} - d_\mathrm{des}) \ , \tag{4.2}$$

where d_des is the desired distance between tool center point and the workpiece.

The outward surface normal $\mathbf{n}$ is calculated by taking the cross product of perpendicular vectors constructed from the positions of the laser projections $\mathbf{p}_n$ in the tool coordinate system,

$$\mathbf{n} = \begin{bmatrix} n_\mathrm{x} \\ n_\mathrm{y} \\ n_\mathrm{z} \end{bmatrix} = (\mathbf{p}_3 - \mathbf{p}_1) \times (\mathbf{p}_4 - \mathbf{p}_2) \ . \tag{4.3}$$

From this normal the error $\mathbf{e}_\theta$ is defined,

$$\mathbf{e}_\theta = \begin{bmatrix} e_{\theta,\mathrm{x}} \\ e_{\theta,\mathrm{y}} \\ e_{\theta,\mathrm{z}} \end{bmatrix} = \begin{bmatrix} n_\mathrm{y} \\ -n_\mathrm{x} \\ 0 \end{bmatrix} \ . \tag{4.4}$$

The correcting angle velocity ω is calculated using a P-controller with the gain k_θ,

$$\boldsymbol{\omega} = \begin{bmatrix} \omega_\mathrm{x} \\ \omega_\mathrm{y} \\ \omega_\mathrm{z} \end{bmatrix} = -k_\theta \mathbf{e}_\theta \ . \tag{4.5}$$

Line-of-sight Path Following

A guidance method was used for trajectory generation that is based on a line-of-sight guidance algorithm. This algorithm is related to guidance controllers used for the navigation of vessels presented in [Fos02] and [Bør08].

Figure 4.6 shows the principle of the used algorithm. The workpiece surface in the area of the projection of the tool center point is recorded by a camera. The tool center point is marked in the figure as a green cross. A circle is laid around the projection with a given radius. Then, the intersections with the line are calculated and set as possible waypoints. The direction is now found towards the desired waypoint, i.e. the waypoint that gives the lowest deviation from the previous direction. This direction is then set as movement direction in the trajectory generator, together with a desired speed in this direction. The direction is updated whenever a new image is received and processed. The behavior of the algorithm can be influenced by changing the radius of the circle. A larger radius makes the system more robust against disturbances and delays in the control system, while a smaller radius results in a faster approach to and less deviation from the desired path. This parameter is very dependent on the application. An analysis of the stability dependent on the radius, velocity and the delay in the control system for the case of approaching a straight line is presented in the attached publication *"Time-analysis of a real-time sensor-servoing system using line-of-sight path tracking"* [SLM11].

The calculation from the waypoint coordinate to the movement direction is done as follows. First, the angle Ψ_LOS is calculated by

$$\Psi_\mathrm{LOS} = \mathrm{atan2}\left(y_k - y, x_k - x\right) \ , \tag{4.6}$$

where (x_k, y_k) are the coordinates of the waypoint and (x, y) are the coordinates of the projection of the tool center point.

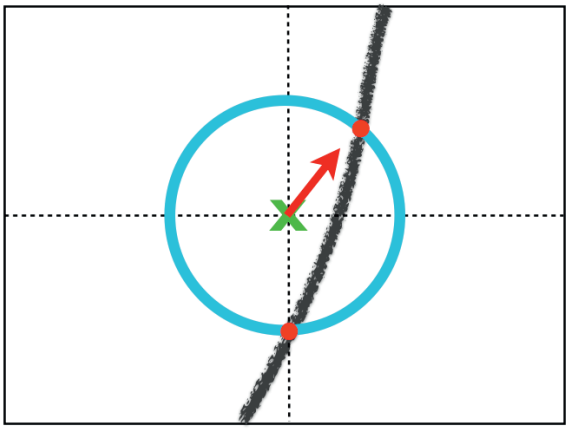


Figure 4.6: The line-of-sight algorithm. The cross marks the projection of the tool center point on the surface. The arrow points in the direction of movement. The blue circle is drawn around the tool center point with a defined radius. The red points are the possible waypoints, from which the waypoint is chosen which gives the lowest derivation from the actual heading.

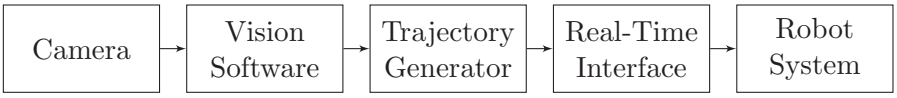


Figure 4.7: Signal flow in the control system.

Then, the velocity in the x-y plane is derived from the movement speed v and the LOS angle Ψ_{LOS} by

$$\begin{bmatrix} v_\mathrm{x} \\ v_\mathrm{y} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\Psi_\mathrm{LOS}) \\ v \cdot \sin(\Psi_\mathrm{LOS}) \end{bmatrix} . \tag{4.7}$$

Together with the previous calculated velocities we get the twist

$$\xi = \begin{bmatrix} v_\mathrm{x} & v_\mathrm{y} & v_\mathrm{z} & \omega_\mathrm{x} & \omega_\mathrm{y} & \omega_\mathrm{z} \end{bmatrix}^\mathrm{T} , \tag{4.8}$$

which is sent to the tool velocity trajectory generator.

4.3.2 Real-time Analysis

In order to model and understand the behavior of the system, an analysis was conducted, documenting the delays in the system. Figure 4.7 shows the components that contribute to the total delay.

Part	Delay
Camera	66 ms
Vision Software	30 ms
Trajectory Generator	15 ms
Real-Time Interface	3 ms
Robot System	120 ms
Sum	234 ms

Table 4.1: Delays in the demonstrator system.

The worst-case delay in the vision system is calculated by adding the time between two successive images, 50 ms, to the communication delay to the PC which is calculated to be 16 ms. The resulting delay is 66 ms. The processing time in the vision software is measured by adding time stamps to the data. The worst-case delay is measured to be 30 ms. The combined time for communication, processing and delays due to the asynchronous control loops of the robot and the vision system is found to be 15 ms. A further 3 ms is added in the real-time interface. The delay that is most difficult to determine is the delay in the robot system. The actual responses are unknown due to the proprietary nature of the low-level system of the robot. An estimation for the delay is the tracking delay, measured in Section 2.5. The presented experiment shows a delay of 120 ms between a commanded and a measured cosine motion of the robot. The total estimated worst-case delay is hence 234 ms. Since most of the delay contributions are independent from each other, and the stated delays are worst-case delays, the real delay is expected to be lower. Table 4.1 summarizes the delays in the system.

To understand the influence of the parameters in the system, a model was derived showing the behavior of the system when approaching a straight line. The starting position is in a given distance from the line with the distance less that the line-of-sight radius. These parameters are the line-of-sight radius R, the robot tool speed v, the cycle time of the sensor system t_s, and the delay in the control loop d. The delay d is the number of discretization steps t_s, forming the complete delay in the control loop. The linear system describing the distance p_y from the line at time k is

$$\mathbf{x}[k+1] = \mathbf{A}_\mathrm{d}\, \mathbf{x}[k] \tag{4.9}$$

with

$$\mathbf{x}[k] = \begin{pmatrix} p_\mathrm{y}[k] \\ p_\mathrm{y}[k-1] \\ \vdots \\ p_\mathrm{y}[k-d] \end{pmatrix}, \mathbf{A}_\mathrm{d} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & -a \\ \hline 1 & & 0 & 0 \\ & \ddots & & \vdots \\ 0 & & 1 & 0 \end{array}\right), \tag{4.10}$$

where a is

$$a = \frac{v\, t_\mathrm{s}}{R}\ . \tag{4.11}$$

v is the tool speed. t_s is the step size, i.e. the cycle time of the sensor system. R is the line-of-sight radius. The system has $d+1$ states. Figure 4.8 shows the stability regions of the parameter a plotted against the delay d. The figure shows that the stable area quickly decreases with increasing delay. This means that it is important to decrease the delay in the system in order to make the system more stable.

It can be seen that a is proportional to the tool speed v, the step size t_s and inversely proportional to the line-of-sight radius R. This means that a larger radius R increases the stability of the system, while the tool speed v and the step size t_s have to be decreased in order to increase the stability.

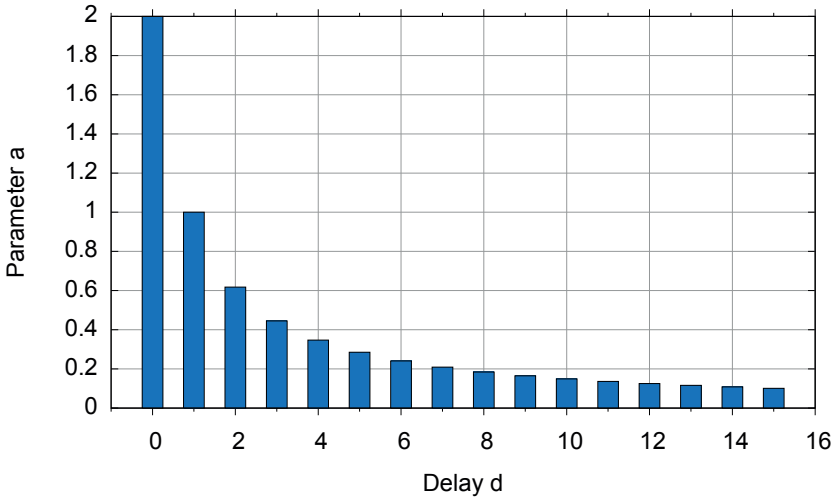


Figure 4.8: The stability regions for parameter a plotted against the delay d. d is the number of delayed steps in the control loop due to the communication and computation delays. The parameter a is proportional to the tool speed and the step size. Furthermore, it is inversely proportional to the line-of-sight radius.

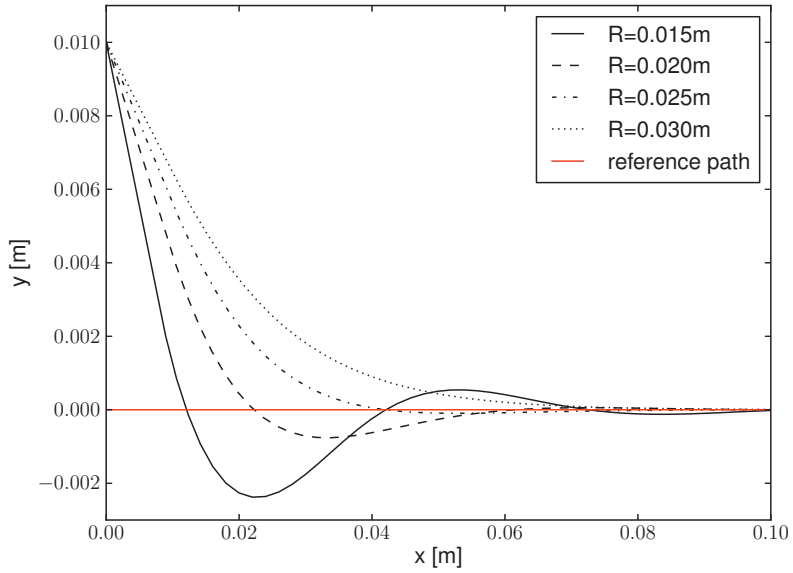


Figure 4.9: Step response as a function of the line-of-sight radius. The starting point is 0.01 m *from the line, i.e. at the coordinate (0.00, 0.01).*

The system was simulated on a PC in order to analyze the behavior in different situations. Figure 4.9 shows the step response of the simulated system while approaching a line from a starting point 0.01 m from the line. The experiment was conducted for different radii. The figure shows that the behavior of the algorithm is very dependent on the chosen radius. A compromise has to be found between fast approach to the line and less oscillation.

A series of experiments was done by comparing simulation results with the real response. Different delays were chosen for the simulation in order to gain an understanding whether the estimated delay is an acceptable approximation. The simulations were conducted with 200 ms delay and 250 ms delay. Figure 4.10 and Figure 4.11 show the system following a corer in the path with two different velocities, $40\,\mathrm{mm\,s^{-1}}$ and $100\,\mathrm{mm\,s^{-1}}$ while keeping the line-of-sight radius constant at 25 mm. It can be seen that the real path is quite near the simulated paths. The deviations can be explained by the unknown behavior of the low-level system and the mechanical system of the robot. A small error is also introduced in the process of fitting the coordinate systems of the simulated and the measured data. The experi-

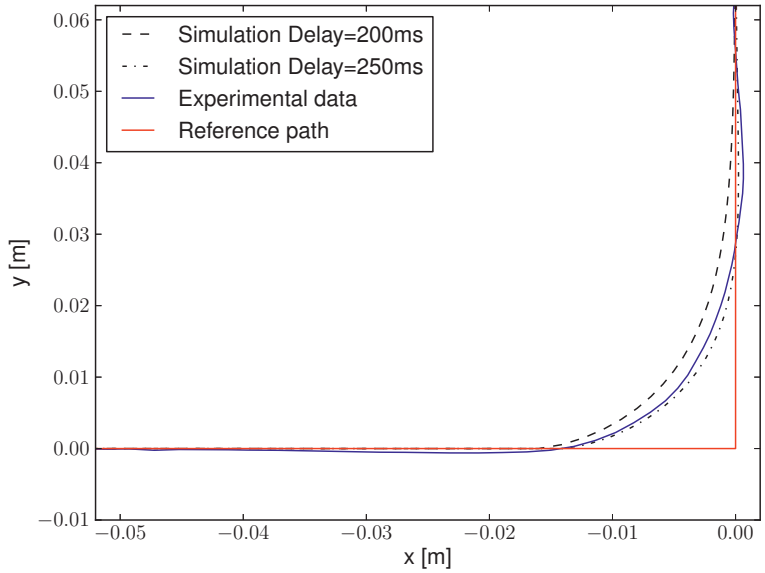


Figure 4.10: Simulation and experiment for a tool velocity of $40\,\mathrm{mm\,s^{-1}}$. *The rounded corner is due to the line-of-sight radius of* 25 mm. *The robot does not react instantly when the corner is detected. This is due to the delay in the control loop.*

ment shows that the delay in the real system is in the magnitude of the estimated delay. The experiment with $40\,\mathrm{mm\,s^{-1}}$ tool speed results only in a small overshoot, while the response of the experiments with $100\,\mathrm{mm\,s^{-1}}$ tool speed results in a damped oscillation. In both cases, the change towards the new direction is not instant when the line-of-sight radius reaches the corner. This is due to the delay in the control system.

4.4 Conclusion

A system was presented demonstrating line tracking on an unknown workpiece with tool orientation control to keep the tool perpendicularly orientated to the workpiece surface. The distance between the tool and the workpiece is controlled to be constant while the robot moves along the line. A triangulation-based sensor system was proposed consisting of a camera and four laser diodes. The sensor system was used to calculate the distance between workpiece and the tool, and the surface normal. A line-of-sight

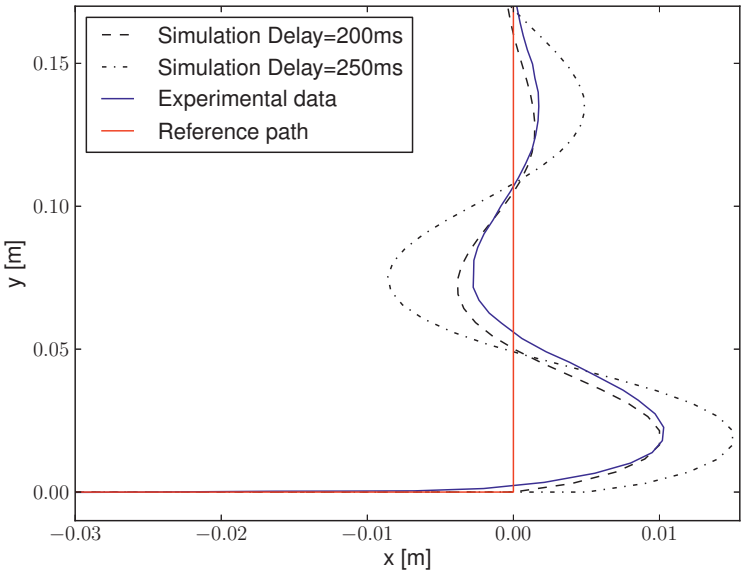


Figure 4.11: Simulation and experiment for a tool velocity of $100\,\mathrm{mm\,s^{-1}}$. *The line-of-sight radius is* $25\,\mathrm{mm}$. *Due to the large velocity and delay the robot is overshooting.*

based algorithm was used to move the robot tool along a line drawn on the workpiece. A real-time analysis was conducted in order to identify the delays in the control loop. Models were created for the case of approaching a straight line. The influence of the line-of-sight radius and the movement speed on the behavior of the system were discussed. Experiments for the case of moving around a corner were conducted and the measurement data was compared with simulated data in order to verify the identified delays.

Chapter 5

Automated Sewing

This chapter presents an automated sewing cell that is able to sew together two parts of slightly different shapes. Sensor-based real-time control is used to hold a constant tension in the workpiece as well as to control the seam allowance. The system was presented in the attached publications *"Experiments towards Automated Sewing with a Multi-Robot System"*[SW12], *"Real-Time System Integration in a Multi-Robot Sewing Cell"*[SWL12] and *"Real-Time Analysis of a Multi-Robot Sewing Cell"*[SLM13].

5.1 Motivation

The automation of the sewing process is a difficult task in automation. Many challenges arise from this subject. Operations that are easily handled by humans result in complex systems when automated, often with extensive use of sensors. The uncertain characteristics of the non-rigid workpieces lead to difficulties during the handling, and to uncertain measurements and estimations. Small variations in the stiffness may result in different mechanical behavior and to a different response in the control system as one example. Although many related tasks can be solved with conventional robot interfaces, the control of the sewing process strongly benefits from high control frequencies and real-time control of the robots. This is especially important at higher sewing speeds. There is a request from the industry to work on the automation of these processes, both due to quality improvements for complex sewing automation and due to the high labor costs, especially in high-cost countries such as Norway.

The demonstrated case is part of several research projects in the working area of automated sewing. The roots of the project go back to 2005 when the TEMPO project was funded by The Research Council of Norway and

industrial partners to work on cases and studies related to usage of sensors in automated production, for example seam tracking during a sewing operation. The follow up project Automatisert 3D Sammenføyning (Automated 3D Sewing) was started in 2008 with focus on sewing of synthetic fiber to leather parts and on initial experimentation on sewing of 3D-shaped assemblies. The delivery was a demonstrator system based on a sewing machine and a real-time controlled robot to handle both the stacking of the different parts and the sewing operation. The demonstrator system was presented in Wetterwald et al. [WDRU08]. The success of the project led to the project Robust Industriell Søm (Robust Industrial Sewing Automation), founded in 2011, that focuses more on industrial stability of the already developed demonstrators as well as on joining of parts with different shapes in order to create 3D-shaped assemblies. This PhD work, which was part of the SFI Norman research program[1], was conducted in close cooperation with the mentioned projects.

5.2 Related Work

Several research groups have worked in the area of handling of automated sewing in recent years. This chapter presents an overview of some projects related to the presented work.

In Gershon and Porat [GP88] and Gershon [Ger90], an automated sewing cell consisting of one robot and a sewing machine is presented. The sewing machine speed is set by a PC. The tension in the work piece is controlled using a force sensor in the control loop. Machine vision is used in a separate control loop for the seam allowance. Experiments compare the system with a simulated model. Discrepancies are due to the assumption of stiff cloth panels, perfect images and an accurate robot. It is reported that the seam quality varies considerably between different fabric types.

Gershon [Ger93] defines different categories for the handling of flexible materials, either sensor-based or sensor-less. Gershon writes that complex tasks may be best solved using a combination of different strategies. He emphasizes the difficulty of automated sewing due to buckling and different forces acting on the fabrics, but points out that satisfactory performance can be achieved using feedback control.

Paraschidis et al. [PFV+95] presents a robotic system for handling flat textile materials. The authors point out the difficulties of the handling of non-rigid materials due to unknown material characteristics. They use

[1] `http://www.sfinorman.no`

vision and force/torque sensors to experiment with different handling operations. They identify problems and challenges during the experiments, including unpredictable behavior of the material, problems with lighting for the vision analysis, calibration errors, and noise in the force/torque measurements. They highlight the benefits of previous knowledge of material characteristics in order to develop fast and reliable algorithms.

In Gottschalk and Seliger [GS96], a device for handling curved fabrics during a sewing operation based on rollers in front of and behind the needle is presented. Different feeding speeds allow for adapting to different seam lengths of the two workpieces. The seam path is planned based on experimentally investigated material properties. The authors mention that the process of insertion of the workpieces into the system has to be automated or simplified in order to apply the system in the industry.

In Seliger and Stephan [SS98], an overview of the challenge of automated sewing is presented, especially the sewing of 3D-shaped products. The focus is on material handling. The authors suggest adaptive control strategies based on measurements of the seam allowance and the feed rate during the sewing operation

In Kudo et al. [KNMB00], a system based on two robots that cooperate to handle a single fabric similar to a human worker during the sewing operation is presented. The implementation includes controllers for tension control, control of the pressing force, and synchronization with the sewing machine speed. A visual tracking controller controls the position of the fabric during the sewing operation. The applied robots are a 4-DOF and a 5-DOF robot. These robots were selected to match the task complexity and the available space. Experiments are presented to show the effectiveness of the system. The experiment that is most related to the work in this chapter is sewing along a curved line, which is done smoothly and only with small errors.

A sewing cell demonstrator that is closely related to this work is presented in Wetterwald et al. [WDRU08]. The sewing cell is able to sew leather parts to similarly shaped fiber parts. It is based on a single robot and a sewing machine. A sensor is included to measure the part position during the sewing operation based on laser-triangulation. An optical velocity sensor is used to synchronize the robot speed with the sewing speed. Experiments are presented, demonstrating that the system is able to produce parts with a satisfactory level of quality. However, the authors state that work has to be done regarding the quality stability.

Another sewing concept is presented by Winck et al. [WDBH09]. The authors remark that as of the date of their publication no fabric control

strategy has made the jump from a demonstrator system to an industrial system. One reason for this has been the lack of robustness of the used control methods, for example synchronization between a robot and the sewing machine in order to prevent buckling. As another reason, they mention the research focus on sewing single fabrics, while there is a demand of sewing together two fabrics. Based on these challenges, they present an approach based on a servo-controlled feed mechanism. Servo motors are mounted on the sewing machine to control the feeding as well as the orientation of the fabrics. The fabrics are controlled independently and separated by a thin plate. By doing this, they move away from a human-like control method. Path control is done by pattern recognition of the fabric together with an open-loop controller. A prototype is presented that successfully controls fabric through multiple trajectories. Due to the open-loop fabric position control, there is inaccuracy in the fabric position. Therefore the authors emphasize the need for sensor-based feedback control.

In Koustoumpardis et al. [KZA06], an overview of a robotic system for handling of non-rigid materials, as well as fabric handling strategies for the whole sewing process is presented. The presented demonstrator system consists of a single robot and a sewing machine. The gripper is designed to press the fabric to the table. A force sensor is mounted on the tool to measure the tension or compression in the fabric. The robot velocity is regulated by a neuro-controller. The goal is to synchronize the robot speed with the sewing machine speed. A fuzzy-based visual servoing controller guides the fabrics during the sewing process. The authors identify the following handling tasks in a sewing process: ply separation, placement on the working table, manipulation towards the sewing needle and tension control during the sewing operation. A set of sub-tasks is described for preparation of the sewing operation including the planning of the sewing process for the present fabric. The sewing operation is divided into manipulation of the part towards the needle, the stitching process and the rotation around the needle. This rotation is used to position the part for the next edge. More details of the neural network control system are presented amongst others in Koustoumpardis and Aspragathos [KA11].

5.3 Design and Implementation

This chapter presents implementation details of the sewing cell that is used as the main demonstrator platform for the presented work. The design and implementation were made by a small team in which the author played a major part.

Figure 5.1: Ekornes recliner with footstool. The sewing of the cover of the footstool was chosen as demonstration case for the sewing cell. Image source: ekornes.no.

5.3.1 Design Criteria

The following criteria were identified prior to the design of the system:

- The system has to be able to sew together two parts with slightly different shape.
- The system should be as robust to material variations and uncertainties in material characteristics and shapes as possible.
- The system should not introduce unnecessary complexity, but rather be constructed to be as simple as possible.
- The system should allow for rapid prototyping in order to experiment with the control system as well as the program logic without big changes in the software.

As industrial case, the automated sewing of a cover for a footstool as shown in Figure 5.1 was chosen. Preliminary experiments, presented in the attached publication *"Experiments towards Automated Sewing with a Multi-Robot System"* [SW12], show that there are large variations of the force response for clamped parts, cf. Figure 5.2. Further variations are

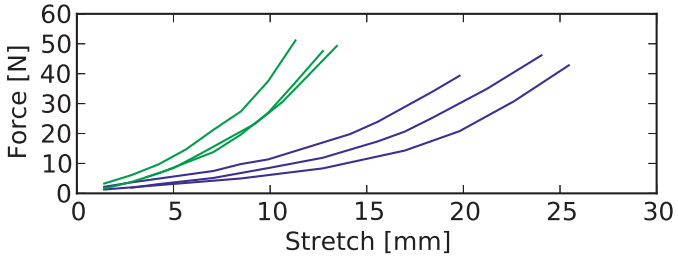


Figure 5.2: The force response for a clamped part with two different lengths. Each experiment was repeated three times (green: 150 mm*, blue:* 250 mm*).*

introduced by varying material thickness and shape. These uncertainties underline the need for sensor-feedback to control the sewing process. Based on the presented challenges, the design criteria, and the experiences of the preceding work, it was chosen to build a system based on two robots. This makes it possible to control the feeding of the two parts independently. Sensors and robots with real-time interfaces were chosen in order to achieve control frequencies in the order of 100 Hz. This frequency range is quite typical for position-based low-level interfaces for industrial robots.

Regarding the software design, it was decided to build a distributed system rather than a monolithic one in order to divide the complex system into simpler components with distinct boundaries of concerns. The main programming language was chosen to be Python due to its support for fast prototyping and the large collection of available libraries. Python has also the possibility to use compiled libraries, for example C/C++ libraries for computation-intensive parts. This makes it possible to run complex computations nearly as fast as in a compiled language. Microcontroller code and low-level drivers were programmed in C/C++.

It was decided to regularly reinvestigate the real-time characteristics of the code in order to verify the decision of choosing a scripted language instead of a compiled language.

5.3.2 Sewing Cell Demonstrator

This section describes the hardware of the sewing cell shown in Figure 5.3 as well as the integration of the hardware components in the control system. The structure of the different components is shown in Figure 5.4.

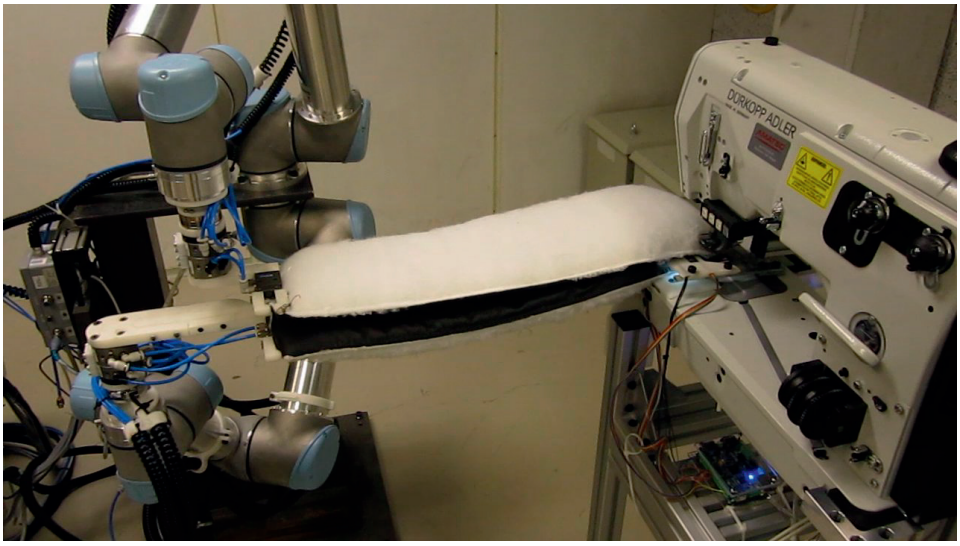


Figure 5.3: The sewing machine with the two robots. Force sensors are mounted on the end-effectors to measure the tension in the work pieces.

Sewing Machine

The core of the sewing cell is a Dürkopp Adler DA195 industrial sewing machine. This sewing machine was chosen due to its ability to set different stitch lengths for the two parts. The stitch lengths are mechanically adjustable and are considered to be included in future control methods. The drive speed is controlled by a microcontroller board connected to the drive controller. The same board is also connected to the pneumatic valves to control the pressure foot, the thread cutter and the thread tensioner. An Ethernet connection to the control PC is used to include the sewing machine functions in the control system.

Industrial Robots

Two UR5 robots from the manufacturer Universal Robots are installed in the sewing cell to lead the workpieces into the sewing machine. The robots are mounted in a configuration that allows for independent control of both workpieces during the sewing operation, as shown in Figure 5.3.

The real-time interface described in Section 2.4 is used to control the robot in combination with the trajectory generator described in Section 3.3.2.

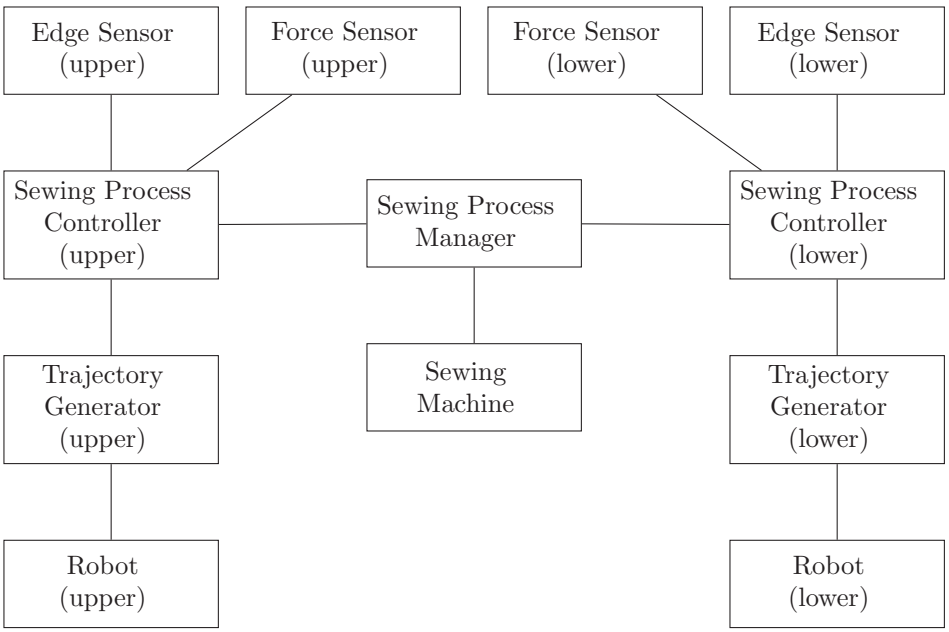


Figure 5.4: Structure of the sewing cell. There are individual components for the control loops of the upper and the lower robot. For each robot there is a sewing process controller which includes an edge controller and a force controller. The sewing process manager is responsible for setting up the sewing process controllers and controlling the sewing machine.

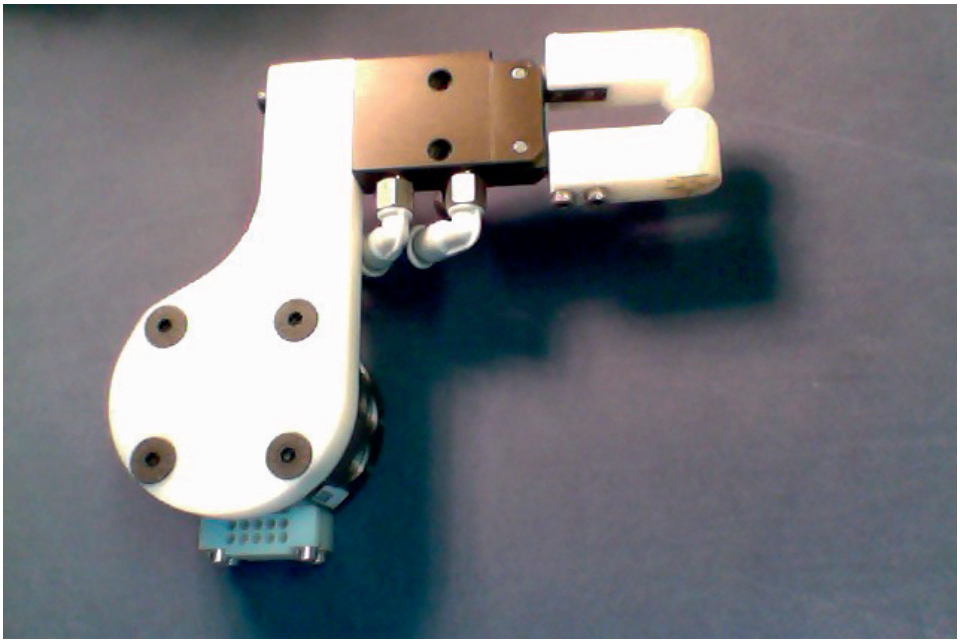


Figure 5.5: The pinch tool for gripping the workpieces.

Robot Tools

Different tools were developed for different tasks in the sewing cell. Figure 5.5 depicts a pinch tool that is used for the sewing operation. The tool is limited to experiments where the gripping point is outside the sewing path. This limitation is due to the simplicity of the gripper and the lack of functionality for grip shifting.

A two-needle gripper is under development to extend the functionality of the pinch gripper. The gripper includes two needles which can hold the workpiece at two gripping points at the same time. The needles can be operated independently. The gripper was designed to be able to sew the main part of a seam while holding in the sewing line. The last part of the seam is done by holding the part in a gripping point away from the sewing machine. To avoid collisions with the sewing machine for the last part of the seam, the tool is rotated around the new gripping point.

For experiments concerning the whole sewing process including material handling, a set of grippers was designed based on a gripping area instead of a gripping point. The grippers are shown in Figure 5.6.

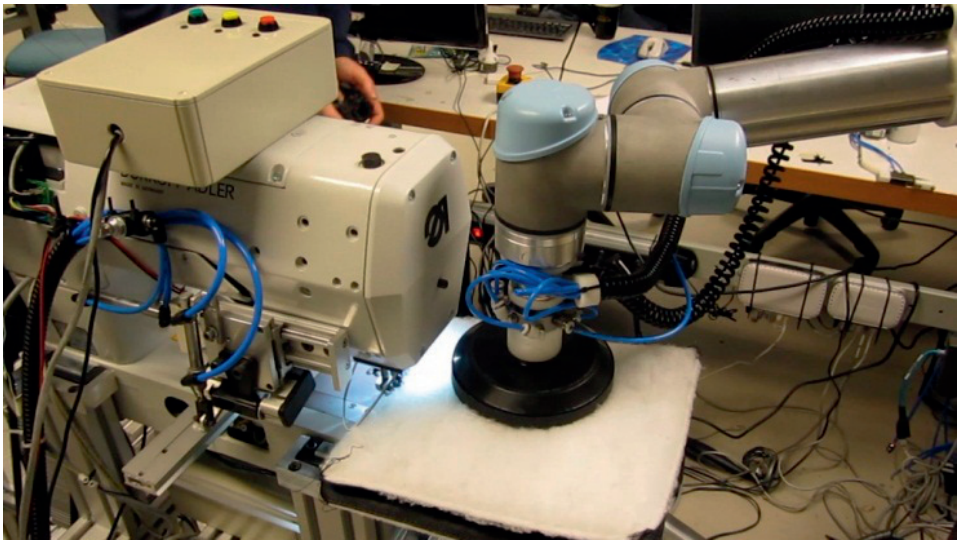


Figure 5.6: An alternative tool for gripping the workpieces.

Force Sensors

Force sensors are mounted to the robot end-effectors to measure the force in the fabrics during the sewing operation. The force sensors are ATI mini 45 with a Net/FT controller box. They are connected to the control PC via Ethernet using an open source driver[2]. The sensor rate is set to 1 kHz. This frequency is well above the frequency of the robot interface which is 125 Hz.

Edge Sensors

Position control is an important part of the sewing operation. The system must be able to place the seam in a given distance from the edge, i.e. with a fixed seam allowance. To achieve this, an edge sensor is needed to measure the position of the work piece in front of the needle. A wing-shaped plastic sensor plate was built and placed between the two parts as shown in Figure 5.7. The plate is the housing for two optical linear arrays, one on each surface. The sensor array communicates a one-dimensional image to a microcontroller board that is connected to the control PC via Ethernet. On the PC, an edge detector computes the edge position based on a threshold for the amount of light that indicates which parts of the sensor is covered by the work piece.

[2] NETFT_RDT_DRIVER from `http://ros.org`

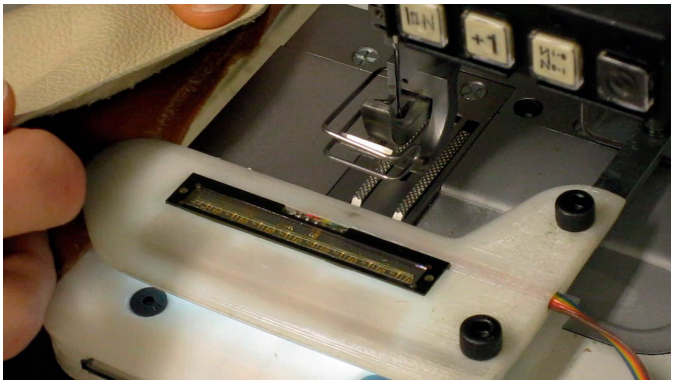


Figure 5.7: The edge sensor plate includes one optical sensor array on each side to measure the position of the edge near the needle. The plate is placed between the two workpieces during the sewing operation

Due to issues with friction at the sensor plate, a new sensor system was designed. It is based on a stereovision system. Figure 5.8 shows the cameras mounted on the sewing table. Another problem was inaccurate measurements for thicker parts due to an air gap between the sensor and the workpiece edge. Figure 5.9 shows a screenshot of the edge detection software. The new system is still under development.

Software Platform, Communication and Middleware

A distributed software system was built that keeps the different parts of the system independent and as simple as possible. ROS[3] (Robot Operating System) was chosen as middleware due to its features as well as the acceptance and distribution in the robotics research community. ROS is a set of libraries and tools for development of robot applications. It provides a communication system including messages and services. Using a C++ or Python API, it is possible to develop a distributed system for the different parts of a software system, e.g. sensor drivers, robot interfaces, controllers, administrative code. The largest amount of the code was implemented in Python, while microcontroller code and low-level drivers were written in C/C++. The operating system of the control PCs was Ubuntu Linux with low latency kernel.

[3] `http://www.ros.org`

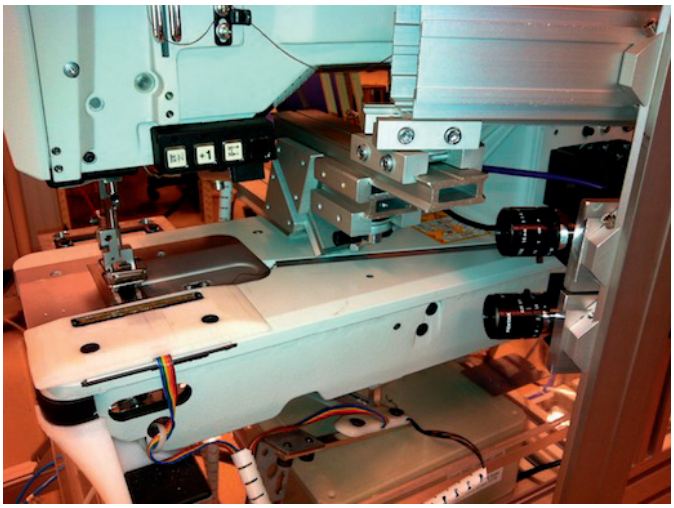


Figure 5.8: A stereo camera is mounted to the sewing machine in order to record the edge position.

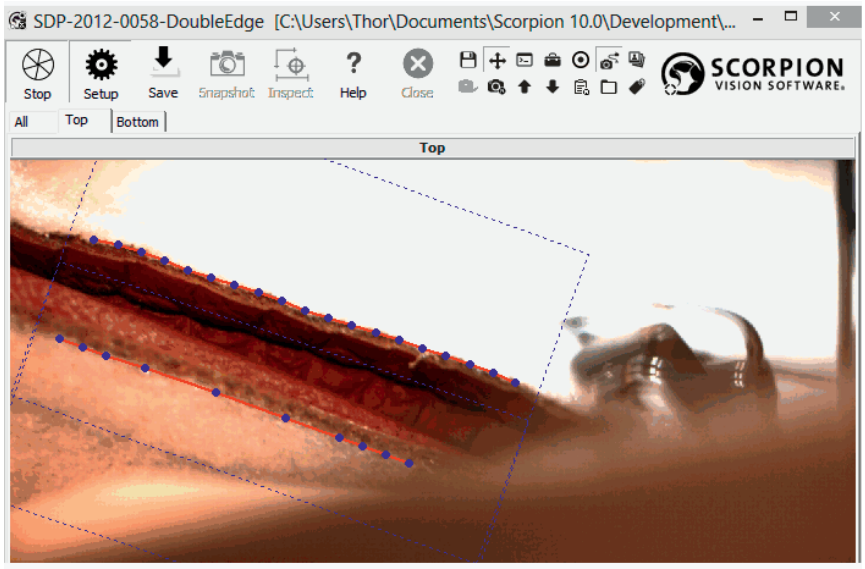


Figure 5.9: Edge detection based on the stereo camera system is done in the vision software.

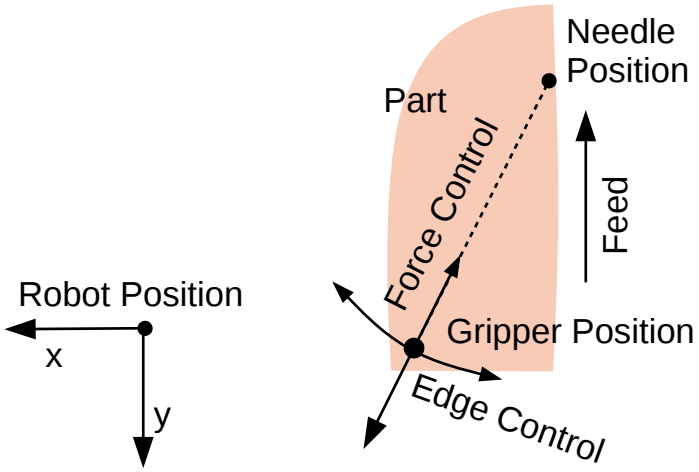


Figure 5.10: The sewing coordinate system.

5.3.3 Control System

This section explains the control system of the sewing demonstrator, presented in the attached publication *"Real-Time System Integration in a Multi-Robot Sewing Cell"* [SWL12].

Control Coordinate System

A sewing coordinate system has been defined to describe the directions of the force control and the edge control, see Figure 5.10.

The force controller is programmed to control the sewing force between the gripping point and the needle, which results in end-effector movements towards or away from the needle. The edge position of the part is controlled by rotating around the needle, which results in end-effector movements on a circular path around the needle. The two components are then combined with a feed-forward velocity to a resulting end-effector velocity.

The end-effector velocity $\mathbf{v}_{\text{cmd}}$ that is commanded to the robot is calculated as follows:

$$\mathbf{v}_{\text{cmd}} = \mathbf{v}_{\text{force}} + \mathbf{v}_{\text{edge}} + \mathbf{v}_{\text{ff}} \tag{5.1}$$

with $\mathbf{v}_{\text{force}}$ being the velocity component from the force controller, $\mathbf{v}_{\text{edge}}$ the velocity component from the edge controller and $\mathbf{v}_{\text{ff}}$ the feed-forward velocity.

The feed-forward velocity v_{ff} is calculated from the frequency of the sewing machine drive f_{drive} and the estimated stitch length l_{stitch},

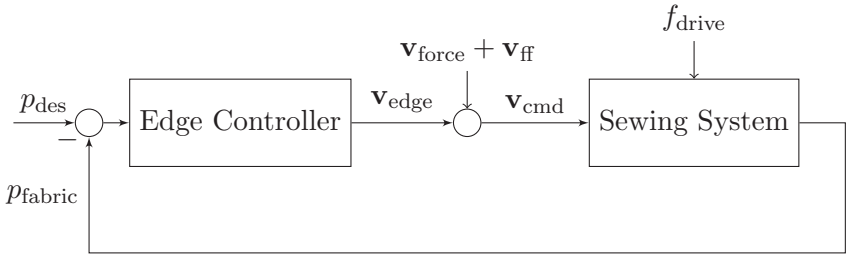


Figure 5.11: The edge control loop.

$$\mathbf{v}_{\text{ff}} = l_{\text{stitch}} f_{\text{drive}} \hat{\mathbf{n}}_{\text{feed}} \tag{5.2}$$

where $\hat{\mathbf{n}}_{\text{feed}}$ is the unit vector in feed direction. The estimated stitch length is amongst others dependent of the stitch length configuration of the sewing machine, the sewing force, the material characteristics, and the thickness of the material.

Edge Control

To keep the seam in a constant distance from the edge, an edge controller was developed. As mentioned before, the edge controlling is done by rotating the work piece around the needle.

The control loop is depicted in Figure 5.11. The control input is the edge error which is constructed from the edge measurement p_{fabric} and the desired edge position p_{des} by

$$\mathbf{v}_{\text{edge\_sensor}} = -k_{\text{edge}}(p_{\text{fabric}} - p_{\text{des}})\hat{\mathbf{n}}_{\text{feed}}^{\perp} \tag{5.3}$$

where $\hat{\mathbf{n}}_{\text{feed}}^{\perp}$ is the unit vector perpendicular to the feed direction. $\mathbf{v}_{\text{edge\_sensor}}$ represents the desired movement on the edge sensor.

A desired angular velocity around the needle $\boldsymbol{\omega}_{\text{needle}}$ is calculated from the desired velocity on the edge sensor,

$$\boldsymbol{\omega}_{\text{needle}} = \frac{\mathbf{r}_{\text{sensor}} \times \mathbf{v}_{\text{edge\_sensor}}}{\|\mathbf{r}_{\text{sensor}}\|^2} \ , \tag{5.4}$$

with $\mathbf{r}_{\text{sensor}}$ being the vector from the needle to the edge sensor. The desired end-effector velocity caused by the edge control $\mathbf{v}_{\text{edge}}$ is then constructed from the desired angular velocity and the gripper-to-needle vector $\mathbf{r}$ by

$$\mathbf{v}_{\text{edge}} = \mathbf{r} \times \boldsymbol{\omega}_{\text{needle}} \ . \tag{5.5}$$

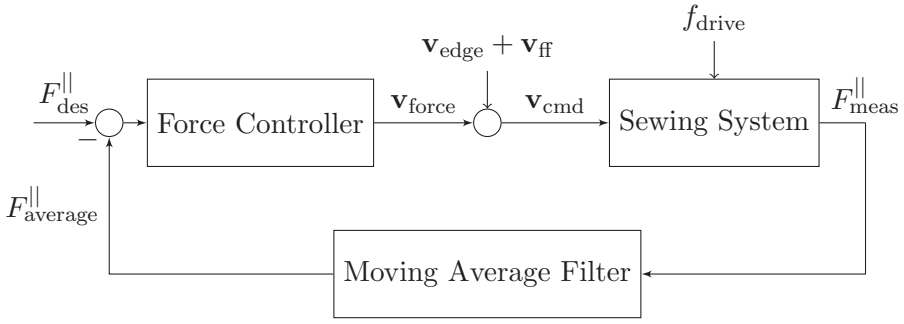


Figure 5.12: The force control loop.

Force Control

Figure 5.12 shows the force control loop that is used to keep the sewing force constant.

Unlike the edge controller, the force controller uses filtered sensor input. This is due to the large influence of the feeding mechanism which leads to oscillations in the force measurement. A moving average filter with the time window of one stitch is used to compute the average force in the sewing direction,

$$F^{||}_{\text{average}} = \hat{\mathbf{r}}^T \mathbf{F}_{\text{average}} \ , \tag{5.6}$$

where $\hat{\mathbf{r}}^T$ is the gripper-to-needle unit vector.

The velocity component at the end-effector is calculated

$$\mathbf{v}_{\text{force}} = \left[k_{\text{p}}(\|\mathbf{r}\|)(F^{||}_{\text{average}} - F^{||}_{\text{des}}) + k_{\text{i}}(\|\mathbf{r}\|)\zeta \right] \hat{\mathbf{r}} \tag{5.7}$$

with

$$\dot{\zeta} = (F^{||}_{\text{average}} - F^{||}_{\text{des}}) \ . \tag{5.8}$$

To compensate the changes in the force response due to the changing gripper-to-needle distance $\|\mathbf{r}\|$), k_{p} and k_{i} are scaled linearly with the distance:

$$k_{\text{p}}(\|\mathbf{r}\|) = k_{\text{p0}}\|\mathbf{r}\|, \ k_{\text{p0}} > 0 \tag{5.9}$$

$$k_{\text{i}}(\|\mathbf{r}\|) = k_{\text{i0}}\|\mathbf{r}\|, \ k_{\text{i0}} > 0 \ . \tag{5.10}$$

5.4 Real-time Analysis

This section summarizes a real-time analysis of the sewing cell. The results have been presented in the attached publication *"Real-Time Analysis of a Multi-Robot Sewing Cell"* [SLM13].

In the planning phase of the sewing cell, it was decided to use real-time controlled robots to solve the task of automated sewing. Another design choice was to use Python as main programming language, motivated by the high flexibility and usability of the language. To verify that the choices were feasible, an analysis of the real-time characteristics of the control loops including the sensors and the mechanical system was carried out. One objective was to analyze whether the control system was able to benefit from the robots' low cycle time and tracking delay. Another objective was to detect possibilities for improvement of the system by identifying large delays that could be decreased by hardware or software modifications.

5.4.1 Delay Measurements

Due to the distribution of the system on different hardware and software platforms, it is difficult to obtain a precise overview of all delays in the system. It was decided to estimate the delay in the main control loops by measuring the delay in different parts of the signal paths. Figure 5.13 shows the different paths that were looked into.

The paths are as follows:

1: red The time from sending a command from the trajectory generator to the robot until a movement is measured at the force sensor

2: green The time from receiving force data until a corresponding command is sent from the trajectory generator to the robot

3: blue The time from receiving edge data until a corresponding command is sent from the trajectory generator to the robot

4: purple The time when the computation in the sew controller is triggered by the new joint state update until a new twist is sent to trajectory generator

Delays in the Robot System

The measurement of delays in the robot system is difficult since it is not possible to directly communicate time stamps through the physical system. Another issue is the proprietary low-level controller of the robot with unknown parameter settings and control strategies. Even though the actual positions of the robot can be obtained from the real-time interface, the accurate point in time when the position is recorded is unknown. To meet these challenges, an experiment was designed that records the delay from

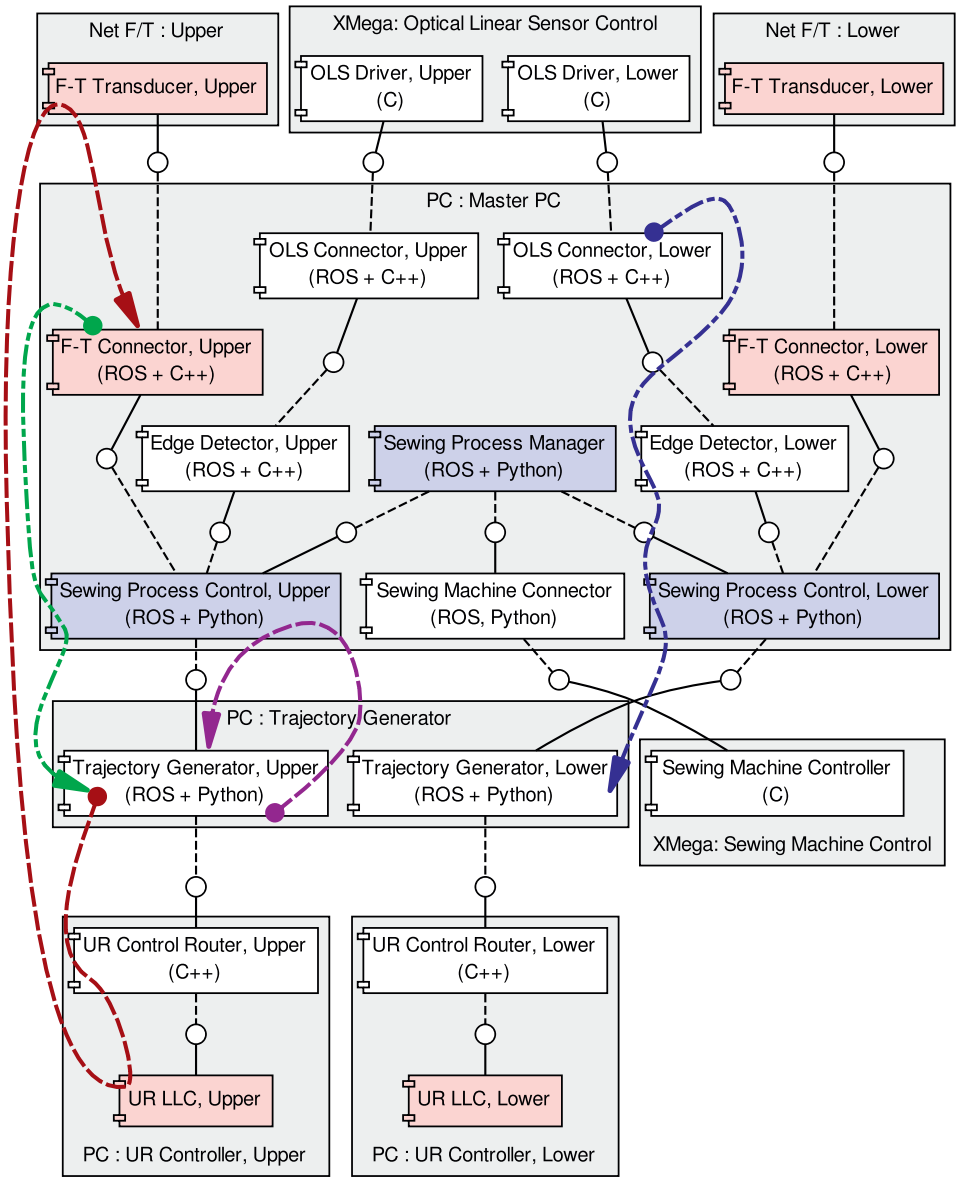


Figure 5.13: The deployment of the different components including the paths for the real-time measurements. The paths are symmetric for the "Upper" and the "Lower" control loop. The "Net F/T" nodes represent the force sensors. The "XMega: Optical Line Sensor Control" node includes the edge detection sensors and the corresponding hardware platform.

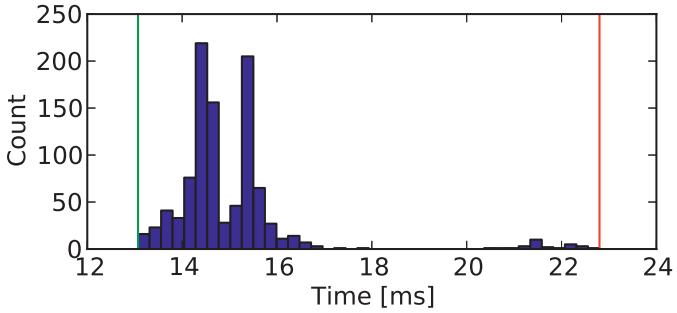


Figure 5.14: Delay through the robot system from obtaining a new robot state in the trajectory generator until a movement can be measured at the force sensor. The green marker is at the position of the minimum response time and the red marker at the position of the maximum response time.

when the robot is commanded to move from a constant position to another position until the robot began to move. A ramp was commanded to the robot through the trajectory generator and a time stamp was recorded. The point in time when the robot began to move was then measured by recording data on the force sensor. This path is marked in Figure 5.13 by the red arrow. Since the force sensor has an update frequency of 1 kHz, this measurement is relatively accurate. The time-stamped data is then compared to the previous recorded time stamp of the velocity command.

The result of 1000 experiments is shown in Figure 5.14, including the delay in the force sensor which is stated in the data sheet to be 0.288 ms.

The measured delays are between 13 ms and 23 ms. Most measurements are in the range of 14 ms to 16 ms, while a small peak also can be found at 22 ms. The peak at 22 ms indicates that there are a few cycles where the movement command is processed in a later cycle than most commands, i.e. about 8 ms later.

Time Analysis for the Force Measurement

The remaining delay in the force control loop is the delay from the point in time when a new force value is received until a velocity command based on this force value is sent to the robot. To measure this contribution to the total delay, the force data is time stamped and the time stamp is evaluated when the data is sent to the robot. This path is marked by the green arrow in Figure 5.13.

Figure 5.15 presents the histogram of 10 000 measured delays. It can be seen that the delays are between 5.1 ms and 10.7 ms. The average delay

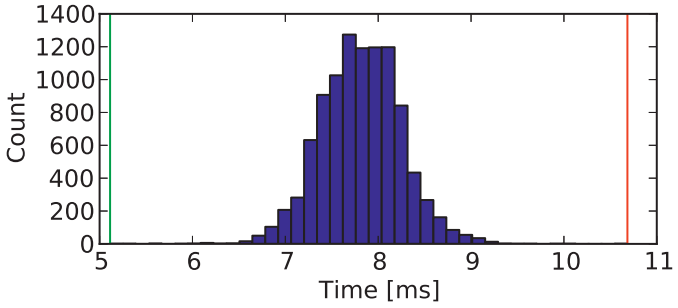


Figure 5.15: Delays for the force measurements through the ROS system. The green marker is at the position of the minimum response time and the red marker at the position of the maximum response time.

is near 8 ms which is the cycle time of the robot. This delay is due to the asynchronous loops in the trajectory generator. This is because the calculation of the next step already begins right after an update is sent to the robot. The calculated twist for the next update is then stored in the trajectory generator until the next actual value is received from the robot. The communication is depicted in Figure 5.16.

Time Analysis for the Edge Measurement

Another delay measurement was conducted for the edge measurement, the same way as done for the force measurement. A time stamp was added to the data from the line sensor and was evaluated at the point in time when the motion was processed in the trajectory generator. Figure 5.17 shows 10 000 measurements of the blue path in Figure 5.13.

The minimum delay was 5.5 ms and the maximum delay was 21.0 ms. The even distribution between of the delays is due to the asynchronous loops for the trajectory generator which runs at 125 Hz, and the line sensor which runs at about 100 Hz.

Delay in the Sewing Controller

The main loop in the sewing process controller is synchronized with the trajectory generator by using a blocking call to receive the actual joint angles. When a new joint update arrives in the trajectory generator, the sewing process controller is notified. The timing constraints in the real-time interface of the robot do not allow calculation of the new twist before a joint update is sent to the robot. Instead the calculation is triggered at

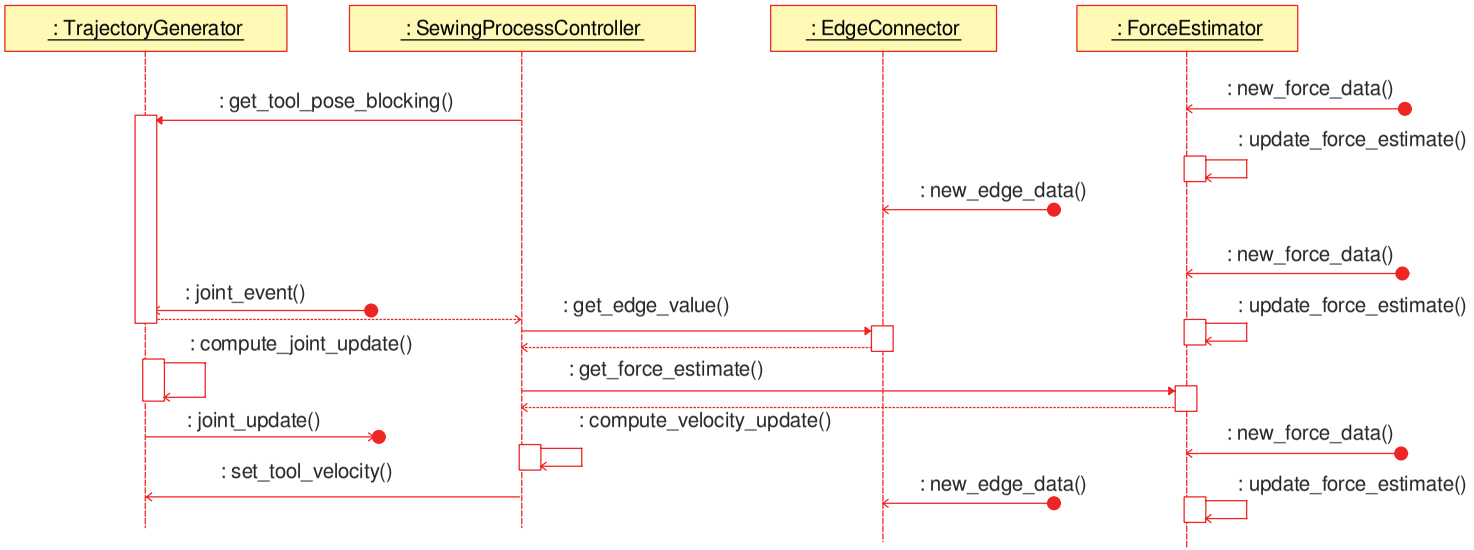


Figure 5.16: The sequence diagram for the sewing operation. The calls "new_edge_data" and "new_force_data" are the entry points for the measurement data. The call "joint_event" is triggered from the low-level interface when a joint position update is present and "joint_update" is called to command new joint positions to the robot.

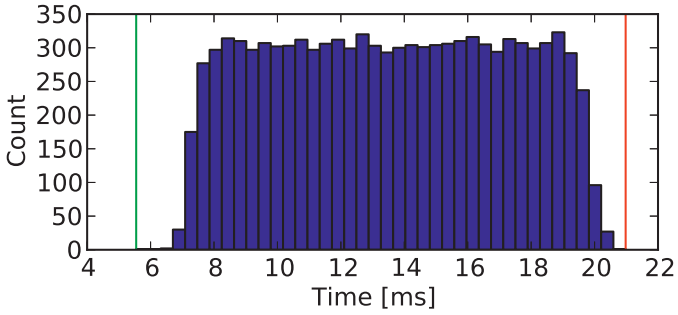


Figure 5.17: Delays for the edge measurements through the ROS system in the active sewing system. The green marker is at the position of the minimum response time and the red marker at the position of the maximum response time.

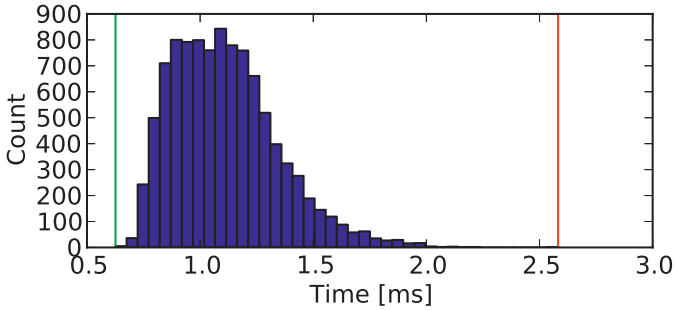


Figure 5.18: Delay for the data from the trajectory generator to the sewing controller and back to the trajectory generator. The green marker is at the position of the minimum response time and the red marker at the position of the maximum response time.

the same time a new joint update is sent to the robot, and then a new twist is calculated and communicated to the trajectory generator to be used in the next cycle. The following experiment examines the time from the notification of the sewing process controller until a new twist is received. The path is shown in Figure 5.13 as the purple arrow. This is to assure that the new twist is not further delayed. The delay for 10 000 measurements is shown in Figure 5.18.

The measured delays have a maximum value of 2.6 ms which is well below the cycle time of the robot. This shows that no further delay is introduced due to computation time.

5.4.2 Delay Summary

The worst-case delays for the different paths in the system are summarized in Table 5.1.

System Part	Max Delay	Path
Tra. Generator - Robot Movement	22.8 ms	Red
Force Sensor - Tra. Generator	10.7 ms	Green
Edge Sensor - Tra. Generator	21.0 ms	Blue
Tra. Generator - Tra. Generator	2.6 ms	Purple

Table 5.1: Worst-case delays for different system parts in the sewing cell.

The overall delays for the force loop and the edge loop are calculated by adding the delays for the robot system and the corresponding delay for the sensor data. It was found to be 33.5 ms for the force loop and 43.8 ms for the edge loop. The experiments show that the largest contribution to the overall delay is due to the delay in the robot's real-time interface, the low-level control system, and the mechanical system. This is a delay which cannot be influenced by the user. The remaining delays are mainly due to synchronization of control loops with different cycle frequencies. Computation times in the different components and communication delays play a minor role in the system.

An unexpected delay of an additional cycle was found for a small number of measurements of the delay from the commanded robot motion until the movement. This delay is object of further investigation.

5.5 Control and Seam Quality

This section is an excerpt from the experiments which have been presented in the attached publication *"Real-Time System Integration in a Multi-Robot Sewing Cell"* [SWL12].

As described in Section 5.3.3, independent force and edge control is used to control the robots. Both robots work in this control mode independently of each other. However, the mechanical setup leads to dependencies due to friction and light-occlusion that may affect the edge sensor. The following experiment was conducted to determine whether the controllers work as intended during the sewing operation. In the experiments, two similarly shaped workpieces were set up manually in the sewing cell and the sewing operation was done automated using force and edge control. The force controller was set up to hold a desired force of 2 N in the workpiece.

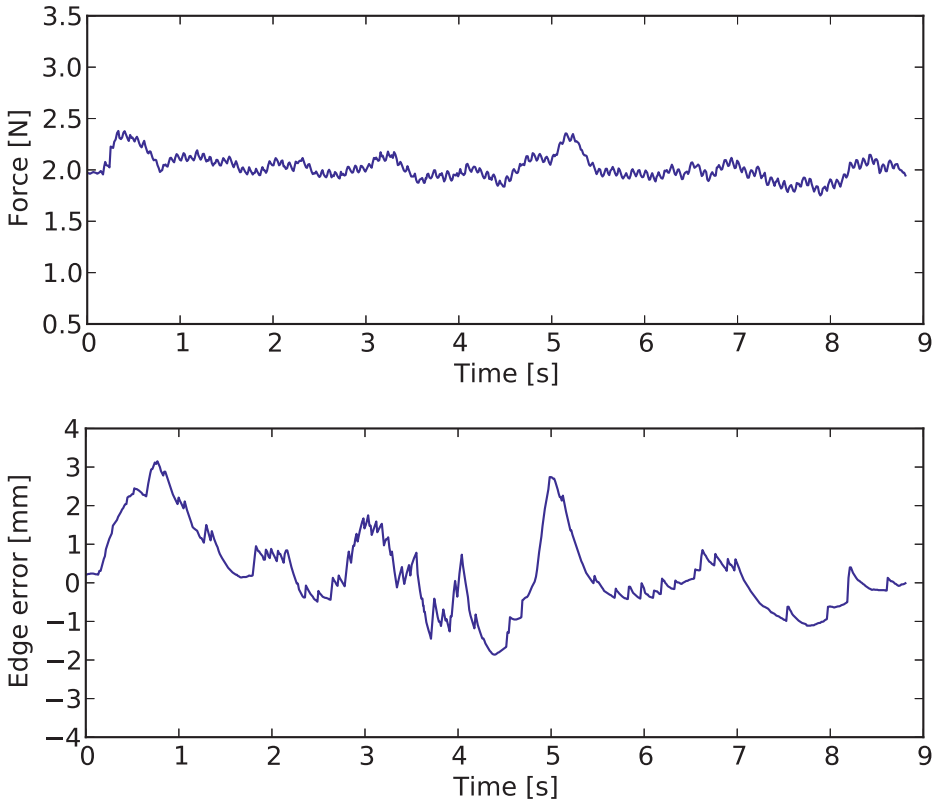


Figure 5.19: Sewing force value and edge error of the upper robot for a typical sewing process with two robots.

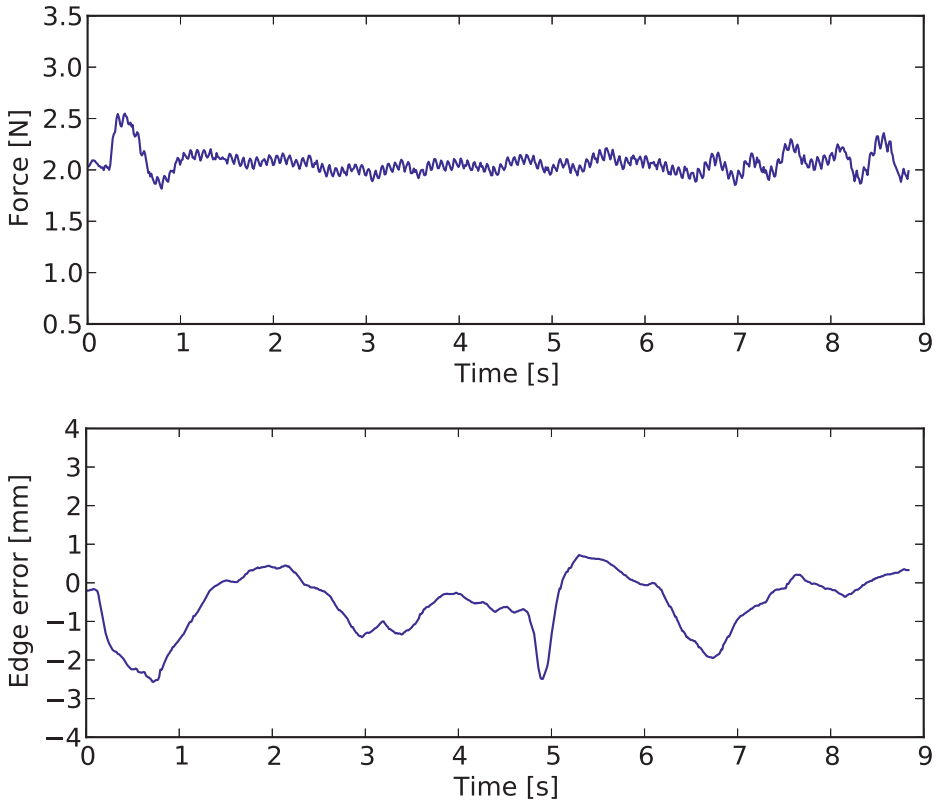


Figure 5.20: Sewing force value and edge error of the lower robot for a typical sewing process with two robots.

Figure 5.19 shows the force and edge measurements for the upper robot, while Figure 5.20 shows the same measurements for the lower robot. It can be seen that the force controller keeps the sewing force at the desired set point. Even though the force measurement is filtered using a sliding average filter with the window size of one stitch, the force variation due to the sewing foot movement can be seen as a small oscillation. The edge measurement represents the displacement of the part at the sensor plate. Since the sensor plate is located some centimeters in front of the needle and the edge correction is done by rotation, the real edge error on the workpiece is generally observed to be lower than the measured edge error. This holds for an ordinary sewing operation. Anyhow, the measured edge error at the sensor plate is below 3 mm, which was found to more than satisfactory for the sewing operation of these parts.

It can be seen that the edge error measurements are quite similar for the two parts, but with different sign. This is due to the mechanical construction in which the sensors are mounted with opposite orientation in respect to each other. The plot of the upper edge measurement shows more disturbances than the plot for the lower measurement. This can be explained by the mechanical system where the upper part is resting on the sensor plate which has an elastic mount allowing for oscillations induced by the feeding mechanism, while the lower part rests on the more stable sewing machine base.

Due to the distance between the edge sensor and the needle, as well as the complex mechanical behavior of the workpiece in the sewing cell, it is difficult to draw conclusions concerning the real seam quality only based on the measurements. Low error measurements indicate that the resulting seam likely is of a good quality, but a quality check of the sewn part is essential.

Figure 5.21 shows an example of a finished seam. Manual inspection shows that the seam is of a satisfactory quality, except for the offset at the corner which is about 10 mm for the depicted assembly. This error concerning the corner matching is caused by noninteracting control loops for the two workpieces. This challenge is further discussed in Section 5.6. It was observed that local peaks in the edge error measurements usually are not significantly visible on the seam. On the other hand, an uneven edge, for example due to markings cut into the edge, can result in false control responses since the markings are unmodeled. This is usually not a problem due to the damping behavior of the mechanical system.

Figure 5.21: The assembly of two parts, sewn in the experiment with two robots. The seam runs from left to right and is nicely placed at a nearly fixed distance from the edge. At the end of the seam an accumulated error of the feeding can be observed as a longitudinal displacement between the corners of approximately 10 mm.

5.6 Velocity Synchronization and Corner Matching

Preliminary work has been done to work on the challenge of matching the corners at the end of the sewing operation. This challenge arises from different feed velocities for the two parts due to independent control loops for the upper and the lower part. Experiments have shown that the feed speed is highly dependent on factors like sewing force and material thickness. Since the demonstrated system is designed to handle materials of different thicknesses and material characteristics, concepts have been designed to compensate for the differences in the sewing speed. Early experiments have shown that the feed speed can be influenced by temporal changes in the sewing force, cf. the attached publication *"Experiments towards Automated Sewing with a Multi-Robot System"* [SW12].

Another convenient way to influence the feed speed is through mechanical adjustment of the stitch length in the sewing machine. Two adjustment wheels on the front of the sewing machine allow for changing of the stitch length independently for the upper and the lower part. A mechanical servo-based system has been suggested to control the feed speeds in real-time. In combination with observation and estimation of the sewing force, such a servo-system could be used for corner matching. However, this system was put back to the benefit of other control methods that do not need me-

chanical adjustments on the sewing machine. This is because the use of the method would constrain the presented methods to sewing machine, which allow this special kind of stitch length control.

Another promising concept is based on synchronization of the two distances between the robot tools and the needle. In this method, the robot with the lower measured force is in the previously presented force control mode while the other robot is programmed to keep the same tool-to-needle distance as the force-controlled robot. The edge control is not influenced by the changed control method. A supervisor system is used to observe the force measurements of the two robots, and if needed to switch the leader and the follower.

The proposed method has the drawback of allowing larger sewing forces in the distance-controlled work piece than the desired set point of 2 N. However, preliminary experiments show that the sewing force does not exceed about 10 N, even for experiments with introduced stitch length differences of about 30 %. This rise of the sewing force is considered to be acceptable since visual inspection of the part does not reveal flaws in the seam quality. Notice that the set point of 2 N is no hard requirement, but rather is chosen based on experience. It is found to be a good compromise between a low sewing force not interfering with the feeding mechanism of the sewing machine and a force high enough to ensure that the workpiece responds on the robots edge controlling.

5.7 Overall Process

To include an automated sewing cell in a larger installation, mechanisms for material handling have to be included into the system.

An overall sewing operation can be divided into the following parts:

- Identification and location of the parts on a table or in a storage system
- Picking and possibly turning of the parts
- Pairing the parts for sewing
- Moving the parts into the sewing machine area and under the sewing foot
- The sewing operation
- Removing the part from the sewing machine and moving it out of the sewing cell

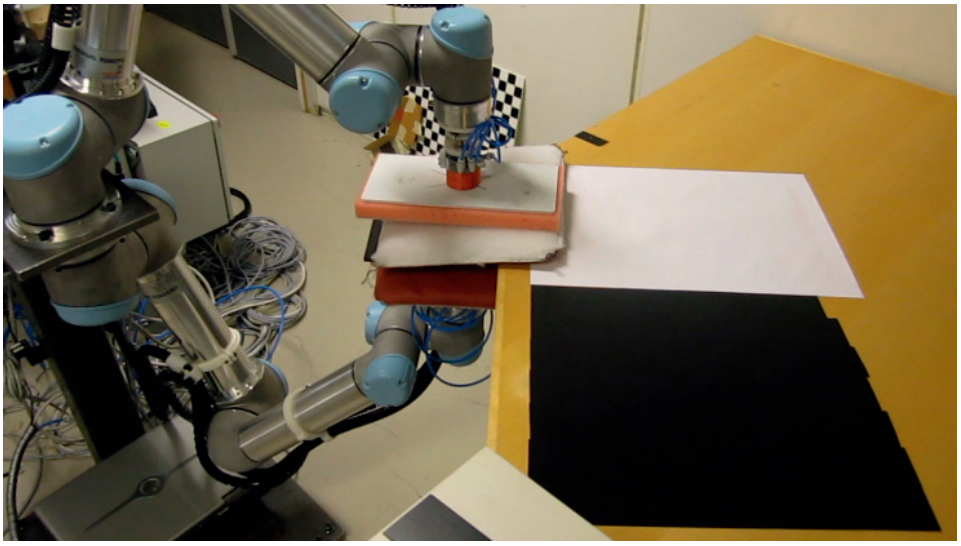


Figure 5.22: Pairing of two parts.

- Quality check

The industrial applicability and hence the overall process is an important part of the project. Even though the material handling was no direct part of the presented work, a series of experiments and concept studies were started due to the importance for the success of the sewing demonstrator. Figure 5.22 shows an experimental pairing operation based on visual detection of the workpieces and force control during the stacking operation.

5.8 Conclusions and Future Work

The presented work is a major step towards the automation of the sewing of 3D-shaped assemblies. A sewing demonstrator was presented that is able to sew two parts with slightly different shapes. The concept of independent force and position control has been demonstrated for a two-robot solution. The feasibility has been demonstrated for cases of nearly straight seams. Figure 5.23 shows an assembly of four parts that has been sewn in the demonstrator cell. The handling of the material was done manually while the sewing operation was fully automated. The assembly has been inspected manually and found to be of acceptable quality concerning the seam, but with some flaws at the end of the seam.

New challenges have been identified and investigated based on the used

Figure 5.23: An assembly of four parts that have been sewn in the sewing cell. The handling before and after the sewing operation has been done manually while the sewing operation was fully automated.

strategies, for example different feeding velocities of the two parts and resulting issues with the corner matching. Promising solutions have been designed and preliminary experiments have been conducted.

Preliminary work has been done towards corner matching and a concept has been presented to accomplish this task. Future work includes the further implementation of the presented method as well as experiments with new tools that make it easier to combine the different stages of the sewing operation. A promising tool concept is based on grippers that can shift the grip during the sewing operation.

Regarding the control of the sewing speed of both parts, it is imaginable to work on a control system that is based on mechanical adjustment of the stitch length for both parts in the sewing machine.

Further improvements of the stability of the system in regard to different material types and material thicknesses can be expected from the integration of the mentioned stereo camera system, replacing the current sensor plate.

Probably the most important task for future work is the integration of the sewing operation in the overall process including the material handling before and after the sewing. This is especially important with respect to industrialization of the presented demonstrator cell.

Since the presented work was designed to be as general as possible re-

garding material shape and characteristics, it is conceivable to work on a demonstrator that is more adapted to a special case in order to gain industrial stability and to accelerate the transition into an industrial system rather than a proof of concept.

Chapter 6

Included Publications

This chapter is a compilation of pre-prints of papers prepared during the period of the PhD scholarship. The compilation is to be considered the main contribution of this thesis. Each paper is presented in its own section which opens with the bibliographic information for the contained paper and a declaration of contributions.

6.1 Open Real-Time Robot Controller Framework

M. Lind, J. Schrimpf, and T. Ulleberg, *Open real-time robot controller framework*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010

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.

Comments

- In early publications, the term *motion control* was used as synonym for *trajectory generation*. We were enlightened that *trajectory generation* was the correct term. To clarify the difference between the

low-level controller system and the trajectory generation system, the term *trajectory generation* was used in later publications.

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.

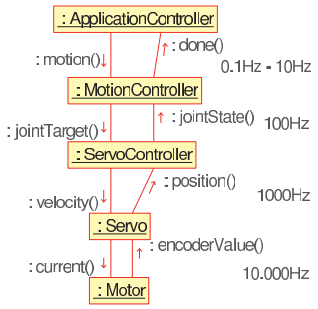


Figure 1: Simplified view of the communication within an industrial robot controller.

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.

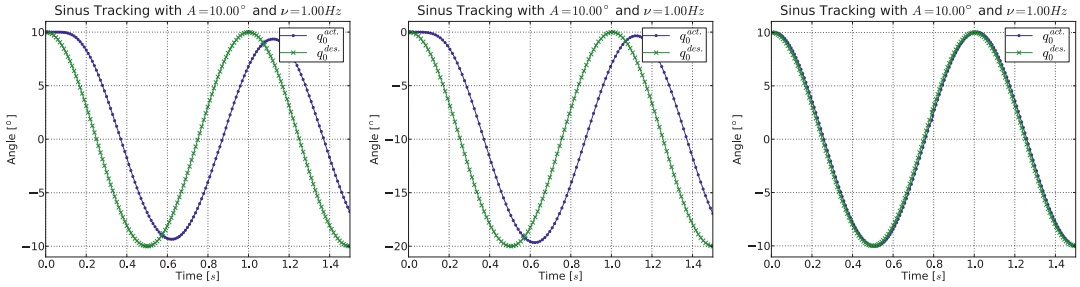
Figure 3 shows plots of step responses from the experiment controllers. For each robot is seen an individual size of the step, which has been experimentally maximized. The maximization is done to ensure as fast and strong a response as possible. Since the KUKA and NACHI servo controller performs filtering 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

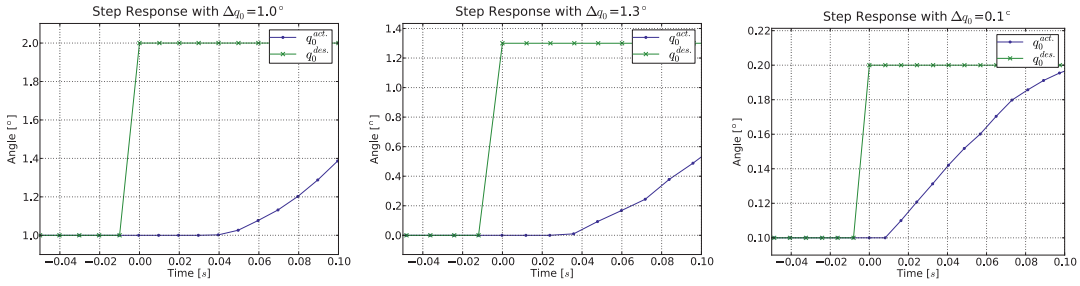
By analysis of Figures 2 and 3 some estimates of the sine- and step-responses for the different robot controllers can be extracted.

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 Uni-

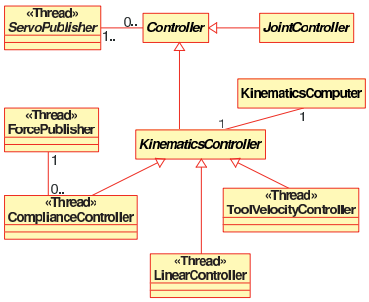


Figure 4: Class diagram of the central classes in the PyMoCo design.

versal Robots servo controller is also extended with actual joint velocities.

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

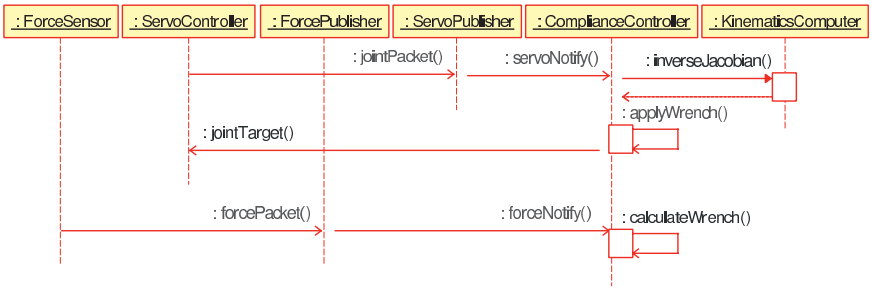


Figure 5: Sequence diagram illustrating the two, asynchronous, cyclic interactions in a single-sensor servoing controller.

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.

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 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., Bolmsjo, G., Brogardh, 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 1999 IEEE International Conference on Control Applications, 2:1642–1647.

[10] Dallefrate, D., Colombo, D., Tosatti, L., 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, Proceedings of the 3rd CIRP Conference on Assembly Technologies and Systems, To be published.

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

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*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010

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.

Comments

- In early publications, the term *motion control* was used as synonym for *trajectory generation*. We were enlightened that *trajectory generation* was the correct term. To clarify the difference between the low-level controller system and the trajectory generation system, the term *trajectory generation* was used in later publications.
- The correct Equation 7 is $\begin{bmatrix} e_{\theta,x} & e_{\theta,y} & e_{\theta,z} \end{bmatrix}^T = \begin{bmatrix} n_y & -n_x & 0 \end{bmatrix}^T$, with the points $\mathbf{p}_1$ to $\mathbf{p}_4$ layed out such that $\mathbf{n}$ is an outward normal.

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-

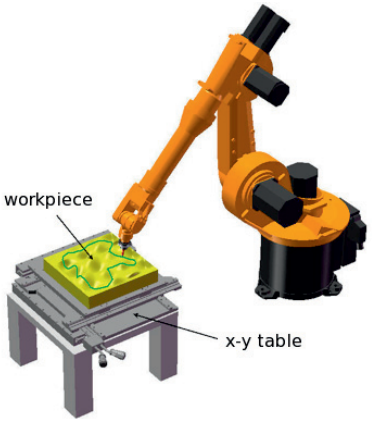


Figure 1: The test platform consists of a NACHI SC15F with a real time sensor interface and a hilly shaped workpiece.

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:

$$\mathbf{e} = \begin{bmatrix} e_x & e_y & e_z & e_{\theta,x} & e_{\theta,y} & e_{\theta,z} \end{bmatrix}^T , \quad (1)$$

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

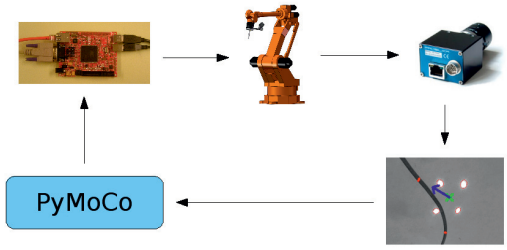


Figure 2: The control structure - from upper left: real-time interface, robot, camera, vision system, PyMoCo (Python motion controller).

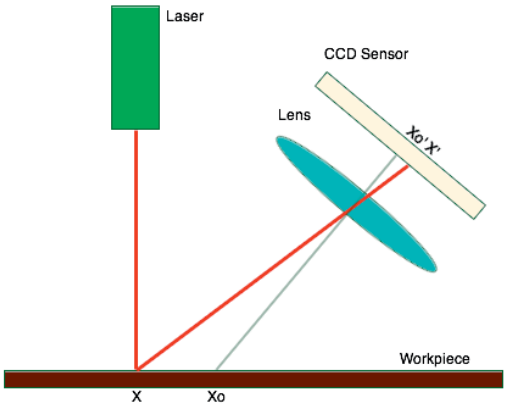


Figure 3: A triangulation-based distance sensor with a CCD-array. When the sensor is moved in vertical direction, the laser point is projected to another position on the CCD array.

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.

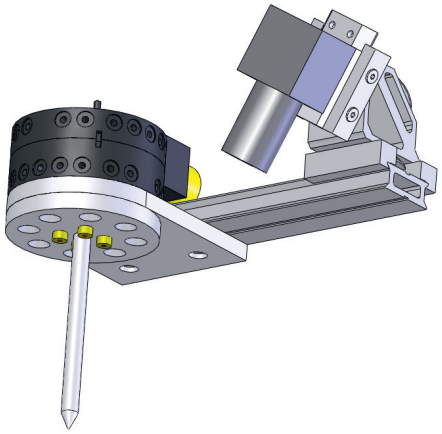


Figure 4: The sensor system attached to a tool changing system. The stick is a placeholder for a real tool.

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:

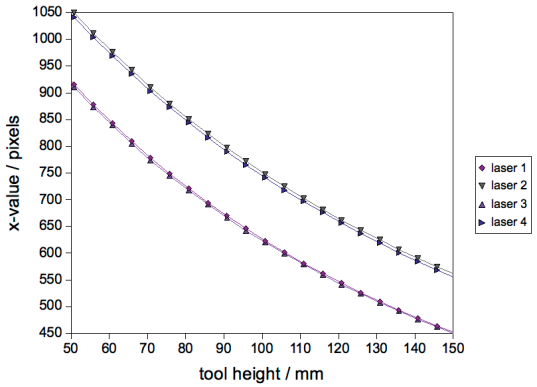


Figure 5: In the vision system the x-values of the laser point projection on the picture are converted to heights. This figure shows the dependency between the x-pixel and the height measurement for each laser.

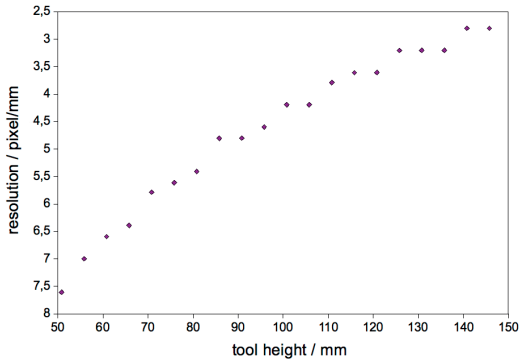


Figure 6: The height measurement resolution is dependent on the actual distance between the camera and the workpiece.

$$\mathbf{p_i} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T \quad \text{with} \quad z_i = d_{zi} \ , \tag{5}$$

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:

$$\begin{bmatrix} e_{\theta,x} & e_{\theta,y} & e_{\theta,z} \end{bmatrix}^T = \begin{bmatrix} n_x & n_y & 0 \end{bmatrix}^T \ . \tag{7}$$

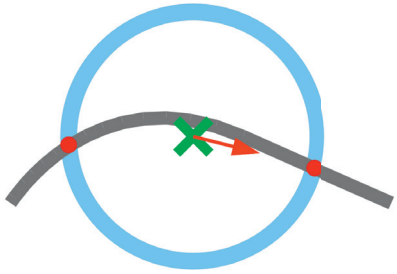


Figure 7: In the line-of-sight algorithm a circle is drawn around the tool center point, and the intersection points between the circle and the marked path are taken as possible way-points. Our algorithm choses the way-point with the smallest angle deviation from the actual movement direction.

$e_{\theta,z}$ 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 $\mathbf{e}$-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 contact-less 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/.

6.3 Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking

J. Schrimpf, M. Lind, and G. Mathisen, *Time-analysis of a real-time sensor-servoing system using line-of-sight path tracking*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 2861 –2866

Declaration of co-authorship

The demonstrator setup was designed and implemented by Johannes Schrimpf. Morten Lind contributed in occasional discussions. The real-time analysis was conducted by Johannes Schrimpf. The simulations of the line-of-sight algorithm were done by Johannes Schrimpf. The real-time experiments were conducted in close cooperation between Johannes Schrimpf and Morten Lind.

The paper was written, prepared and submitted by Johannes Schrimpf. Geir Mathisen was involved in discussion about the structure and in the mathematical part of the paper. Morten Lind and Geir Mathisen contributed in the review of the final version.

Comments

- In early publications, the term *motion control* was used as synonym for *trajectory generation*. We were enlightened that *trajectory generation* was the correct term. To clarify the difference between the low-level controller system and the trajectory generation system, the term *trajectory generation* was used in later publications.

Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking

Johannes Schrimpf, Morten Lind, Geir Mathisen

Abstract— **This paper presents a study of the real-time control conditions for a robotic system with visual servo control. The system is based on an industrial manipulator with a modified controller allowing real-time joint-level control. The work is particularly concerned with delays and path deviations. The focus is on the Line-of-Sight based path tracking controller. The paper describes an analysis of the different delays in the sensor-robot system and a model is presented based on the overall delay. Further the stability of the modeled system is analyzed in respect to a specific control case. The correctness of the estimated system delay is indicated by comparison of simulated and experimental results.**

I. INTRODUCTION

A challenge in sensor-integrated, real-time controlled robot systems is to achieve acceptable performance in speed and accuracy, since such systems are burdened with undesired system delays [1]. The total system delay has contributions from different sources, some delays are tunable, some sources are replaceable and some delays or sources are strongly connected to the application and are thus inherent. Due to stability, performance and quality requirements, the existing system delays should be considered and allowed for in the control algorithms.

A test platform for the demonstration and evaluation of different sensor servoing techniques was built by the authors and presented in [2]. It focuses on cases where the robot tool has to follow a curve at a given distance from the surface. A tool orientation controller was introduced to the system to enable the tracking of 3D-shaped objects and surfaces. An important design criterion was to build a flexible system which is free from knowledge of the workpiece.

Another design criterion was the use of consumer grade sensor hardware, ensuring affordability and easy availability. A consequence of this is that the setup is applicable for even small research projects, and is thus attractive to commission in non-critical production installations.

In the present version of the system a eye-in-hand configuration is used, based on a triangulation distance sensor, related to the Opto3D sensor [3]. Caccia, [4], used a similar sensor to detect surface normals and distances for the navigation of underwater vehicles. A related sensor servoing system was built by Hosoda et al. [5]. This system navigates visually guided on unknown surfaces by utilizing an external camera and a force sensor at the tool. Another relevant system was built by Zhang et al. [6] who proposed a system which allows for automatic robot program generation. The method utilizes an eye-in-hand vision system for path tracking, combined with a force sensor for orientation detection. For orientation, the tool moved in a zigzag path on the surface to detect the local geometry. The Line-of-Sight (LOS) path-tracking algorithm is employed on the test platform. This is known from the navigation of ships [7], [8].

The main contribution of this paper is the derivation of a simplified linear model of the combined experimental sensor and control system. The analysis of eigenvalues of the model yields a computational method for determining stability; in terms of limits on characteristic parameters; and with the total system delay as input. Estimation and measurement of contributions to the total delay in the sensor and control loop; provides a basis for evaluating the usability of different sensor and robot systems for given application requirements.

The presented experimental system, with different types of hardware, has already been planned for use in a robotized sewing application. Other applications in manufacturing systems such as welding or grinding are conceivable.

II. SENSOR SERVOING TEST PLATFORM

A test platform was built to study real-time sensor servoing. A detailed overview of the system and its controllers was presented in [2]. Since the following sections depend on understanding the different parts of the system, a summary is given in this section.

The test platform is built around a NACHI SC15F 6-axis industrial manipulator. A real-time interface was built into the main controller to make it possible to connect an external motion controller to the robot system.

A sensor system is attached to the robot tool flange. The sensor system is made up of an Ethernet camera and four laser pointers, arranged at an angle to the camera, allowing distance measurements by triangulation. A vision controller system was designed in C++ using OpenCV and included controllers for the tool distance, the tool orientation and a Line-of-Sight target generation algorithm.

The test platform was used to demonstrate a case where the tool follows an optical marked line on the workpiece surface at a given distance, and with the tool aligned perpendicularly to the workpiece surface. The system is depicted in Fig. 1.

The control loop consists of the triangulation-based sensor system, the vision system, the controller system, and the robot, including the real-time interface and the low-level

J. Schrimpf and G. Mathisen are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: johannes.schrimpf@itk.ntnu.no.

M. Lind is with the Department of Production and Quality Engineering, Norwegian University of Science and Technology, Trondheim, Norway.

G. Mathisen is also with SINTEF ICT, Trondheim, Norway.

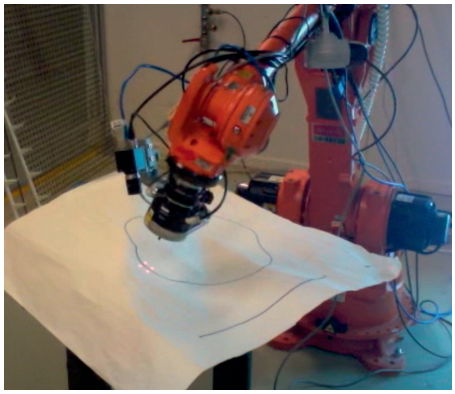


Fig. 1. The test platform consisting of a six-axis manipulator with sensor system attached to the tool, and a workpiece attached to a x-y table.

controller, as shown in Fig. 2. The parts are described in the following subsections.

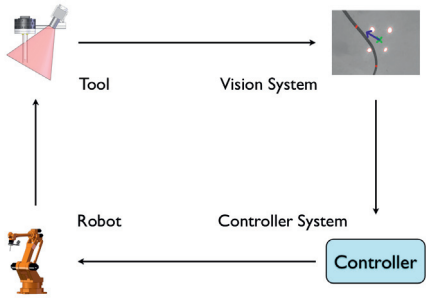


Fig. 2. The control system consists of four main parts: The tool including an Ethernet camera and four laser pointers, the vision system consisting of the height and orientation detection as well as the Line-of-Sight waypoint algorithm, the controller system including the Python Motion Controller PyMoCo, and the NACHI SC15F 6-axis industrial manipulator with the real-time interface and the low-level controller.

A. Tool Distance and Orientation Detection

The basic requirement for the control system is to keep the tool at a defined distance to the tracking point, and with a defined orientation to the surface vicinity of the tracking point. Since the shape of the workpiece is unknown, a sensor system was built to provide distance measurements around the tool center point. The distance is measured by laser triangulation. An Ethernet camera and four lasers are mounted on the tool-holder to measure the distances.

The camera used in the sensor system is a GC1350 Gigabit Ethernet Camera with a maximal resolution of 1360x1024 and a frame rate of 20 fps at this resolution. It is mounted on the tool flange, together with four low-cost laser LEDs.

To detect the orientation of the surface and the distance between tool and workpiece, the vicinity of the projection of the tool center point was assumed to be approximately flat. The tool-distance is defined as the average distance of the four separate distance measurements. The 3D laser points in the tool coordinate system are derived from the four distance measurements and the surface normal is computed by the cross product of two vectors between the laser points.

B. Line-of-Sight Path Following

A Line-of-Sight (LOS) guidance algorithm was used as the target generation algorithm. The LOS algorithm is related to the approaches for marine vessel navigation mentioned in [7] and [8]. This is a waypoint-based navigation algorithm. The vessel follows the course in the direction of the actual waypoint on the desired path until it enters a circle-of-acceptance around the waypoint. Then a new waypoint on the desired path is chosen, which is located in a predefined look-ahead distance on the path. This distance is called the Line-of-Sight radius. The heading of the vessel is then derived as follows:

$$\Psi_{LOS} = \operatorname{atan2}\left(y_k - y, x_k - x\right) \ , \tag{1}$$

where (x_k, y_k) are the coordinates of the waypoint and (x, y) are the coordinates of the vessel.

In the demonstrator system, a sliding waypoint is used, which is updated every time new sensor data is available. To chose a new waypoint, a picture is taken of the area around the projection of the tool center point on the surface and the intersections between a circle around this point and the marked line are found. These points are possible waypoints. The actual direction is determined by choosing the waypoint which gives the lowest deviation from the actual direction of motion of the tool; see Fig. 3.

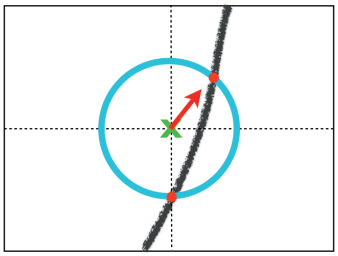


Fig. 3. The Line-of-Sight algorithm. The cross marks the projection of the tool center point on the surface. The arrow points in the direction of movement. The blue circle is drawn around the tool center point with a defined radius. The red points are the possible waypoints, from which the waypoint is chosen which gives the lowest derivation from the actual heading.

It is important to mention that the line-of-sight radius is a central design criterion of the system, since it influences both the robustness and the accuracy of the path tracking; confer Sect. IV.

C. Application Controller

The outputs from the vision algorithm are a vector including deviations in the tool orientation and tool height from the desired tool position relative to the surface, and the direction towards the next waypoint. In the application controller these values are converted into a spatial velocity vector, which

leads the robot tool towards the desired pose. A P-controller is used for the height and orientation control. However, this will be replaced by a more advanced controller in the next version of the demonstrator.

The velocity in the x-y-plane is derived from the movement speed v and the LOS angle Ψ_{LOS}:

$$\begin{bmatrix} v_x & v_y \end{bmatrix}^T = \begin{bmatrix} v \cdot \cos(\Psi_{LOS}) & v \cdot \sin(\Psi_{LOS}) \end{bmatrix}^T \quad (2)$$

The velocity vector is passed to the Tool Velocity Motion Controller.

D. Motion Controller

The Python Motion Controller framework, PyMoCo [1], is used as the motion controller for the robot system. It provides a tool velocity controller, based on an inverse Jacobian controller. This controls the joint position vector for the low-level controller leading to the desired spatial velocity of the tool. It is coupled to the real-time robot control interface via a UDP Ethernet connection.

E. Real-Time Interface and Robot

The real-time interface used to control the NACHI SC15F 6-axis industrial manipulator was created by installing an embedded single board computer into the original controller. This intercepted the communication between the motion controller and the low-level controller. It offers an interface for external motion controllers, connected via Ethernet, to send commanded joint angles and read the current joint angles of the manipulator. The internal communication frequency between the motion and the low-level controller in the NACHI controller is kept at the default value of 100 Hz. The interface offers only position control of the joints, using the original low-level controller.

III. REAL-TIME ANALYSIS OF THE TEST PLATFORM

The overall performance of the complete system is heavily dependent on the transport delays and computation times in the system, as well as on the cycle time in the sensor system. This section describes and estimates the delays in the different subsystems. To ensure stability of the system according to the calculation in the following section, the delay is estimated in a conservative way from worst case delays of the individual subsystems. The subsystems are depicted in Fig. 4.

Fig. 4. The data flow in the system.

A. Camera System

A Prosilica GC1350 Gigabit Ethernet Camera is used for image capturing. The delay in the sensor system consists basically of the time for the camera to capture and prepare the image and the transfer time. The transfer time from the camera to the PC was measured to be 16 ms, while the time between two successive pictures is 50 ms, based on the frame rate of 20 fps. This leads us of a total dead-time of 66 ms in the sensor system. Since the following delays in the sensor system are less than 50 ms, the frame-rate of 20 fps is the limiting value for the cycle time of the sensor system. Time measurements in the vision system, which were done by adding timestamps to the incoming data, verify that this update frequency is achieved.

B. Vision Software

The worst case image processing time in the vision program was measured to be 30 ms. Both the laser tracker and the line tracking algorithm need about half this time to process the image data.

C. Motion Controller

In the tool velocity controller, the sensor data is processed after the arrival of a new joint position vector from the low-level controller. Since the cycle time in the real-time communication to the low-level controller is 10 ms, this value is taken as the worst case value for the delay between the arrival of the sensor data and the processing time in the motion controller. The calculation of the motion controller takes an average time of around 3 ms, before the updated joint values are sent to the real-time interface. The transfer time between the PC and the real-time interface is around 2 ms.

D. Real-Time Interface

The timing in the real-time interface is specified by communication between the main controller and the low-level controller. In normal operation, the data from the motion controller is sent to the low-level controller 3 ms after it reaches the real-time interface. This delay is due to synchronization with the native control flow.

E. Low-Level Controller

The original NACHI low-level controller (LLC) is used and directly addressed by the real-time interface board. Since the input for the low-level controller are joint positions, or actually joint encoder values for each joint, filtering and low-level trajectory interpolation are performed in the low-level controller unit. Since the behavior of the LLC-robot system is not known, and there is no possibility of interfacing the low-level controller at torque level, the robot system can be seen as a black box. To describe the characteristics of the low-level controller, the *tracking delay* is defined as the time that the actual motion is trailing the commanded motion.

The behavior of the low-level controller of the NACHI SC15F was analyzed in [1]. In the presented work, for computational purposes the proprietary low-level controller

Part	Delay
Camera	66 ms
Vision Algorithms	30 ms
Motion Controller	15 ms
Real-Time Interface	3 ms
Servo Controller	120 ms
Sum	234 ms

TABLE I
DELAYS IN THE SYSTEM.

is approximated as a delay element. The tracking delay in [1] was found constant at 120 ms within each measurement series, and across all measurement series. The main reason for this delay is filtering in the low-level controller. It was observed that the delay is independent of the velocity.

Industrial low-level controllers typically use a cascade control structure with control parameters for the different control loops [9]. At present, the real-time interface is not able to access these control parameters.

F. Overview

An overview of the different delays in the system is shown in Table I.

The total system delay is around 234 ms, where the contribution from the sensor system is found to be 96 ms and the contribution from the robot control system is 138 ms. As mentioned before, these are worst case delays. It was observed that the average delay for the different parts are some milliseconds below the worst case delay. However, the worst case delays are used in the analysis.

The cycle time of the sensor system was found to be 50 ms.

G. Discussion

Since fast response times are very important in sensor servoing and long delays and large cycle times can render the system unstable, it is desirable to reduce the delays in the individual subsystems. Different solutions are possible for these subsystems. The sensor system, for example, is very dependent on the hardware; e.g. the link between camera and PC may be a bottleneck.

The vision algorithm is a part of the system which is not yet optimized, so it would be relatively easy to gain time savings in this part, yet the calculations in the vision system make up just 13% of the total system delay. The obvious improvements are to use a faster PC, to implement more time-efficient image processing algorithms, or to distribute the processing to several CPUs, alternatively use a fast FPGA. All three improvement techniques are independent and any subset could be realized.

The greatest delay is due to motion filtering in the low-level controller. This is tied to the chosen robot platform. Robots are available on the market, which use other control strategies for real-time control which reduces the delay in the filter. For example, the low-level controller in the Universal Robots UR-6-85-5-A requires commanding by a triple of joint positions, velocities and accelerations. This relieves the burden of filtering and trajectory planning in the low-level controller. The external motion controller has to take over the calculation of these values, which is not hard to implement due to the desired motion scheme being known.

IV. LINE-OF-SIGHT ALGORITHM

In some applications, for example in welding and sewing, the path may not be complete unknown. Quite frequently the path will be known to be a sequence of straight line segments.

Hence the special simple case of following a straight line is modeled and analyzed. For this case the equations for motion in the x- and y-directions are given from the following delay differential equations 4 and 3. The line is defined to lie on the x-axis of the coordinate system, so p_y describes displacement from the given path, see Fig. 5. v is the commanded speed and τ is the system delay.

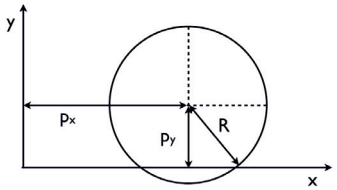


Fig. 5. Sketch of the Line-of-Sight algorithm for the case that the system has to follow the x-axis. The controlled tool point, which is located at (p_x, p_y) has to move to the forward intersection point between the tool-centered circle with the radius R and the x-axis.

$$\dot{p}_y(t) = -v\,\frac{p_y(t-\tau)}{R} \tag{3}$$

$$\dot{p}_x(t) = \sqrt{v^2 - \dot{p}_y^2(t)} = v\,\sqrt{1 - \frac{p_y^2(t-\tau)}{R^2}} \tag{4}$$

It is obvious from the equations that the motion is determined by three parameters: the speed v, the Line-of-Sight radius R and the delay τ. The translational symmetry of the problem along the line to track (the x-direction) is manifested in complete independence of p_x in the equations. It is further interesting that for the equation of motion for the y-direction, the functional dependence of parameters v and R is only through the ratio v/R.

Though the speed, v, is always the same, the velocity in the x-direction is dependent on the error in the y-direction. The maximal velocity in the x-direction is reached when the actual point lies on the line.

Since the demonstrator system is time-discrete due to the cycle time of the sensor system, the system is rewritten into a time-discrete form. The longest cyclic period in the system is used as discretization step t_s. For the presented system, this is the measured cycle time from the sensor system, which is 50 ms.

d is defined as the number of discretization steps t_s, forming the complete system delay τ_d. τ_d is a multiple of t_s which is close to τ. For stability analysis $\tau_d > \tau$ should be

chosen to ensure that the stability regions also hold for the simplified system.

$$p_y[k+1] = p_y[k] - v\ t_s\ \frac{p_y[k-d]}{R} \tag{5}$$

$$p_x[k+1] = p_x[k] + v\ t_s\ \sqrt{1 - \frac{p_y^2[k-d]}{R^2}} \tag{6}$$

Since the movement in the x-direction has no influence on the stability of the system, only the motion in the y-direction is analyzed. This is done by analyzing the linear system:

$$x[k+1] = A_d\ x[k] \tag{7}$$

with

$$x[k] = \begin{pmatrix} p_y[k] \\ p_y[k-1] \\ \vdots \\ p_y[k-d] \end{pmatrix},\ A_d = \left(\begin{array}{ccc|c} 1 & 0 & 0 & -a \\ \hline 1 & & 0 & 0 \\ & \ddots & & \vdots \\ 0 & & 1 & 0 \end{array}\right) \tag{8}$$

where a is

$$a = \frac{v\ t_s}{R} \tag{9}$$

One can see that the model has $d+1$ states. From the model it is possible to derive the stability criteria (all eigenvalues within the unit circle), which will be dependent on the parameter a. Fig. 6 depicts the regions of stability with respect to the above-named parameter a and the delay.

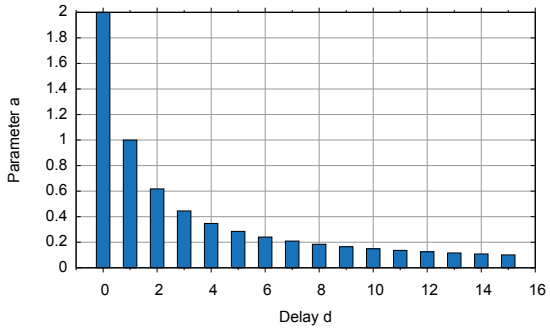


Fig. 6. The stability regions for parameter a plotted against the delay d.

The figure clearly illustrates that the stable area for a decreases quickly with an increasing number of delayed steps d. Therefore it is important to identify and minimize the delays in the system and try to speed up the calculations.

Since a is also dependent on the cycle time, and thereby the chosen t_s, the region of stability can be increased by choosing a sensor system with a smaller cycle time. If the cycle time gets lower than the cycle time of the robot system, the control cycle time of the robot is the limiting factor in the system. Since both t_s and d are given by the equipment used, the parameters left variable in the system are the velocity and the LOS radius. For many applications, the desired tool velocity is given by the process, and therefore the radius has to be adjusted to fit the system requirements.

For many systems it is desirable to choose a radius that prevents the system from oscillating, but at the same time approaches the desired path as fast as possible. It has also to be taken into account that for applications with other paths than a straight line the system may overshoot even if it acts overcritically damped in the case of a line. If additionally external disturbances have to be taken into account it can be necessary to increase the radius even more. This leads to a more robust system that is less accurate and less sensitive.

The influence of different LOS radii on the system with constant velocity is shown in the simulation in Fig. 7. The figure presents the above analyzed case of a line. The cycle time used in the simulation is 50 ms and the delay is set to 250 ms, which is the nearest multiple of the cycle time, that fits our estimated delay. The velocity is set to $0.05\ \frac{\mathrm{m}}{\mathrm{s}}$. The initial displacement is set to 0.01 m.

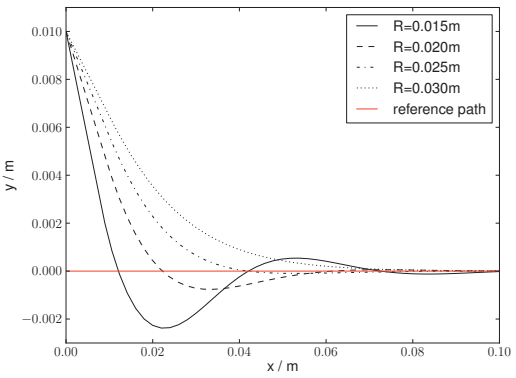


Fig. 7. Step response as a function of the Line-of-Sight radius.

V. EXPERIMENTS

To indicate the correctness of the estimated delays in the system, a test case was defined where the robot has to follow a 90° corner in the path. A simple geometric computation was used to generate trajectories. Simulations for different speeds were done with two different delays: 200 ms and 250 ms. These delays were chosen since they fit best with the expected delay found in Sect. III. The results were compared with the behavior of the real robot-sensor system. From the history of experimentation on the path tracking setup, a radius of 25 mm has been found to be a comfortable compromise between precision and robustness for tool speeds up to $100\ \frac{\mathrm{mm}}{\mathrm{s}}$. Fig. 8 shows the results for a velocity of $40\ \frac{\mathrm{mm}}{\mathrm{s}}$, which was found by simulation to lead to a system with roughly critical damped behavior. Fig. 9 illustrates the result for a test with a tool speed of $100\ \frac{\mathrm{mm}}{\mathrm{s}}$, which was the maximum velocity used in the experiments.

In Fig. 8 a small deviation from the reference line is visible on the negative x-axis. The real coordinate system is fitted to the simulation coordinate system with nodes outside the corner region. Since there are small fluctuations in the robot

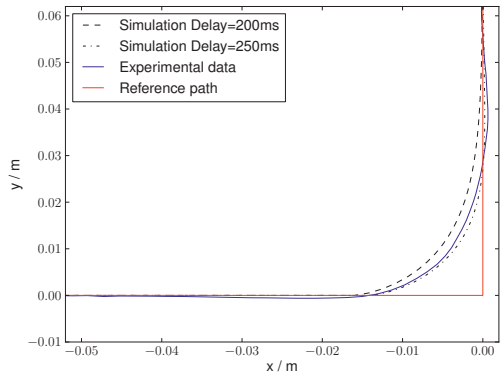


Fig. 8. Simulation and experiment for a tool velocity of 40 mm/s.

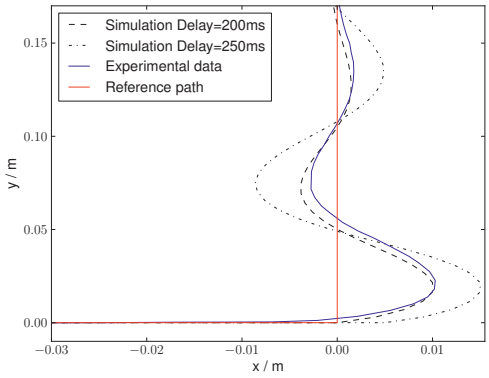


Fig. 9. Simulation and experiment for a tool velocity of 100 mm/s.

movement on the axis, there is a small offset between the theoretical movement and the measured movement. Other uncertainties in the fitting occur if the system still oscillates in the regions of the node on the y-axis. This behavior can be seen in Fig. 9, in which the measured curve seems to be offset slightly in the x-direction. The deviation during the transient oscillation in both figures can be explained by the unknown behavior of the motion filtering and therefore uncertainties in the modeling. However, the figures show that the measured behavior of the system fits fairly well with the expected behavior.

VI. CONCLUSION

A real-time sensor servoing system, using Line-of-Sight trajectory generation was analyzed. A test case was defined for straight line tracking and a theoretical model, based on delay differential equations, was developed to analyze stability criteria for the case of approaching a straight line. The worst case delay in the real system was investigated by analyzing different part systems and the result was used in the model as an upper boundary.

Parameters in the Line-of-Sight trajectory generation were identified. Especially the influence of the Line-of-Sight radius for a given tool velocity was studied. The influence of the delays and the parameters on the stability of the system was analyzed.

Experiments were done to verify that the delay found for the parts of the robot control system and the theoretical model fit together with the behavior of the overall system. It was found that the results of the experiments confirm the theory. The methods presented in this paper can be generalized to analyze similar real-time sensor systems, such as systems for sewing or welding.

A major part of the overall delay is caused by the low-level controller. To improve the performance of the system, it is planned to use a Universal Robots UR-6-85-5-A which offers a low-level controller with smaller tracking delay. Further, a camera with a higher frame rate and a more efficient implementation of the vision algorithm can lead to a performance gain. By controlling the LOS radius in real-time, dependent on the deviation from the path, a larger stability region and greater robustness can be achieved while keeping the accuracy when the deviation is small. This improvement is planned to be included into the next version of the test platform.

VII. ACKNOWLEDGMENTS

The authors wish to thank the SFI Norman programme for financial support and supplying equipment to the project, in which this work was a part.

Also thanks to Ivar Halvorsen, SINTEF ICT, Department of Applied Cybernetics, for given valuable inputs on the discrete system modeling.

REFERENCES

[1] M. Lind, J. Schrimpf, and T. Ulleberg, "Open external real-time servo-level robot control framework," in *3rd CIRP Conference on Assembly Technologies and Systems (CATS 2010)*, 2010.

[2] 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 *3rd CIRP Conference on Assembly Technologies and Systems (CATS 2010)*, pp. 19 – 23, 2010.

[3] L. Marques, U. Nunes, and A. de Almeida, "A new 3d optical triangulation sensor for robotics," in *Advanced Motion Control, 1998. AMC '98-Coimbra., 1998 5th International Workshop on*, pp. 512–517, Jun-1 Jul 1998.

[4] M. Caccia, "Laser-triangulation optical-correlation sensor for rov slow motion estimation," *Oceanic Engineering, IEEE Journal of*, vol. 31, pp. 711 –727, July 2006.

[5] K. Hosoda, K. Igarashi, and M. Asada, "Adaptive hybrid control for visual and force servoing in an unknown environment," *Robotics & Automation Magazine, IEEE*, vol. 5, pp. 39 – 43, December 1998.

[6] H. Zhang, H. Chen, and N. Xi, "Automated robot programming based on sensor fusion," *Industrial Robot: An International Journal*, vol. 33, no. 6, pp. 451 – 459, 2006.

[7] T. I. Fossen, *Marine Control Systems - Guidance, Navigation and Control of Ships, Rigs, and Underwater Vehicles*. Trondheim: Marine Cybernetics, 2002.

[8] E. Børhaug, *Nonlinear Control and Synchronization of Mechanical Systems*. PhD thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2008.

[9] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: modelling, planning and control*. Springer Verlag, 2009.

6.4 Experiments towards Automated Sewing with a Multi-Robot System

J. Schrimpf and L. E. Wetterwald, *Experiments towards automated sewing with a multi-robot system*, International Conference on Robotics and Automation (ICRA), 2012

Declaration of co-authorship

The sewing cell was designed and implemented by Johannes Schrimpf in close cooperations with Lars Erik Wetterwald and Morten Lind (not coauthor).

The control concept, the implementation and the block diagrams were designed by Johannes Schrimpf.

The static force experiment and the sewing force experiment were designed and conducted in close cooperation between Johannes Schrimpf and Lars Erik Wetterwald. The pull/release method was designed by Johannes Schrimpf and the experiment was done in cooperation with Lars Erik Wetterwald. The data processing was done by Lars Erik Wetterwald. The data plots were created by Johannes Schrimpf.

The paper was single-handed written by Johannes Schrimpf. Lars Erik Wetterwald contributed in occasional discussions and in reviewing the final versin of the paper.

Experiments towards Automated Sewing with a Multi-Robot System

Johannes Schrimpf, Lars Erik Wetterwald

Abstract— **In this paper a concept for automated multi-robot-aided sewing is presented. The objective of the work is to demonstrate automatic sewing of 3D-shaped covers for recliners, by assembling two different hide parts with different shapes, using two robots to align the parts during sewing. The system consists of an industrial sewing machine and two real-time controlled Universal Robots 6-axis industrial manipulators. A force feedback system combined with optical edge sensors is evaluated for the control of the sewing process. The force sensors are used to synchronize the velocity and feed rate between the robots and the sewing machine. A test cell was built to determine the feasibility of the force feedback control and velocity synchronization. Experiments are presented which investigate the ability of the robot to feed a hide part into the sewing machine using a force sensor and different strategies for velocity synchronization.**

I. Introduction

Because of the labor intensive nature of sewing operations, the industry in high cost counties like Norway has moved most of their sewing processes to low cost countries. There are some exceptions that are usually driven by the need for short time to market and high customer flexibility, combined with logistical advantages. However, the increasing labor costs in high cost countries is challenging for the whole value chain even if the majority of the production process is highly automated and cost effective compared to low cost countries. Hence, failing to increase productivity and capacity in the non-automated processes like sewing operations will challenge more than just the non-automated processes, and this has motivated the development of new advanced automation such as automated sewing.

An approach to automate complex sewing operations needs to address the challenges of high product and process variance. The technical solutions must handle both the low volume and high mix that is common in customized production, and the high complexity of handling limp materials. This suggests that sensor systems and in-process measurement able to describe the sewing process, in addition to real-time control based on the measurements, are needed to solve the complex task of sewing. This paper describes an approach to automated sewing using force sensors and sensor-based real-time control.

J. Schrimpf is with the Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: johannes.schrimpf@itk.ntnu.no.

L. E. Wetterwald is with SINTEF, Trondheim, Norway.

II. Related work

Gershon et al. presented an approach for robotised sewing, based on a system consisting of a sewing machine and a robot manipulator with a multi-fingered end-effector [1]. They proposed cameras, encoders, proximity sensors and fabric tension sensors as sensor systems to control the sewing process.

In 1998, Seliger and Stephan presented an overview of the challenge of automated sewing [2]. They emphasize that there are significant difficulties when automating the sewing and handling process due to non-linear material behavior. They also point out the difficulties that arise from 3D shaped products, which are sewn from 2D fabrics. Further they suggest the need for adaptive control strategies with sensor input capable of measuring both the feed rate and seam allowance of the process.

Koustoumpardis et al. presented a robotized sewing approach that is based on artificial intelligence [3]. They used neural networks to control the tension in the fabric using force measurements on the robot tool, focused on controlling a single piece of fabric.

In 2009, Winck et al. [4] presented an approach for automated sewing of two fabrics including an industrial sewing machine and servo controlled feed dogs which both fed and controlled the fabric and showed the feasibility of such a sewing cell by building a prototype. They focused on high-speed control and precise actuation. The fabrics were separated by a thin plate to enable independent control of the fabrics. The position information was gained by a vision system which detected individual tracks in the fabric. Their prototype implemented open loop control using a predefined path and they emphasized the need for feedback control of the fabric position.

Wetterwald et al. presented a sewing cell which was able to attach a hide part to a fiber part as a subassembly for later use in recliners [5]. The setup consisted of a KUKA KR60L30 HA Robot and a DA550 sewing machine. The sewing machine was capable of crimping the part by using different in-feed and out-feed rates in the sewing process. In addition to natural variations in the process, the crimping generated the need for the

continuous synchronization of the robot speed with the actual feed rate of the sewing machine. To synchronize the robot speed with the sewing machine's feed rate, an optical x/y displacement sensor was used to measure the sewing speed, and a controller program computed the robot speed in the sewing direction. A vision system using laser triangulation detected the edge of the hide, and a controller computed correction values for the pre-programmed trajectory such that the seam was located at a constant distance from the edge.

While the previous work mainly focuses on using a single robot to control the fabrics, this work focuses on sewing application that require two robots to control the sewing process due to different shapes of the workpieces. Winck et al. also focus on two pieces of fabric, but in contrast to this work they used servo controlled feed dogs and focus therefore on a different sensor system.

III. Hardware

Our sewing test cell consists of four main parts:

- The sewing machine,
- two industrial robots with real-time interface,
- two force sensors and
- an edge sensor.

An overview of the above-named hardware will be given in this section.

A. Sewing Machine

The sewing machine used in the test cell is a DA195. This is an industrial sewing machine with differential feed for the upper and lower fabrics. The feed rates can be adjusted mechanically by two wheels on the front of the sewing machine, but are not adjustable by the control system of the test setup. The sewing speed is controlled by an Efka AB321A drive and connected to an Xmega-based development board which sets the speed reference signal and enables real-time speed control using Ethernet communication from the central control PC. Other sewing machine function units, e.g. sewing-foot control and stitch counter are also connected to the Xmega board, and thus are accessible from the control PC.

B. Industrial Robots

Two Universal Robots UR-6-85-5-A 6-axis industrial manipulators are installed in the sewing cell to handle the materials before, during and after the sewing process. The controller of the UR-6-85-5-A consists of a PC running Debian GNU/Linux and a low-level controller. The robot offers an interface that allows to send motion commands via TCP with a frequency of up to about 20 Hz.

Additionally, a real-time connector is programmed and runs on the controller PC to obtain access with a higher update frequency. This uses an API to access the low-level controller which allows joint position, velocity and acceleration control. The internal cycle time of the low-level controller is 8 ms, cf. Fig. 1.

The Python-based robot controller framework PyMoCo is used for motion generation [6]. It utilizes an inverse Jacobian controller, which updates the robot joint angles with a frequency of 125 Hz, according to the commanded tool velocity in the task space. This interface is planned to be used in the control program.

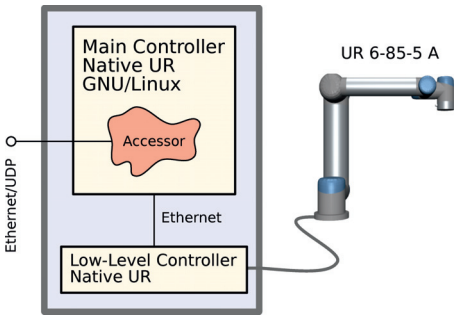


Fig. 1. The Universal Robot with controller. An accessor was built to access the low-level controller (LLC) and enable motion control from an external PC. ©Morten Lind 2011.

C. Force Sensor

The robots in the sewing cell are equipped with force sensors between the robot tool flange and the gripper to measure and control the pulling force during sewing. The sensor type is ATI Mini45 with 6 DOF, and can measure approximately ±300 N with a resolution of approximately 0.1 N. The force and torque measurements are captured via an ATI Net F/T and communicated to the control PC over an Ethernet connection. The force measurement is integrated in a closed control loop for the robots. The low-level robot controllers have an internal cycle time of 8 ms, and the sewing machine has a typical operating range of 120 – 3600 rpm causing an oscillation with a frequency of 2 – 60 Hz in the force measurements due to the movement of the feed dog in the sewing machine.

D. Edge Sensor

In order to maintain the correct seam allowance and alignment between the two parts, a sensor system capable of measuring the actual position of the part edges in front of the needle point must be integrated into the sewing cell.

A sensor system was built which consists of two 1D-optical arrays. These sensors act like a 1D camera and can measure the amount of light in each pixel. The sensor system is designed to be as flat as possible and is installed between the two parts. The sensor system is mounted at a distance of about 5 cm from the needle to ensure smooth feeding for both parts.

Stephan [7] uses a similar approach utilizing an active infrared sensor to track contours and detect edge defects during sewing.

Using a PID controller promising tests were performed which verified the feasibility of edge control using this sensor, cf. Fig. 2.

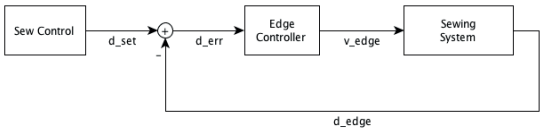


Fig. 2. Edge controller for a one robot solution.

IV. Velocity Synchronization

Velocity Synchronization in automated sewing is a complex task. The movement speed of the robots has to be synchronized with the speed of the sewing machine and with each other. The control task is to match the seam length of the two parts while keeping the tension between an upper and lower force limit. In the case of different feed rates of the two parts due to material properties and other disturbances, a force difference will be accumulated during the sewing process which needs to be compensated for.

The following challenges arise from the complex nature of a sewing process and make a modelling of the process difficult:

- Properties such as stiffness and thickness of different parts vary to a large degree.
- Dependent on the different material properties the same sewing machine speed results in different feed rates and thereby actual sewing speeds.
- Changes in the force applied to the workpiece influence the feed rate in ways that are difficult to predict. Examples are changes in the stitch length and slip in the feeding mechanism. These are not only dependent on the force, but also the material characteristics.
- The parts that have to been sewn together have different shapes resulting in different performance during sewing.
- The periodical movement of the feed dog disturbs the measurements, especially when the gripper position is near the feed dog.

In this section different solutions for a one-robot system are discussed and then extended to a two-robot system.

The setup for a one-robot test system is illustrated in Fig. 3. It consists of the sewing machine and the robot with a force sensor attached to the tool flange. The fabric is fastened to the robot at point $P_R = (x_r, y_r, z_r)^T$. The needle $P_N = (x_n, y_n, z_n)^T$ is the reference point for the sewing coordinate system. The starting point $P_0 = (x_0, y_0, z_0)^T$ is the point on the fabric where the seam is started. When the sewing machine is active, it moves the fabric with a velocity v_{sew}.

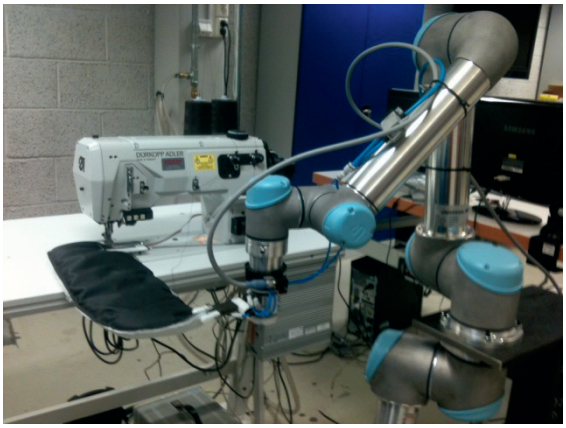


Fig. 3. Test setup for velocity synchronization between the sewing machine and the robot.

Since the feed rate depends on the properties of the fabric, the sewing speed is not constant when applying a constant reference speed in rpm to the sewing machine. The parameters which define the sewing process are the sewing speed v_{sew} and the robot speed v_{rob}.

The following two strategies are suggested for a **one-robot** solution:

- The robot matches the sewing machine speed by pulling with a constant force.
- The robot and the sewing machine move with a constant speed. The tension is controlled by pull/release pulses of the robot.

In the first approach, the sewing speed is controlled by a force controller, for example a PID controller, which keeps the stretch in the workpiece constant. This is done by forward feeding of the estimated sewing speed to the robot, while compensating for deviations and disturbances using force measurements at the robot tool. A control scheme is depicted in Fig. 4.

In the second control method, both the robot and sewing machine move with a constant speed. The system is controlled to keep constant tension. If deviation between the measured force and the force set point

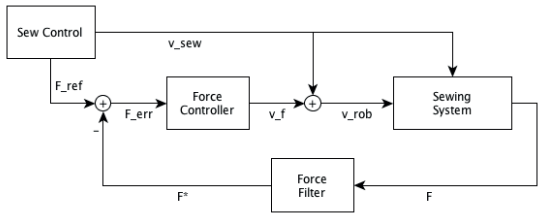


Fig. 4. Force velocity controller. The robot matches the sewing machine speed by pulling with a constant force.

exceeds a predefined threshold, a temporary speed change is applied in the robot controller to create a pull pulse or release pulse, see. 5.

In other words:

- To decrease the tension, the robot is temporarily slowed down, then speeded up, and finally set back to the predefined speed when the robot has returned to the planned trajectory.
- To increase the tension, the robot is speeded up, then slowed down and finally set back to the predefined speed when the robot has returned to the planned trajectory.

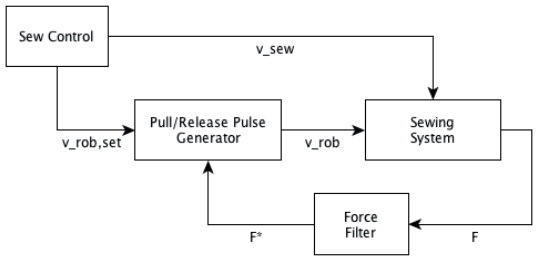


Fig. 5. Force velocity controller. The robot and the sewing machine use constant speed. The pulling force is controlled by pull/release pulses of the robot.

While the first method is sufficient for a single-robot system. For a two-robot system, the second method, which allows sewing with constant force and constant average speed, may be necessary to meet the corners after the sewing process.

When two robots are used to feed two parts into the sewing machine the feeding processes have to be controlled independently. It is necessary to synchronize the speed of movement between the robots in order to get a uniform seam. Using this strategy, the corners will match at the end of the seam.

Three different strategies for a **two-robot system** are suggested:

- Synchronous robot movement.
- Asynchronous robot movement.
- Differential feeding in the sewing machine.

The first control strategy is based on parallel movement of the two robots. During the sewing process, the robot velocity is controlled by a force controller which utilizes the force measurement of the two robot tools to keep constant tension in the workpieces. When a difference between the two measured forces is detected, the movement of the robots is slowed down to increase the tension in the fabrics. Due to nonlinearity, the tension in the fabric which has the higher tension will increase more than the tension in the other fabric. This leads to a reduced velocity in this part at the sewing point. After returning to the desired robot velocity, the force difference will be decreased.

The second strategy is asynchronous feeding, allowing different speeds in the two robots. In this way increasing force on the one robot and a decreasing force on the other robot can lead to a faster change in the feeding velocities at the needle. After the correction, the robots will return to the parallel feeding to ensure a smooth seam process for the remaining part of the seam. The control strategy for each robot is related to the second approach in the single-robot system.

While the first two approaches use the robot movement speed to control the feeding, the third approach is based on control of the feeding mechanism in the sewing machine. The feeding for the two different fabrics can be adjusted by wheels on the front of the sewing machine. By controlling the feeding speed using servo motors, this mechanism can be used to to control the sewing velocities independently, while the robots move parallel to each other with a constant speed.

V. Experiments

A series of experiments was conducted to investigate the suggested sewing control strategies. First, the mechanical behavior of the workpieces was tested with a static force measurement setup. Second, a series of sewing experiments was performed to analyze the resulting force when sewing the parts.

The following experiments were conducted:

- Static force measurement of the workpiece to determine the nonlinear material stiffness.
- Force measurement during sewing with constant sewing machine speed and force control.
- Force measurement during sewing with pulling and releasing pulses on the robot movement.

Only one robot was used in these experiments and the same part is used for all experiments.

A. Static Force Measurement

This experiment was performed to measure the behavior of the workpieces when a stretching force is applied. It was expected to see a spring-like behavior of the part, with a nonlinear relationship between the distance and resulting force. From previous research [5] it is also known that the workpieces will have varying stiffness and stretch properties depending on the type of hide and thickness, and that these properties also can vary over the part.

The experiment was conducted by fixing the workpiece to a vice at one end, and to the force sensor on the robot at the other end. Then the force was increased by moving the robot increments away from the fixed end. This experiment was repeated several times with two different lengths between the fixed end and the robot. The results are shown in Fig. 6.

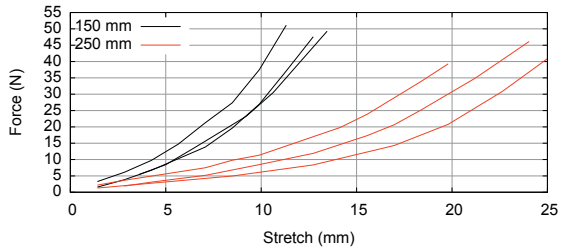


Fig. 6. Relation between the applied force and the stretch of the textile parts.

The experiment clearly shows the nonlinear stiffness. It can be seen that the variation in the stiffness is very large for the same part length. This shows that it is difficult to find a reliable stiffness model, even for a single part.

B. Force Measurement During Sewing with Force Control

The second experiment uses the force control scheme depicted in Fig. 4. A simple P controller is used, with a working point of 2 N. A part of approximately 450 mm length is fixed to the force sensor on the robot, and the other end is placed under the feed dog of the sewing machine with as little pre-tension as possible. Then the sewing machine is started with a constant speed. The robot movement is updated at a frequency of 20 Hz. An integral term in the controller was neglected due to stability issues with the low update frequency, but is planned to be included when using the 125 Hz robot interface.

Five series are shown in Fig. 7 as a running average plot to filter the oscillating force caused by the feed dog movement.

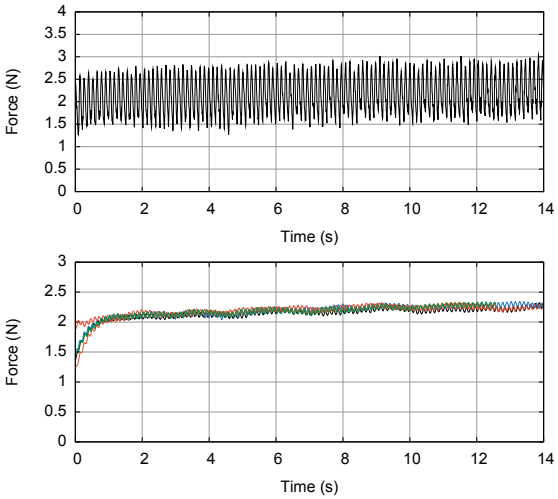


Fig. 7. Unfiltered (one series) and filtered (five series) force during sewing with force control. The high-frequent oscillation is due to the feed dog movement.

The experiment shows that it is possible to control the sewing force with simple means using a force sensor on the tool. However when using this control scheme with a P(ID) controller, the sewing time for each part may be different since the feeding rate for different parts varies for the same sewing machine speed due to material properties. In a sewing process where two parts have to be sewn together, it is important that both parts are sewn at the same speed which results in varying forces when using a PID controller for the sewing speed. Therefore a more advanced control scheme has to be considered which controls the force and the sewing speed at the same time.

C. Force Measurement During Sewing with Pull and Release Pulses

The setup of the third experiment was identical to the second experiment, but with constant speed on both robot and sewing machine. Tests were conducted first without control and then with a pull and release pulse of the robot. The objective of the experiment was to confirm that a pull pulse can be used to decrease the tension in the workpiece while a release pulse can be used to increase the tension. This effect is due to different feeding rates at different sewing forces while holding the sewing machine speed constant.

The pull pulse was applied by stopping the robot for 0.5 seconds and then doubling the speed for 0.5 seconds, causing the robot to return to the original trajectory after 1 second. The opposite was done to apply a release pulse.

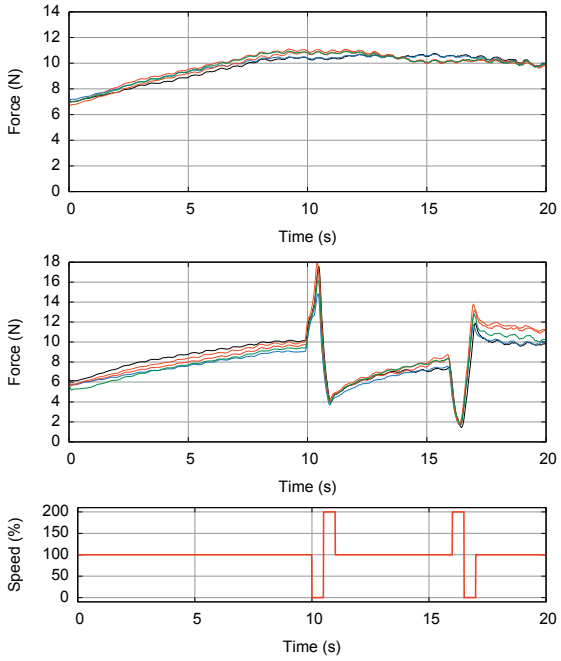


Fig. 8. Force and robot speed during sewing with constant speed on both robot and sewing machine with and without pull and release pulses. The lower figure shows the robot speed when applying pulses.

Five series are shown in Fig. 8 as a twelve data point running average (0.12 s) plot to filter the oscillating force caused by the feed dog movement. The force plot clearly exemplifies the drop in the sewing force after applying the pull pulse, and the sewing force increases after applying the release pulse. When sewing two parts together this effect can be used to control the sewing speed of the two parts individually and at the same time make sure that the total feed length of the two parts is identical. It should be mentioned that the velocity changes in the experiment were quite high to show the principle of the method, while in a productive system it is suggested to used much smaller pulses in regular time intervals with the pulse time and intensity dependent on the force error. In this way a nearly linear movement with a constant speed can be applied on the workpieces while controlling the force.

VI. Conclusion and Future Work

A. Conclusion

An approach for sewing two parts using two force controlled robots has been presented. It focuses on the sewing of parts with edges of nearly identical shapes, but allows for deviations which makes it necessary to control the two parts independently. The concept includes strategies for velocity synchronization between the robots and the sewing machine as well as a strategy for edge control. The complexity of velocity synchronization is discussed and different strategies are proposed.

A test setup was built to verify the possibility to control the robot movement in the sewing direction by using a force sensor on the tool holding the workpieces. Different control strategies were suggested, and sewing tests were performed to measure the actual response of the applied control method. The results show that it is possible to control the tension of the workpiece during sewing without actively controlling the sewing machine, and at the same time follow the planed trajectory when allowing short-period control deviations. This is important because the robot trajectories are constrained by the need for matching the two corners of the workpieces at the end of the seam.

B. Future Work

Further tests have to be performed including a test setup with two robots to investigate any dependencies between the part systems for the two fabrics. Also the control algorithm for velocity adjustment in case of variations in the feed-in speed have to be implemented and tested to ensure that there is an even seam, and that the corners fit together.

VII. Acknowledgements

The authors wish to thank the SFI Norman programme and the *Automated 3D Sewing* project founded by the Research Council of Norway. The authors also want to thank Ekornes ASA for close cooperation.

References

[1] D. Gershon, "Strategies for robotic handling of flexible sheet material," *Mechatronics*, vol. 3, no. 5, pp. 611 – 623, 1993.

[2] G. Seliger and J. Stephan, "Flexible garment handling with adaptive control strategies," in *Proceedings of the 29th ISR*, 1998, pp. 483 – 487.

[3] P. Koustoumpardis and N. Aspragathos, "Robotized sewing of fabrics based on a force neural network controller," in *Intelligent Robotics and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, vol. 7101, pp. 486–495.

[4] R. Winck, S. Dickerson, W. Book, and J. Huggins, "A novel approach to fabric control for automated sewing," in *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, July 2009, pp. 53 – 58.

[5] L. Wetterwald, S. Dransfeld, H. Raabe, and T. Ulleberg, "Flexible robotic sewing with real time adaptive control," in *Proceedings 2nd Conference on Assembly Technologies and systems (CATS 2008)*, 2008.

[6] M. Lind and J. Schrimpf, "Python-based robot motion control," *Journal of Software Engineering for Robotics*, 2011, submitted.

[7] J. Stephan, "Efficient sensor system for online contour tracking in textile machinery," *Sensor Review*, vol. 22, pp. 328–333, 2002.

6.5 Implementation Details of External Trajectory Generation for Industrial Robots

J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, *Implementation details of external trajectory generation for industrial robots*, Proceedings of IWAMA 2012 - The Second International Workshop of Advanced Manufacturing and Automation, 2012

Declaration of co-authorship

The overview and structure of the paper were designed by Johannes Schrimpf in close cooperation with Morten Lind.

The example applications and the experiment setups were a collaborative effort among Morten Lind and Johannes Schrimpf. The used trajectory generators were developed as follows:

- 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 trajectory generator implementation based on OROCOS PyKDL was designed and programmed by Johannes Schrimpf. It was tested and used in cooperation among Johannes Schrimpf and Morten Lind.
- The trajectory generator implementation based on the OROCOS C++ library was designed, programmed and tested by Johannes Schrimpf.

The evaluation and presentation of the experiments is cooperative work of Johannes Schrimpf and Morten Lind.

The paper was written by Johannes Schrimpf. Geir Mathisen was involved in discussions during the whole writing process. Amund Skavhaug and Morten Lind were especially involved in discussions on real-time linux.

All authors contributed in the review of the final version.

Implementation Details of External Trajectory Generation for Industrial Robots

Johannes Schrimpf[1], **Morten Lind**[2], **Amund Skavhaug**[1], **and Geir Mathisen**[1]

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

Abstract Sensor-based robot control is an important field in flexible and reconfigurable production systems. External robot controllers can be used to achieve real-time interaction between the user application and the robot. Only few robots offer the possibility to interact with the low-level robot system in real-time. This paper describes implementation details of external robot control that are used in visual and force servoing applications. An overview is given over different layers in the control structure of a typical robot controller system. The focus is on the development of trajectory generators and real-time interfaces to the industrial robots. The used real-time interfaces are described and the real-time requirements for the external trajectory generator are discussed. Experiments are presented showing the performance of the example implementations when using different frameworks, libraries and programming languages.

Keywords Industrial Robots, Trajectory Generation, Real-Time Robot Control

1 Introduction

In the past decades industrial robots have gained an important role in manufacturing systems. This is due to the flexibility and reconfigurability of robots. While in the past pre-programmed robots were used to execute repeatable operations without interfacing with the environment, today most industrial robot systems use sensors to react on external inputs. One example are pick-and-place operations where the robot controller uses a vision system to detect stationary workpieces that have to be picked and moves them to a target position. Dependent on the input of the vision system, the workpiece positions are given to the trajectory generator in the robot to move the robot to the desired position. In contrast to the pick-and-place scenario, the control of complex processes like sewing demands much higher update frequencies for the sensor system and a high-frequency interface to the robot controller is needed to control the robots trajectory in real-time while the robot is moving. While pick-and-place robot systems are well-established in the industry, sensor-based real-time robot control is mainly encountered in research applications. Reasons for the slow

The authors wish to thank the SFI Norman programme and the *Automated 3D Sewing* project founded by the Research Council of Norway.

establishment of real-time control are the high complexity of the resulting control systems, but also the low availability of industrial robots offering real-time interfaces and controller software that can be used with the few industrial robots that offer real-time interaction.

Industrial Robots usually ship with a proprietary robot controller including a tech pendant to program the robot. The user can write programs in the robot specific programming language including commands for robot movements and for the program flow. Usually digital or analog inputs and outputs can be addressed to interact with the environment and most robots allow communication via a serial port or Ethernet. This is sufficient for standard operations where sensor analysis and robot motion are processed one after another. However, when the sensor analysis must affect the robot motion in real-time, special interfaces are required. Few commercial industrial robots offer this possibility of real-time robot control. Robots often have to be modified in order to control them from an external entity. In some cases software modification are sufficient while in other cases hardware modifications are needed to access the desired low-level functions. A survey on low-level robot control is presented in [Kröger and Wahl, 2010].

On the other hand there is a growing number of frameworks that help the user to create their own external motion controllers for the available low-level interfaces. Due to the increase of CPU power it becomes also possible to use user-friendly high-level programming languages such as Python that require less programming effort than usually used languages as C or C++. One example for a library for kinematics and dynamics calculations that can be used in trajectory generators is the KDL library of the OROCOS project [Bruyninckx, 2001]. OROCOS KDL is written in C++ and offers bindings for Python named PyKDL. Another tool for external trajectory generation is the PyMoCo framework [Lind, 2012].

This paper intents to give insight in different implementations of external trajectory generation based on sensor input and how they are integrated into the control system. The implementations are using C++ or Python as programming language and use the OROCOS KDL library or the PyMoCo framework.

Section 2 presents different applications that use real-time trajectory generation. Section 3 gives an overview over the different parts of an external robot controller. Section 4 describes the concepts of trajectory generators. In Section 5 different implementations are described and the response time of these implementations is measured in Section 6. Section 7 concludes the paper with the results gained from the experiments.

2 Example Applications

This section presents different applications that use external application controllers and trajectory generators connected to real-time interfaces of the corresponding robot to achieve sensor-based real-time control.

Line Following: Applications like grinding or welding may require that the tool is orientated perpendicularly to the workpiece surface. A demonstrator system was build to demonstrate following of a visible line on the workpiece with the tool aligned perpendicularly to the surface, cf. Fig. 1. A line-of sight algorithm was

used for the line following and a laser-based triangulation sensor was developed to measure the tool orientation. A Nachi SC15F 6-axis industrial robot was used as hardware platform. The system was presented in [Schrimpf et al., 2011].

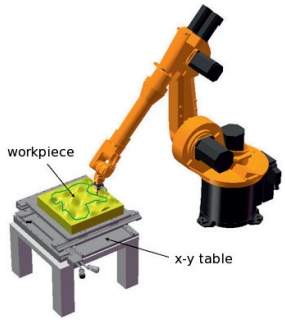


Fig. 1 Model of the line-following test platform. The robot tools follows a marked line.

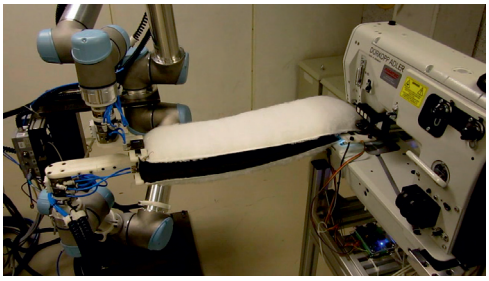


Fig. 2 Automated sewing cell: The sewing process is controlled in real-time by integrating sensors to measure the tension in the workpiece and the location of the edge in the control loop.

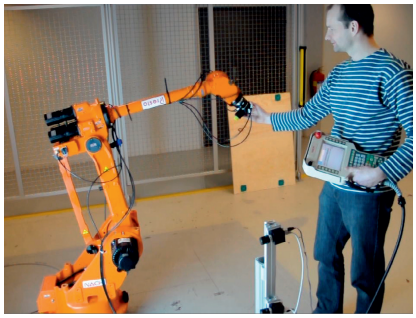


Fig. 3 Force control: The robot can be controlled by moving the tool.

Automated Sewing: Automated sewing is a challenge in production technology. Usually pre-programmed movement patterns cannot be used for the robot to control a sewing process due to the unpredictable behavior of limb material and variations in the material characteristics. Fig. 2 shows a sewing cell that is able to sew different workpieces without knowledge of the exact shape of the work-pieces. Sensors are integrated for detection of the stretch in the workpiece and the position of the edge in front of the needle. Two Universal Robots UR-6-85-5-A are used to control the work-pieces in real-time based on the sensor measurements. The system was presented in [Schrimpf and Wetterwald, 2012].

Force Control: Force control of the robot tool is a demanding application in respect to timing constraints. Force control is used in applications where a tool

has to apply a given force to a workpiece, for example in grinding applications. Other applications is teaching the robot by moving the tool and recording the movement or waypoints of the robot for later use in programs. Due to the stiff behavior of most industrial robots small delays can lead to damages or non-responsive behavior of the system when the real-time requirements are not met. An example for controlling the robot by moving the tool is presented in [Lind et al., 2010], cf. Fig. 3.

3 Real-Time Robot Control

To describe the structure of external robot controllers, Fig. 4 introduces a model that defines different layers from the application to the robot servos.

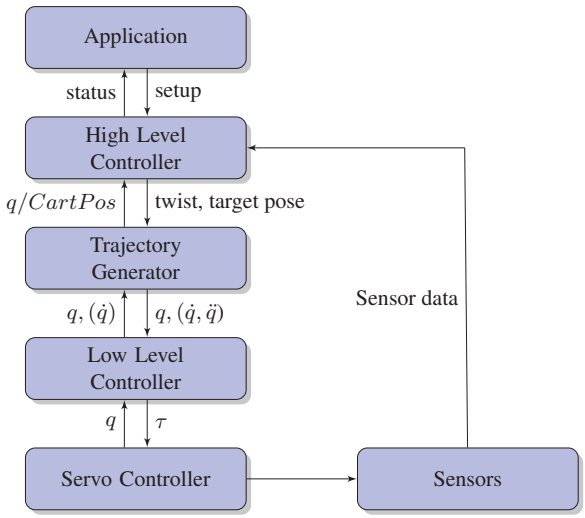


Fig. 4 The model of a real-time robot controller. CC BY-SA J. Schrimpf.

It is important to notice that the borders between the different controllers can be floating and that the different controllers are not necessarily implemented separately.

The High Level Controller is used directly by the application. It generates commands to the trajectory generator by processing sensor data from the process. In many sensor-based real-time controllers the commands to the trajectory generator are tool velocities to move the robot tool in the desired direction.

The Trajectory Generator is responsible to calculate a trajectory from the commands from the high-level controller. The high-level controller usually commands target positions or velocities in Cartesian coordinates that have to be converted into angles q and possibly angle velocities $\dot{q}$ and accelerations $\ddot{q}$ in the joint space. The trajectory generator includes the necessary kinematics calculations and trajectory interpolators for movements to a target.

The Low-level Controller is responsible for calculating the inputs for the servo controller motors. The input of the low-level controller is a trajectory in joint space. It typically includes PD controllers and inverse dynamics calculations that include gravity compensation and a dynamic model of the robot.

The Servo Controller consists of the electro-mechanical system of the robot, typically consisting of servo motors and their respective controllers. These controllers may be integrated in the motor hardware or can be separate. In this layer there is no coupling between the different motors and each controller only controls one motor.

4 Real-Time Trajectory Generators

To control a robot in real-time from an application, a real-time trajectory generator is needed to generate a trajectory. Different trajectory generators are necessary for different control scenarios. Simple trajectory generators move the joints of the robot independently, either taking a target joint vector or a vector of joint speeds. These movements generally result in a nonlinear movement in the Cartesian space. More sophisticated trajectory generators are used to move the tool linearly in the tool space. As input either a target tool pose or velocity is given by the application controller.

Examples of trajectory generators are:

Joint Linear Generator The joint linear generator takes as input a target joint configuration q_{target} and interpolates the trajectory linearly in joint space.

Joint Velocity Generator The input for the joint velocity generator is a target joint speed $\dot{q}_{target}$. The joints are moved with the commanded joint speed until a new command arrives.

Tool Linear Generator The tool linear generator moves the robot linearly in tool space to a target pose P in tool space.

Tool Velocity Generator The tool velocity generator moves the tool linearly in tool space with a given twist (6D-movement speed). The tool is moved with the target speed until a new command arrives.

Implementations of trajectory generators differ in functionality and strategies for different situations. Examples are limits for robot joint positions, velocities and accelerations or the behavior near singularities resulting in downscaling of the movement when limits are reached, or changes in the trajectory to avoid singularities. Other differences may be found among the representation of the input values and different back-ends for handling of the real-time communication to the robot. Linear trajectory generators can be implemented to provide point-to-point movements or more complicated trajectories using blending mechanisms.

Limits for velocities and accelerations can be defined by the robot hardware or by the control task. The constraints by the robots are usually defined per joint while task constraints usually are defined in tool space.

5 Implementations

Three different implementations are presented in this section based on different frameworks and with different implementation focus. The focus of the implementa-

tions is on *tool velocity generators* since these are commonly used in sensor-based robot control, but also other trajectory generators are provided, for example linear point-to-point trajectory generators to move the robot to the starting position.

PyMoCo: PyMoCo is a real-time trajectory generation framework that is entirely implemented in Python. Python was chosen as platform to allow for rapid prototyping of new controllers and interfaces and examine the possibilities to use the Python programming language in real-time robot control. The focus of PyMoCo is to provide general interfaces that can be adapted to a large extent to different robot interfaces and application controllers. It includes a wide range of possibilities to constrain the generated motion to maximum joint velocities and accelerations both in actuator space and joint space[3].

Trajectory Generator Based on OROCOS PyKDL: A real-time trajectory generator has been implemented based on the Kinematics and Dynamics Library that is part of OROCOS [Bruyninckx, 2001]. It provides the minimal functionality that is needed to control a Universal Robots UR-6-85-5-A linearly in tool space, either by setting a target pose or a twist. For the interpolation in tool space to move the robot to the target position, the python-math3d library is used instead of KDL due to the lack of Python bindings for the according functions. However the inverse kinematics and the inverse Jacobian calculations are done by KDL.

Trajectory Generator Based on OROCOS KDL: Another implementation of a real-time trajectory generator was programmed entirely in C++ using the OROCOS KDL libraries for both kinematics calculations and trajectory interpolation.

6 Experiment: Respone Times of the Implementations:

To evaluate the response times of the different real-time trajectory generators a test platform was set up. The used PC had an 8 core 2.8 GHz CPU and 8 GB RAM. The low-level interface was emulated by a small C++ program on the same PC as the trajectory generators. The time between sending a status update and getting the response from the trajectory generator was recorded for a robot motion where the tool moves with a constant low speed in the workspace. The response times were recorded for 5000 steps for each implementation. The power management of the Linux PC was set to *performance* to prevent higher response times due to frequency scaling. The results are shown in Fig. 5.

The figures show that the C++ implementation using the OROCOS KDL library gives the lowest response time with a mean value of 0.265 ms. The Python implementations respond slower with mean response times of 0.345 ms for the implementation using OROCOS PyKDL and 0.537 ms for the trajectory generator using PyMoCo.

Without using the network the response time of all samples were less than 1 ms. This will increase slightly with the use of external Ethernet. Of more concern are the few response times that are about two times larger than the mean response time.

[3] This is important for robots that have a different joint and actuator space for example due to parallel links in an otherwise serial robot.

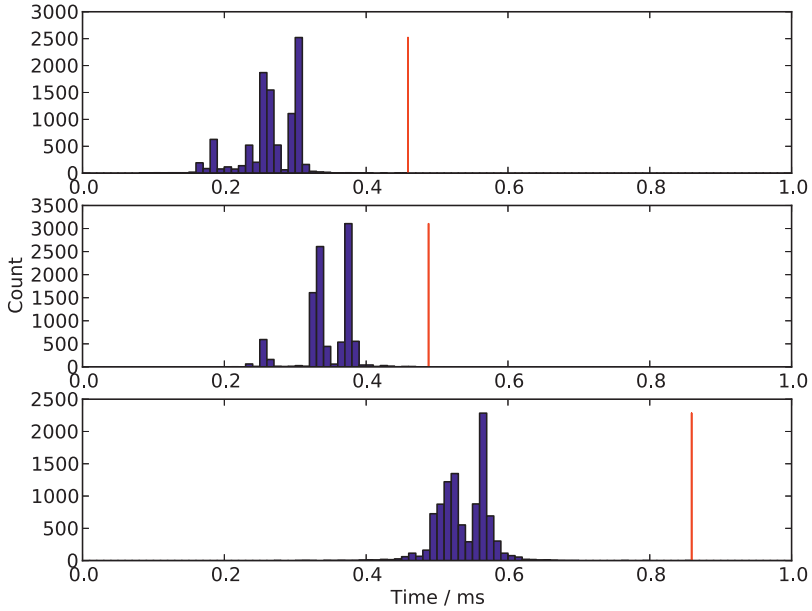


Fig. 5 Response times for the different implementations for 10000 samples: OROCOS KDL, OROCOS PyKDL, and PyMoCo. The red marker is at the position of the maximum response time.

Such behavior must be expected using non-specialized hardware and not using a real-time operating system. However, a system using an external controller must be designed failsafe and with a graceful degradation regarding lost samples in mind anyway. In this particular case of the Universal Robot, a response time exceeding 4 ms results only in a skipped interpolation cycle. Further it is implemented a strategy that allows for response times within 8 ms before the connection is defined as lost. These are examples of deadlines for usefulness of results and for lost communication and are both application and implementation specific. That is, different strategies may lead to different values.

The results indicate that it is feasible to run a trajectory generator with a response time not exceeding the deadline of 4 ms at a frequency of 125 Hz.

The distinction between soft and hard real-time in a formal sense has not been addressed in the presented work. The authors are well aware that a hard real-time system may need to be formally proven to be classified as such. The focus of the real-time quality of the control system is on usability instead of formal proof of worst case response times. There are numerous issues with plain Linux and real-time, as well as several solutions to address these problems, e.g. special configuration of the kernel before compilation regarding timers, interrupt handling and preemption, as well as extensions with separate full real-time systems such as RTLinux

and RTAI [Arthur et al., 2007]. The authors regard it as important to use an unmodified "desktop system", both because of maintainability and ease of use in general.

One performance related issue was shown to be of such importance that it had to be addressed despite this. Common Linux distributions include an automatic frequency scheduler to save energy when there is no heavy usage of the CPUs. This scaling of the CPU can lead to a higher response-time when active.

7 Conclusion

An overview over the concept of external real-time robot control has been demonstrated and examples of industrial applications have been presented. The focus was on real-time trajectory generation. Three different implementations of real-time trajectory generators have been described that differ in functionality and the underlying frameworks. Experiments have been conducted to evaluate the response times of these different implementations. The experiments were intended to empirically evaluate whether deadlines can be expected to be exceeded or not. The results indicate that it is feasible to run an external trajectory generator on common off-the-shelf hardware and standard desktop Linux and still fulfill the real-time requirements of the low-level control interface of a typical industrial robot.

References

Arthur, S., Emde, C., McGuire, N., (Nov 2007). Assessment of the Real-time Preemption Patches (RT-Preempt) and their impact on the general purpose performance of the system. In: Ninth Real-Time Linux Workshop. http://www.realtimelinuxfoundation.org/events/rtlws-2007/ws.html

Bruyninckx, H., (2001). Open robot control software: the orocos project. In: Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on. vol. 3, pp. 2523 – 2528

Kröger, T., Wahl, F., (2010). Low-level control of robot manipulators: A brief survey on sensor-guided control and on-line trajectory generation. In: Kubus, D., Nilsson, K., Johansson, R. (eds.) ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications. pp. 46–53. Technical University of Braunschweig, http://www.rob.cs.tu-bs.de/en/news/icra2010

Lind, M., Schrimpf, J., Ulleberg, T., (2010). Open external real-time servo-level robot control framework. In: Proceedings 3rd Conference on Assembly Technologies and systems (CATS 2010)

Lind, M., (2012). Open Real-Time Control and Emulation of Robots and Production Systems. Ph.D. thesis, Norwegian University of Science and Technology, Department of Productions and Quality Engineering

Schrimpf, J., Lind, M., Mathisen, G., (sept 2011). Time-analysis of a real-time sensor-servoing system using line-of-sight path tracking. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. pp. 2861 –2866

Schrimpf, J., Wetterwald, L.E., (May 2012). Experiments towards automated sewing with a multi-robot system. In: International Conference on Robotics and Automation (ICRA 2012) (accepted)

6.6 Real-Time System Integration in a Multi-Robot Sewing Cell

J. Schrimpf, L. E. Wetterwald, and M. Lind, *Real-time system integration in a multi-robot sewing cell*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012

Declaration of co-authorship

The sewing cell was designed by Johannes Schrimpf in close cooperations with Lars Erik Wetterwald and Morten Lind. The implementaton was done by Johannes Schrimpf and Morten Lind.

The control concept was designed and implemented by Johannes Schrimpf.

The real-time interface to the Universal Robots UR5 was programmed in close cooperation between Johannes Schrimpf and Morten Lind.

The coordinate system calculations and the corresponding illustration were created by Morten Lind and Johannes Schrimpf.

The experiments were planned by Johannes Schrimpf and coducted by Johannes Schrimpf and Morten Lind in collaboration.

The block diagrams were prepared by Johannes Schrimpf.

The paper was single-handed written by Johannes Schrimpf. Lars Erik Wetterwald contributed in occasional discussions. Lars Erik Wetterwald and Morten Lind contributed in reviewing the final versin of the paper.

The video appendix was prepared by Johannes Schrimpf.

Real-Time System Integration in a Multi-Robot Sewing Cell

Johannes Schrimpf[1], Lars Erik Wetterwald[2] and Morten Lind[2]

Abstract— **The sewing process is a manufacturing technology which presents severe challenges for automation. Due to variations in the material properties and unpredictable mechanical compliance, real-time sensor-based control strategies are necessary to achieve satisfying results. This paper presents a sewing cell consisting of two lightweight industrial robots and a sewing machine. Sensors are included both for force and edge positioning control. Experiments are presented showing the performance of the proposed real-time control framework. The experiments focus on the real-time control loop for force and edge control during the sewing process.**

I. Introduction

Sewing is a labor intensive process, which is hard to keep as part of manufacturing in high cost countries. Most of the sewing industry, including the equipment manufacturers have outsourced their activities to low cost countries. The motivation for keeping these processes in high cost countries are usually the need for short lead time in order-based production, and that the sewing process is highly integrated with the rest of the value chain. This is the case for a Norwegian producer of recliners, and different approaches to automate their manual sewing processes of leather covers for recliners have been a major research activity the last years.

This paper describes an automated sewing cell developed for joining parts with different shapes using two individually controlled robots. The specific case presented involves the joining of four leather parts which make up the footstool cover shown in Fig. 1. At the same time it is an important goal for the research work to develop flexible solutions capable of handling different part geometries and material properties. The chosen cell design uses two lightweight robot arms which can control two parts individually. This enables the system to sew parts with different geometry, resulting in 3D shaped assemblies. Force measurements are used to control the tension in the parts and edge measurements are used to control the seam position relative to the edges of the parts. The controller system has no prior knowledge of the shape of the parts. The main focus in this paper is verification of the control framework and measurements of the force and edge error during the sewing process.

Several research groups have worked in the area of automated sewing:

[1]Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: johannes.schrimpf@itk.ntnu.no.

[2]SINTEF Raufoss Manufacturing AS, Trondheim, Norway.

Fig. 1. Processed part which has been sewn in the sewing cell.

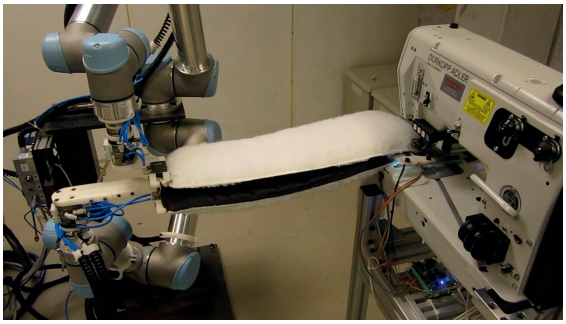


Fig. 2. The sewing cell with two robots and a sewing machine.

Gershon et al. [1], [2] presented a sewing demonstrator with sensor feedback. It consisted of a single robot and a sewing machine. They used fabric tension measurement and an edge sensor system based on cameras to control the robot in real-time. They proposed different sensors such as cameras, encoders, proximity sensors and fabric tension sensor to control the sewing process.

An overview of the challenge of automated sewing was given presented by Seliger and Stephan [3]. They emphasize the difficulties of material handling and sewing that arise from the non-linear behavior of the material. They suggest adaptive control strategies that use measurement of the seam allowance and the feed

rate during the sewing process.

Koustoumpardis et al. [4] presented an approach of automated sewing using adaptive control strategies. They used a controller that was based on neural networks to control the tension in the work-piece. They focused on sewing of a single piece of fabric.

Winck et al. [5] described a sewing machine with a servo-controlled feeding mechanism that both controlled the edge and fed the work-pieces into the sewing machine. A prototype was built to verify the concept. They used independent control of the two work-pieces that were separated by a thin plate. A vision system tracked patterns on the fabric. The implemented prototype used open loop path control and they emphasize feedback control for the sewing process.

Wetterwald et al. [6] built a sewing cell that was able to attach fiber and leather parts for use in recliners. It used a single robot and a sewing machine. For sensor feedback a triangulation-based edge tracking system was included to control the seam position on the part. The robot speed was synchronized with the sewing machine speed using an optical movement sensor.

The mentioned projects mainly focused on sewing systems using either a single robot or servo-based feeding mechanisms. The use of a single robot limits the use cases to sewing of parts with the same edge shape.

The work presented in this paper uses a concept based on two independently working robots in order to be able to control parts with different shapes that may result in 3D-shaped subassemblies. Preliminary experiments and synchronization between two parts were discussed in [7].

II. Hardware

A. Sewing Machine

The sewing machine is a DA195 industrial sewing machine providing individual feed rate for the upper and the lower part. The controller of the machine is connected to a development board that provides access to the functions of the sewing machine via Ethernet. These functions are control of the feed dog (up/down), the thread cutter, the needle position and the drive motor frequency. The sewing speed is determined by the variable drive frequency and the mechanically adjustable stitch length. It is important to mention that the feeding speed of the parts is highly dependent on the fabric characteristics, in particular the thickness and stiffness, meaning that the actual material feed rate will differ from the set theoretical material feed. An overview of the sewing cell with the sewing machine is shown in Fig. 2.

B. Lightweight Industrial Robots

Two 6-axis industrial manipulators are used to hold the parts and to control their position during the sewing process. The robots are Universal Robot UR-6-85-5-A. Finger grippers are mounted on a tool changing system at the robot tool flange. The grippers are designed such that they can reach over the table, see Fig. 3.

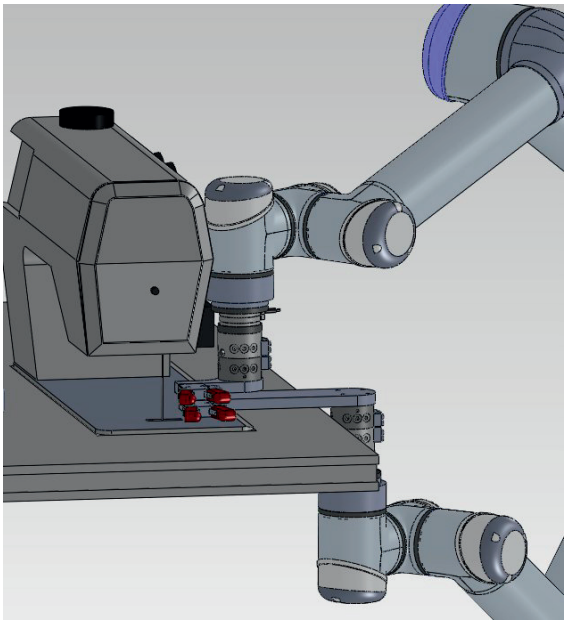


Fig. 3. The robots with finger grippers.

C. Force Sensors

Force sensors are mounted on the robot to measure the force vector at the grip point towards the sewing machine. The sensors are ATI Mini45 sensors that are connected to Net/FT boxes. These provide access to the sensor data via Ethernet.

D. Edge Sensors

To detect the position of the edge near the needle, a sensor system based on two line sensor arrays is installed in front of the presser foot, see Fig. 4. It is installed such that the sensors are placed between the two parts during the sewing process. The sensor arrays act as one-dimensional cameras and can detect the light falling onto the sensor. An algorithm is implemented to detect the position of the edge on the sensor. LEDs are used to control the background light. The sensors are connected to development boards that provides an Ethernet interface.

III. Software and Communication

A. Robot Interface

Two different interfaces are provided by the robot manufacturer to communicate movement commands to the robots:

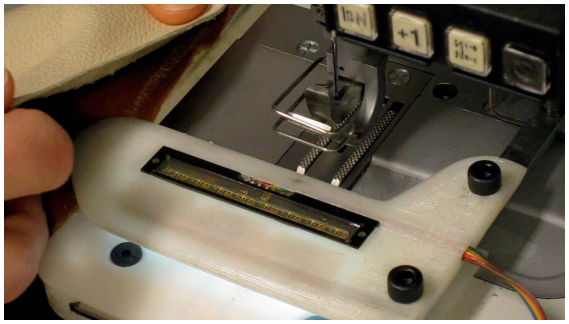


Fig. 4. The edge sensor system. One part is placed under the sensor plate, the other part above.

- *secondary client interface* and
- *UR_COMM_API*.

The *secondary client interface* is an interface to the high level controller in the Universal Robots controller. It exposes the same functionality as provided to programs that are created using the user interface on the robot. In contrast to programming directly on the teach pendant, it is possible to send the commands over a TCP socket to the robot while the robot is running. Status packets are sent on the same TCP socket from the robot to the external controller including joint angles, velocities, Cartesian tool coordinates, currents, etc. Using this interface it is possible to create an external controller that interacts with sensors and controls the robot with a frequency of about 10 Hz.

The second interface used in the sewing cell is the *UR_COMM_API*, a C-library that provides access to the low-level controller of the robot. The controller system of the UR-6-85-5-A is based on a Linux PC that runs different controller facilities, for example the GUI, the kinematics and dynamics and the low-level access to the servo controller. By using the API it is possible to run a custom program on the controller gaining access to the low-level controller. In a loop of 8 ms it is possible to read the current joint angles and angular velocities and to send new joint set points $(q, \dot{q}, \ddot{q})$ to the controller. The main benefit of the API is the high update frequency of 125 Hz.

B. Motion Controller

In order to control the robots from the PC, connector programs were designed to connect to the TCP sockets of either the *secondary client interface* or the router program that uses the *UR_COMM_API*. In case of the *secondary client interface*, the functions for robot movement, i.e. tool velocity control and tool linear control can be used directly from the PC, while in case of the low-level interface the kinematics calculations have to be done on the external PC.

A kinematics controller has been programmed based on Orocos KDL. It offers a *tool linear trajectory generator* that moves the robot linearly in the base coordinate system to a given pose and a *tool velocity trajectory generator* that moves the tool with a given twist.

C. Force Filtering

Due to the high update frequency of the force measurement, it is possible to record the force characteristics in the part during each stitch while sewing. Since the feeding is not linear, the force curve follows a periodical shape during sewing. For the control algorithms it is not desired to follow the high-frequency movement of the feeding system, but to measure the mean force over the period of one stitch. A force filter was implemented using a running average with a time window of one stitch.

D. Middleware

ROS is used as middleware in the sewing cell [8]. ROS is a framework that provides functionality for communication between different nodes using services and topics. A service is similar to a remote procedure call in the way that the calling node sends a request to another node and gets a response. Topics are used to publish data from a publisher to one or more subscribers. Both services and topics can be used to communicate different data types between nodes. Additionally ROS provides several tools and a packet system to easily include 3rd party nodes and libraries into the system.

The sensor and robot connectors in the sewing cell are encapsulated in ROS nodes that can run on different PCs and are connected through ROS via Ethernet.

The main nodes in the sewing system are:

- ***ati_net_ft*** - the interface to the force/torque sensor
- ***edge_sensor_server*** - the interface to the edge sensors
- ***edge_finder*** - the edge detection algorithm
- ***ur_rt_connector*** - the interface to the robot
- ***sewing_machine_controller*** - the controller for the sewing machine
- ***sewing_process_controller*** - process controller including force- and edge-controller
- ***sewing_process_manager*** - the main application

The *sewing_process_manager* and the *sewing_machine_controller* are common nodes for the system. All the other nodes are duplicated, hence there is one corresponding node for the upper robot and one node for the lower robot.

E. Process Control System

The sewing cell includes two process controllers for each robot:

- an edge controller to control the edge position at the needle and
- a force controller to hold constant tension in the part.

F. Control Coordinate system

In order to ensure that the edge controller and the force controller can act independently, and with as little influence on each other as possible, a control coordinate system was defined. This coordinate system describes the position of the robot relative to the needle. To compensate for displacement of the edge in front of the sewing machine, the part is rotated around the needle. To compensate for force error, the robot tool is moved towards or away from the needle, see Fig. 5.

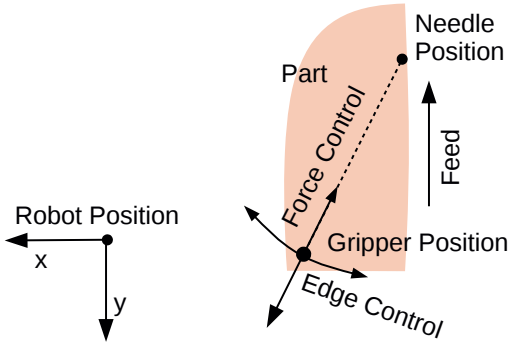


Fig. 5. The control coordinate system. Edge control results is a rotation around the needle while force control moves the robot towards or away from the needle. These movements are transformed into the robot base coordinate system.

The output of the force controller and the edge controller is a tool speed in force and edge direction. This velocity vector is then transformed to the robot base coordinate system. The robot command is calculated from the control velocities of the force and edge controller, and the feed-forward speed of the sewing machine.

G. Force Control

Both the elasticity in the part and the non-predictable feeding speed lead to varying tension in the part and inaccuracy in the predicted sewing speed. A constant force has to be applied to the fabric to ensure stable edge control. Too low tension leads to wrinkling and, hence, inaccurate edge-measurements due to a possible gap between the sensor and the part. Too high tension increases the friction between the part and the sensor housing, leading to a stick-slip effect; especially in the initial phase when the part is not moving in the sewing feed direction. Too high tension can also lead to lower seam quality due to disturbance of the feeding system of the sewing machine.

The tension in the part is measured by the force sensors on the tool flange. The control scheme is depicted in Fig. 6. A PI controller is used to control the force. Due to the delay in the system, a derivative feedback is neglected. Due to the reducing distance between the grip point and the needle point the mechanical system of the fabric increases stiffness as the sewing process progresses. The control parameters are scaled with the distance to compensate for this behavior.

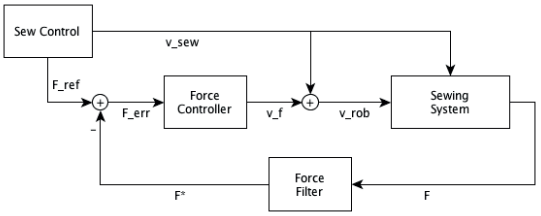


Fig. 6. Force controller for one robot.

For future synchronization between the two robots, it is optimal to measure the tension in the fabric along the sewing direction. However, due to the current gripper design it is necessary to grip the parts at an offset from the seam line such that it is possible to control the seam until the last stitch without having to release the grip of the part; i.e. to allow for the grippers to progress with the part past the sewing machine. As previously described, the edge control is done by a rotation around the needle. The force controller acts perpendicularly to the edge controller by controlling the force on the line between the grip point and the needle. Hence, the two controllers should work independently. A coupling was observed due to the fact that changes in the force have influence on the pressure of the part on the sensor, which leads to changes in the measured edge position.

H. Edge Control

Edge control is necessary to control the distance of the seam from the edge of the part. To control the edge, the part is rotated around the needle. The edge error is measured by the edge sensor system that is installed in front of the needle. The controller is depicted in Fig. 7. It consists of a simple P controller. A derivative feedback is not implemented due to the delay in the system. An integral feedback is also neglected in the current system due to mechanical issues that can lead to a temporary sticking of the part on the sensor plate or sewing table. Since the output of the P controller is a tool velocity, the controller keeps moving the robot until the part slips free. An integral feedback can easily lead to instability since it increases the velocity further, leading to a large oscillation; this effect has been experienced.

IV. Experiments

A. The Sewing Process

A sewing process is executed in different stages:

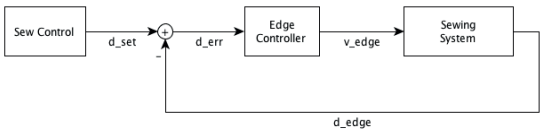


Fig. 7. Edge controller for one robot.

1) Move the robots to the starting positions.
2) Enable compliance control of the robots for operator-assisted fixing of parts.
3) Start force control to achieve the right tension in the part.
4) Start the edge control to adjust the parts to the set seem allowance (edge position).
5) Start the sewing process

B. Sewing of one Part with Different Robot Interfaces

Experiments have been conducted to compare the *secondary client interface* and the *UR_COMM_API*.

The frequency of the sewing machine was set to 12 Hz, which leads to a sewing speed of approximately 50 mm/s. Only the upper robot was used in order to exclude influences from the second part. The P and I parameter for the force controller were tuned to the corresponding robot interface. The force set point was set to 2 N. The update frequency for the robots were 10 Hz and 125 Hz. The log frequency of the sensors was the same as the update frequency.

The filtered force value and the edge error for the two interfaces are shown in Fig. 8 and 9. The plots show the measurements during a typical sewing process starting after the initial force and edge adjustment. The measurement of the last part of the sewing process (approximately 5−10 cm) are not shown since the robot is not feedback controlled due to invalid measurements after the part has passed the edge sensor.

It is important to mention that the actual edge error is lower than the measured error since a rotation around the needle results in a larger movement at the sensor than at the point of the next stitch.

It is evident that the controller using the *UR_COMM_API* controls better to the set point and has a comparable edge error. The controller using the *secondary client interface* has problems to reach the force set point even with a PI controller. This is caused by the limit of the P and I contribution of the controller due to the dead time in the system.

C. Sewing with two Parts

This experiment demonstrates sewing with two parts. As controller interface, the *UR_COMM_API* was used. The force and edge error measurement for the two robots are shown in Fig. 10 and 11.

The plots show that the behavior of the sewing of two parts has no considerable influence on the sewing

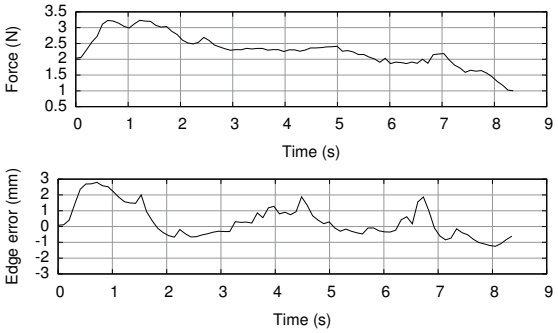


Fig. 8. Sewing force and edge error for a typical sewing process using the *secondary client interface* with a single robot.

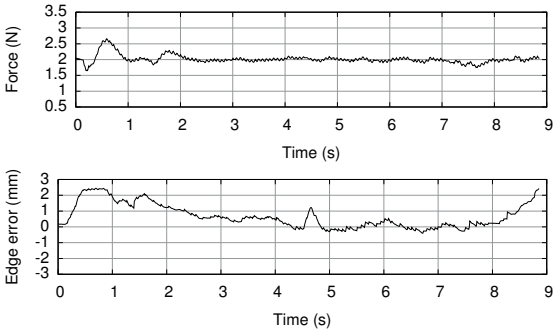


Fig. 9. Sewing force value and edge error for a typical sewing process using the *UR_COMM_API* interface with a single robot.

quality compared to sewing of a single part with the same robot interface. The sign of the edge error of the lower robot is opposite to that of the upper edge measurement because of the reversed physical mounting of the lower sensor. Not visible in the plots, but visible when examining the sewn part is that the lower part is fed slightly faster into the sewing machine than the upper part which results in a displacement of the corner, see Fig. 12. This error is systematic and can be met with synchronizing the two robot systems with a superior controller that adjusts either the control parameters or the feeding parameters in the sewing machine. This issue is theoretically addressed in [7].

V. Conclusion and Future Work

A. Conclusion

This paper has presented details of a flexible, automated sewing cell, which it is able to sew together parts of different shapes and materials. Sensor feedback is used to control the sewing force in the parts and the location of the seam on the parts. The system controls the process in real-time without prior knowledge of the part geometries. The system includes two robots that can handle the two parts independently. Tests have been conducted to evaluate the quality of the seam

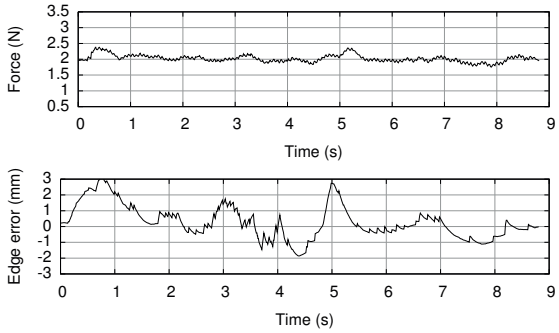


Fig. 10. Sewing force value and edge error for the upper robot for a typical sewing process using the UR_COMM_API interface with two robots.

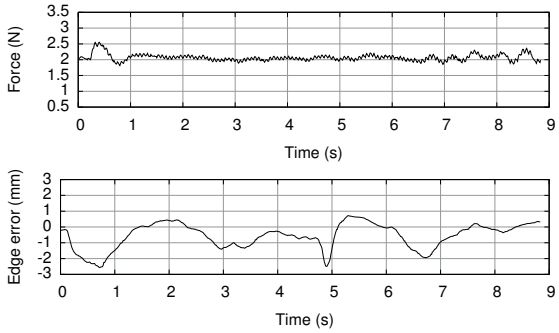


Fig. 11. Sewing force value and edge error for the lower robot for a typical sewing process using the UR_COMM_API interface with two robots.

for the cases of a single robot sewing or both robots sewing. Two different robot interfaces with update frequencies of 10 Hz and 125 Hz have been used and the results of the control errors were compared. The tests show that the faster interface results in a smoother and more responsive control of the sewing force. The results for the edge control were comparable. Both control methods result in comparable sewing result when investigating the sewn objects visually, but the higher update frequency may be necessary when introducing an adaptive control strategy to match reference points along the seam path. It was also shown that the behavior of the control system was independent of sewing with one part and one robot or two parts with both robots.

B. Future Work

To introduce path matching into the system a synchronization of the two robots has to implemented. Preliminary tests and experiments are discussed in [7].

Another focus for future work is an automatic feeding system to automate the whole process including arrangement and handling of material.

Fig. 12. The assembly of two parts as sewn in the experiment with two robots. The seam runs from left to right and is nicely placed at a nearly fixed distance from the edge. At the end of the seam an accumulated error of the feeding can be observed as a longitudinal displacement between the corners of approximately 10 mm.

VI. Multimedia Contents

A video was submitted as appendix for this paper. It shows sewing of different parts. It consists of five different parts:

1) Sewing of two parts with the same shape as in the experiments.
2) Sewing of two parts with different shapes.
3) Sewing of two identical parts with attached fiber.
4) Sewing of a curved path to test the edge control.
5) Time lapse video of five subsequent sewing processes (10x speed).

VII. Acknowledgements

The authors wish to thank the SFI Norman programme and the *Automated 3D Sewing* project founded by the Research Council of Norway. The authors also want to thank Ekornes ASA for close cooperation.

References

[1] D. Gershon and I. Porat, "Vision servo control of a robotic sewing system," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, apr 1988, pp. 1830 –1835 vol.3.

[2] D. Gershon, "Strategies for robotic handling of flexible sheet material," *Mechatronics*, vol. 3, no. 5, pp. 611 – 623, 1993.

[3] G. Seliger and J. Stephan, "Flexible garment handling with adaptive control strategies," in *Proceedings of the 29th ISR*, 1998, pp. 483 – 487.

[4] P. Koustoumpardis and N. Aspragathos, "Robotized sewing of fabrics based on a force neural network controller," in *Intelligent Robotics and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, vol. 7101, pp. 486–495.

[5] R. Winck, S. Dickerson, W. Book, and J. Huggins, "A novel approach to fabric control for automated sewing," in *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, July 2009, pp. 53 – 58.

[6] L. E. Wetterwald, S. Dransfeld, H. Raabe, and T. Ulleberg, "Flexible robotic sewing with real time adaptive control," in *Proceedings 2nd Conference on Assembly Technologies and systems (CATS 2008)*, 2008.

[7] J. Schrimpf and L. E. Wetterwald, "Experiments towards automated sewing with a multi-robot system," in *International Conference on Robotics and Automation (ICRA 2012)*, May 2012.

[8] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

6.7 Real-Time Robot Trajectory Generation with Python

M. Lind, L. Tingelstad, and J. Schrimpf, *Real-time robot trajectory generation with python*, IROS2012 Workshop on Robot Motion Planning: Online, Reactive, and in Real-time, 2012

Declaration of co-authorship

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. Lars Tingelstad has been actively using and testing PyMoCo in near-industrial installations. The evolution of the presented framework has taken place over several years and is based on intensive discussions and consensus between Morten Lind and Johannes Schrimpf.

The general knowledge base regarding the use of Python on Linux for real-time robot control was cooperatively achieved over several years by Morten Lind and Johannes Schrimpf.

The presented experiments were conducted by Morten Lind.

The paper was written, prepared, and submitted by Morten Lind with contribution of Lars Tingelstad. Johannes Schrimpf contributed with reviewing of the final version. All graphics were designed and produced by Morten Lind.

The publication was presented in an interactive workshop by all three authors.

Real-Time Robot Trajectory Generation with Python*

Morten Lind[1], Lars Tingelstad[1] and Johannes Schrimpf[2]

Abstract— **Design and performance measurements of a framework for external real-time trajectory generation for industrial robots is presented. The framework is implemented entirely in Python. It serves as a proof of concept for performing real-time trajectory generation in Python, from a PC with connection to the motion controller in an industrial robot controller. Robotic applications requiring advanced, custom trajectory generation, and a high level of integration with sensors and other external systems, may benefit from the efficiency of Python in terms of reduced development time, lower code complexity, and a large amount of accessible software technologies.**

The presented framework, dubbed PyMoCo, supplies a set of simple trajectory generators, which are comparable to those found in contemporary industrial robot controllers. Designing and implementing new trajectory generators and integrating or extending the included trajectory generators is central to the design of PyMoCo. Laboratory applications involving real-time sensor- and vision-based robot control has demonstrated the usability of PyMoCo as a motion control framework and Python as a robotics application platform. For robotics applications with a control frequency not exceeding a couple of hundred Hz, computation deadlines no shorter than some couples of milliseconds and jitter tolerance at the order of a millisecond, PyMoCo may be considered a feasible and flexible framework for testing and prototype development.

I. Introduction

Robotic tasks of limited complexity such as simple positioning tasks, trajectory following or pick-and-place applications in well structured environments, are straightforward to develop and integrate in the application platform of the native robot controller using current commercial robot control software (*de Schutter, et al. (2007)* [1]).

If robots communicate or interact with other robots or systems, the implementation is most often based on vendor-specific proprietary protocols and with limited performance specifications that preclude online sensor-based control (*Decré (2010)* [2]). However, there is a strong market pull for more flexible and cost effective robotic systems which are able to integrate a multitude of sensors and operate in unstructured environments. An example of this is the increased use of industrial robots in small and medium-sized manufacturing enterprises, often characterized by a combination of low-volume, high variety, and custom-made goods (*EURON (2005)* [3]). In order to meet these requests from the industry, new methods for programming and system integration are needed.

Many research laboratories therefore attempt to circumvent the application platform of the native robot controller, which either precludes real-time interaction or does not offer an appropriate set of technologies for solving the pertinent problem, in order to directly interface the motion control level. The motion control level is described as the 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 100 Hz to 1 kHz. The ability to address the motion control level from an external application platform may thus give full control of choosing hardware peripherals, programming software and control algorithms (*Decré (2011)* [2]). The motion control level is often referred to as *low-level control* in literature.

A. Related Work

Applications that utilize low-level interfaces, to the motion control level, 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 and RTAI+Linux. The obvious reasons for these choices are among requirements to hard real-time performance; efficiency of computation with short cycle times; and latency tolerance on the time scale of microseconds.

Dallefrate et al. (2005) [4] used RTAI+Linux to control the Mitsubishi PA10 robot at the motion control level in 1 kHz over Arcnet.

Kubus et al. (2010) [5] modified Stäubli controllers and gained external joint level position control rates of 10 kHz and 250 Hz from a QNX system on a standard PC.

Buys et al. (2011) [6] present a teleoperation setup using two KUKA Light-Weight Robots (LWR) coupled to a Willow Garage Personal Robot (PR2). The two KUKA LWR robots are controlled over the KUKA Fast Research Interface (FRI) (*Schreiber et al. (2010)* [7]) for the KUKA KRC2LR industrial controller from an external control unit running RTAI+Linux. The communication is based on the UDP protocol and has a configurable communication rate of up to 1 kHz. The application was integrated using the two component based robotic frameworks OROCOS (Open Robot Control Software) (*Bruyninckx (2001)* [8] and *Bruyninckx et al. (2003)* [9]) and ROS (Robot Operating System) (Quigley et al. (2009) [10]).

A contemporary overview of the directly available low-level accessibility in some industrial robot controllers can be

*The work presented has been financially supported by the The Research Council of Norway through the research programmes "SFI Norman", "BIA Robust, industriell sømautomatisering" and "KMB Next Generation Robotics".

[1]Department of Production and Quality Engineering, Norwegian University of Science and Technology, Trondheim, Norway

[2]Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway

e-addresses: {morten.lind, lars.tingelstad, johannes.schrimpf} at ntnu.no

found in *Kröger and Wahl (2010)* [11].

B. Motivation and Goals

The work underlying this paper is motivated by the desire for making quick prototype development of real-time, sensor-based robotics applications in a laboratory setting with industrial robots. Our main application domain is industrial manufacturing automation, and all laboratory projects involve industrial robots for various types of tasks, ranging from standard offline programmed robot control to sensor-based real-time trajectory generation.

The presented work started out as a simple need for experimenting with motion control interfacing, and developed into the robot control framework we call PyMoCo. When developing real-time robotic applications, there are many demanding issues involved. We aim at addressing two of these:

- Maintenance and knowledge of specialized real-time operating systems and platforms (hardware and software).
- Development of C/C++ applications on real-time enabled software frameworks or platforms.

The goals of the presented work were to establish a sufficiently stable real-time framework which is:

- based on a stock GNU/Linux kernel and a freely available operating system,
- and using a high-level scripted programming language in pure user-mode.

Obtaining these two goals may have driven our development away from supplying directly usable industrial solutions. On the other hand it has been the enabling factor for having many researchers as well as projects making progress in advanced sensor-based robot control applications.

The specific choices of using stock Real-Time Linux[1] kernels with the Debian/Ubuntu operating systems and Python as the programming language were well-considered in terms of previous experiences and expertise.

As will be demonstrated later, see Section III, there is not much effect on the performance from using the Real-Time Linux kernel compared to using a standard Linux kernel. The major concern towards real-time performance regards the Python run-time efficiency and the implementation of PyMoCo. While there exist a possibility, however remote, that Real-Time Linux may some day guarantee an upper bound to latency, the Python run-time system in its current form, and possibly far into the future, does not possess hard real-time quality.

The efficiency of using Python as a development language, and even as an end-target platform has been well known for some time (*van Rossum (1998)* [12]). Further, the general scientific computational performance of Python is well document by many papers and projects; see e.g. the comprehensive paper by *Cai et al. (2005)* [13].

It is the purpose of this paper to give an overview of PyMoCo at the design and architectural level and to convey an impression of its level of feasibility as a software technology for real-time trajectory generation in prototype development of sensor-based applications of industrial robots.

[1] `https://rt.wiki.kernel.org/`

C. Paper Outline

The remainder of this paper is outlined as follows. An overview of PyMoCo is presented in Section II, performance test setup and results are presented and discussed in Section III, and general discussion and mention of further work is presented in Section IV.

II. PyMoCo Overview

PyMoCo is a free and open source[2] software framework implemented entirely in Python, using the efficient NumPy[3] library for numerical computations. This section gives an overview of the architectural structure of PyMoCo.

The development of PyMoCo has been proceeding over the past five years and by now amount to some 4500 lines of Python source code[4]. It includes back-ends to two different robot types: A software-modified Universal Robots[5] controller and hardware-modified Nachi Robotics AX10 and AX20 controllers.

A. Applications

Though PyMoCo is a work in progress it has played a central role in many manufacturing automation prototype projects at our research laboratories.

The dual robot, real-time sensor-based sewing cell described by *Schrimpf et al. (2012)* [14] has a setup that uses PyMoCo trajectory generators.

Lind (2012) [15] used PyMoCo in the development of a joint offset calibration method for industrial robots.

Tingelstad et al. (2012) [16] used PyMoco for a tight tolerance compliant assembly task of critical aero engine components.

Schrimpf et al. (2011) [17] used PyMoCo for a real-time sensor-based control system with multiple sensors in a line-following application.

Lind and Skavhaug (2011) [18] used PyMoCo's ToolLinearController trajectory generator intensively for a real-time emulated production system setup involving several robots.

B. Architecture and Design

The PyMoCo run-time provides three core interfaces to the trajectory generation and application level systems. These are described in the following.

1) RobotDefinition Interface: is a placeholder for all static information about the robot in use. It provides such information as static link transforms; joint transform parameters; translators between different joint spaces: actuator, encoder, and serial; the home pose of the robot; and it is a factory for a set of joint transform function objects for the robot.

[2] PyMoCo can be branched from Launchpad: `https://launchpad.net/pymoco`

[3] `http://numpy.org/`

[4] Measured using David A. Wheeler's 'SLOCCount' `http://www.dwheeler.com/sloccount/`.

[5] `http://www.universal-robots.com/`

2) FrameComputer Object: is the computational entity for all kinematics computation. It is currently a single, unspecialized class for unified kinematics computation for all robot structures. For joint transform objects and static link transforms, it relies on information retrieved from the RobotDefinition interface at construction time.

3) RobotFacade Interface: is the main interface covering the robot specific backend subsystem. Ultimately, in the backend subsystem, there is a connection to the motion controller entity of the operating robot. At any time, some trajectory generator must be answering real-time requests propagated from the robot motion controller through the robot facade subsystem.

For illustrating the relationships of entities in setups for real-time trajectory generation and robot application control involving PyMoCo, two UML object diagrams are shown and described in the following.

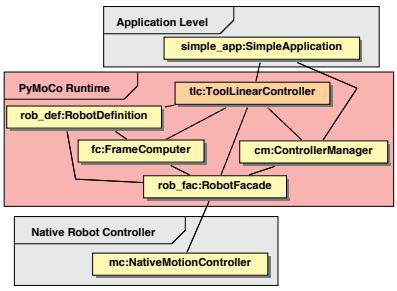


Fig. 1. UML object diagram giving an overview of a simple PyMoCo application, utilizing built-in trajectory generators managed by a ControllerManager object from PyMoCo.

The most simple runtime setup using PyMoCo for robot control, illustrated by the UML object diagram in Fig. 1, uses an object of the ControllerManager class, included with PyMoCo. The ControllerManager class is managing the switch of trajectory generators at the request of the application code, ensuring that the switch will not skip a control cycle request from the motion control level.

In the diagram in Fig. 1 weak or temporary associations are represented by dashed lines and more persistent object associations are illustrated by solid lines. The trajectory generator is exemplified by an object of the ToolLinearController class. It uses the core PyMoCo entities and provides its operational interface to a simple application; which is not specified by PyMoCo. The simple application, developed and provided by the user, thus only has to interface with the ControllerManager object and the trajectory generator objects that it requests from the controller manager.

Fig. 1 also indicates a layered structure, where the native robot controller containing the motion controller is lowest, the PyMoCo run-time system is in the middle, and the application level at the top. In a simple setup as the one illustrated, PyMoCo may be considered more as a software service than a software framework, since the client system, i.e. the simple application, is cleanly separated from the PyMoCo code.

The specific set of trajectory generators that are managed by the controller manager are the ones supplied with PyMoCo, and they will be discussed shortly in Section II-C.

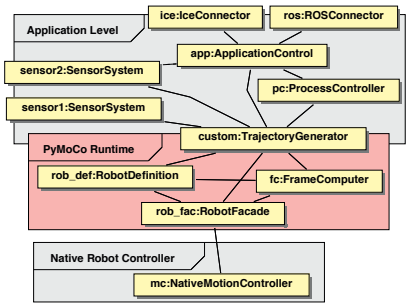


Fig. 2. Overview of an advanced PyMoCo application, utilizing the core PyMoCo objects and implementing custom trajectory generators with PyMoCo resources.

A more advanced, and realistic setup for sensor-based real-time trajectory generation, is illustrated in Fig. 2. It shows an application control at the application level which is strongly integrated with network communication systems, illustrated by connectors over ZeroC Ice™ (*Henning (2004)* [19]) and ROS; process control; sensor systems which naturally connect externally; and with a custom trajectory generator. The custom trajectory generator is developed using the PyMoCo software framework resources and takes on the real-time obligations toward the pertinent robot motion controller through the robot facade.

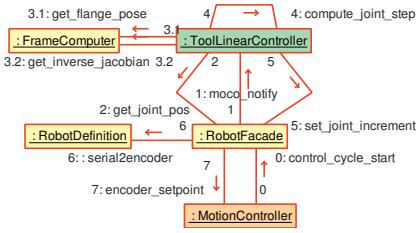


Fig. 3. The real-time cycle illustrated as a UML collaboration diagram among core PyMoCo entities, the robot motion controller, and a trajectory generator (of the class ToolLinearController).

The detailed mechanisms of the control cycle involving the core elements of PyMoCo may be perceived from the UML collaboration diagram in Fig. 3. The focus here is on the computational real-time cycle from the trajectory generation level and down, and hence the application logic and control is not included. The MotionController class is not a real class, but included for representing the motion controller in the native robot controller. The trajectory generator used for illustration here is, again, of the class ToolLinearController.

The control cycle is started by a notification from the motion controller to the robot facade in PyMoCo; typically

containing a lot of status information such as encoder readings, velocities, etc. The robot facade propagates the notification internally to a subscribing trajectory generator, which then, using PyMoCo run-time facilities, computes a control step in response. This control step is returned in serial joint kinematics coordinates to the robot facade, which translates it to encoder values and then in turn responds to the motion controller.

C. Included Trajectory Generators

A (real-time) trajectory generator is an entity which ultimately carries the real-time responsibility of timely responding to the motion controller request for a new control-setpoint in joint space; or rather the joint encoder space. Neither the motion controller or the trajectory generators are core PyMoCo entities.

PyMoCo includes a set of simple trajectory generators. They cover the typical trajectory generators that are represented by motion commands in the application platforms of standard industrial robot controllers. None of the included trajectory generators implement any advanced strategies for dealing with arm configuration singularities, joint speed or acceleration violations, joint limits, or other types of circumstances that may lead the motion control system to fail. Self-motion, or internal, singularities are dealt with by using a configurable singular value cutoff in the inverse Jacobian computation; which is probably the simplest possible strategy.

The most common trajectory generators in standard robot controller are included: joint space linear motion, tool space linear motion, and real-time correction-responsive tool space linear motion. An additional two real-time responsive trajectory generators are included, which are rarely found in standard robot controllers, but immensely useful in real-time sensor-based robot control: tool space velocity motion and joint space velocity motion. The tool space velocity generator is the most frequently used in real-time sensor-based robot control applications at our laboratories.

III. Real-Time Performance

The high flexibility and versatility of Python as an application platform for robot control, and as the implementation language of PyMoCo alike, come at the cost of computational performance and real-time quality. The real-time performance of a PyMoCo-based application is thus crucial to investigate. It is the outcome of such an investigation which will clarify whether PyMoCo is usable and feasible, and, if at all, for which applications and robots.

This section presents results of an experimental setup based on the Universal Robots controller. The Universal Robots UR5 robot is used extensively in our laboratories, since it may be externally controlled and exhibits fairly low control delay and short motion response time; see *Lind et al. (2010)* [20].

Schrimpf et al. (2012) [21] compares three different setups for real-time trajectory generation; one of which is PyMoCo and the others based on OROCOS kinematics. Though their experiments are performed on one PC using local loop back networking, and thus do not measure the over-the-wire performance, the comparison is instructive. The purpose of the experiments presented in this section is different, in that it aims at making absolute, over-the-wire, realistic performance tests that are valid for PyMoCo-based trajectory generation applications.

A. Experiment Setup

The motion controller in the Universal Robots controller is interfaced at $125\,\mathrm{Hz}$, i.e. a control period of $8\,\mathrm{ms}$, and requires a response in $4\,\mathrm{ms}$. In the real controller, the native application platform and trajectory generator can be shut down, and a custom "router" application started. This router application listens for external connections over TCP, and mediates contact with the motion controller internally in the robot controller. The router application, representing the motion controller, can be emulated on an ordinary PC, the purpose of which it is to log the response times from a PyMoCo application running off another PC and connecting through a switch.

All hardware used is consumer grade and not of highest performance. Two PCs, both with an Intel i7 processor are used for performance measurements, connected through a standard $100\,\mathrm{Mbit\,s^{-1}}$ switch, and using the on-board Ethernet cards. The most important hardware to detail is the PC running the PyMoCo application. It is an Intel i7-860 processor running at $2.80\,\mathrm{GHz}$ with four cores and two threads per core.

Both PCs use the stock GNU/Debian Linux systems with Preempt-RT patched kernels of version 3.2.0-3-rt-686-pae; i.e. Real-Time Linux kernels. The most important software versions to mention are Python, 2.7.3rc2, and NumPy, 1.6.2-1. All software and kernels involved are taken from the official Debian testing repositories[6].

Starting from a standard Debian desktop installation, a checklist of simple tweaks to ensure the best possible real-time performance was followed:

1) Switch to single user mode. (`$ telinit 1`)
2) CPU frequency scaling should be set to "performance". (`$ cpufreq-set -c [0..7] -g performance`)
3) Disable garbage collection in the Python code for the real-time critical computations. (`gc.disable()`/`gc.enable()`)
4) Put the control process in a real-time scheduler queue. (`$ chrt 99 ...`)
5) Run the RT-critical processes from a remote login-shell. (`$ ssh ...`)
6) Boot the Real-Time Linux kernel.

All experiments were conducted at a length of $100\,000$ samples, which at $125\,\mathrm{Hz}$ amounts to about $13\,\mathrm{min}$ running time.

[6] `http://ftp.debian.org/debian/dists/testing/`

B. Best Condition Performances

The most important experiments were to measure the inherent response time of the PyMoCo run-time, by using the ZeroVelocityController, and to performance test the two most useful of the included trajectory generators: ToolVelocityController and ToolLinearController. All experiments were executed under the best obtainable real-time conditions, as per the check list in Section III-A.

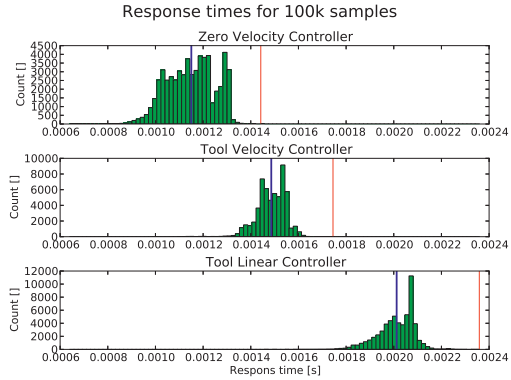


Fig. 4. Response time distribution for three different controllers. Average response time is marked by a vertical blue line and worst-case is marked by a vertical red line.

The results are visually observable from Fig. 4. Statistical summaries of the response time samples are shown in Table I.

Trajectory Generator	Worst [s]	Average [s]	Std. dev. [s]
ZeroVelocityController	0.00144	0.00115	0.00010
ToolVelocityController	0.00174	0.00149	0.00006
ToolLinearController	0.00236	0.00201	0.00008

TABLE I
STATISTICS OF MEASUREMENTS UNDER BEST REAL-TIME CONDITIONS.

These results show that the ToolLinearController is computationally much heavier than the ToolVelocityController; which was expected since it performs various checks along its path to control bounded acceleration ramp-up and ramp-down of the velocity. More importantly, the results also show that both of the usable controllers are well within the $4\,\mathrm{ms}$ response time required by the Universal Robots motion controller. The inherent response time of PyMoCo indicated by the worst-case response time of the ZeroVelocityController gives the impression of the availability of control computation time for any useful trajectory generator. In case of a required $4\,\mathrm{ms}$ response time, there is of the order of $2.5\,\mathrm{ms}$ time available for any trajectory generator in each control cycle.

C. DH-Kinematics Performance

PyMoCo has a native kinematics formulation which is flexible for specifying separately static link transforms and joint transform functions. However, a DH formulation of the kinematics is also supported, reducing the number of matrix multiplications in the forward kinematics computation. The DH formulation was used in one run with the ToolVelocityController under the same conditions as the ones used in Table I. The comparable statistical results are seen in Table II

Kinematics	Worst [s]	Average [s]	Std. dev. [s]
PyMoCo	0.00174	0.00149	0.00006
DH	0.00180	0.00151	0.00007

TABLE II
MEASUREMENT OF KINEMATICS IN DH FORMULATION.

It turned out that the DH formulation, contrary to the expected, was slightly inferior to the standard formulation in PyMoCo. This can be traced to NumPy being relatively inefficient in assigning matrix element compared to multiplying matrices.

D. System Tweak Performance Effects

The last experiments addressed the effects of individual omission of the various real-time enhancement tweaks, shortlisted in Section III-A. Results are given in Table III.

Tweak	Worst [s]	Average [s]	Std. dev. [s]
All tweaks	0.00174	0.00149	0.00006
- Single user	0.00249	0.00206	0.00009
- CPU freq. sched.	0.00265	0.00212	0.00007
- Disable GC	0.00223	0.00156	0.00009
- RT scheduling	0.00203	0.00155	0.00005
- RT kernel	0.00183	0.00125	0.00005

TABLE III
EFFECT OF VARIOUS TWEAKS ON REAL-TIME PERFORMANCE.

It is observed from the table that all tweaks have significant effect on the worst-case performance. The lower average and higher worst-case response times of the standard kernel are natural, since the low-level real-time enhancements in the real-time kernel sacrifice some computational efficiency for gaining lower worst-case latency. The fact that the performance difference between a standard and a real-time kernel is so low is evidence of the flow of the real-time patches into the mainline Linux kernel over the recent years.

IV. DISCUSSION AND FURTHER WORK

This paper has presented an overview of the structure of PyMoCo, a flexible, Python-based software framework for trajectory generation and motion controller interfacing.

Various real-time performance experiments for assessing its usability have been conveyed and the results have been presented and discussed. Under the presented experiment conditions, in terms of hardware, software, and system setup, it can be inferred that PyMoCo may be a usable software technology for trajectory generation in robot control applications where the over-the-wire response time limit is no lower than about some $3\,\mathrm{ms}$.

The main contribution of PyMoCo is to provide users with a very flexible framework for building real-time sensor-based robot control applications at the laboratory prototyping stage. Many laboratory prototyping projects have already utilized PyMoCo, and it is considered good for learning and fast prototyping. However, being tied to the Python language and the Python run-time platform, it has no outlook of becoming industrially real-time reliable.

The computational performance of contemporary CPUs together with the current implementation and design of PyMoCo is the limiting factor for its use in various setups. For instance, it is currently precluded that a KUKA LWR could be controlled by a PyMoCo-based application over FRI with maximum control rate. However, with CPU performance increasing over time, such setups may be achievable for PyMoCo in a not too distant future.

Notwithstanding the automatic performance gains of future CPUs, there are a whole range of possibilities for increasing the inherent performance of a pure Python application. These range from downright porting of functional code to C/C++ extension modules, whereby some flexibility may be lost; over Cython (*Behnel et al. (2011)* [22]) for automated translation and compilation of computationally critical code blocks; with PyPy, a very fast re-implementation of the Python run-time; to simply using more optimal and specialized technologies within PyMoCo, e.g. integrating PyKDL for kinematics computations as demonstrated by *Schrimpf et al. (2012)* [21].

Among useful and functional features that will be addressed in the future work with PyMoCo are facilities for trajectory blending. The methods described by *Lloyd and Hayward (1993)* [23] and *Volpe (1993)* [24] are under consideration.

References

[1] J. D. Schutter, T. D. Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, May 2007.

[2] W. Decré, "Optimization-Based Robot Programming with Application to Human-Robot Interaction," Ph.D. dissertation, Katholieke Universiteit Leuven, 2011.

[3] Euron, "Sectorial Report on Industrial Robot Automation," European Robotics Network, Tech. Rep., July 2005, http://www.euron.org/miscdocs/docs/euron2/year2/dr-14-1-industry.pdf.

[4] 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: http://www.realtimelinuxfoundation.org/events/rtlws-2005/ws.html

[5] D. Kubus, A. Sommerkorn, T. Kröger, J. Maaß, and F. M. Wahl, "Low-level control of robot manipulators: Distributed open real-time control architectures for stäubli rx and tx manipulators," 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. 38–45. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010

[6] K. Buys, S. Bellens, W. Decre, R. Smits, E. Scioni, T. D. Laet, J. D. Schutter, and H. Bruyninckx, "Haptic coupling with augmented feedback between two KUKA Light-Weight Robots and the PR2 robot arms," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, Sept. 2011, pp. 3031–3038.

[7] G. Schreiber, A. Stemmer, and R. Bischoff, "The Fast Research Interface for the KUKA Lightweight Robot," 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. 15–21. [Online]. Available: http://www.rob.cs.tu-bs.de/en/news/icra2010

[8] H. Bruyninckx, "Open Robot Control Software: the OROCOS project," in *International Conference on Robotics and Automation*, vol. 3. IEEE, 2001, pp. 2523–2528.

[9] H. Bruyninckx, P. Soetens, and B. Koninckx, "The Real-Time Motion Control Core of The Orocos Project," in *International Conference on Robotics and Automation*, vol. 2. IEEE, Sept. 2003, pp. 2766–2771.

[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009. [Online]. Available: http://www.willowgarage.com/sites/default/files/icraoss09-ROS.pdf

[11] T. Kröger and F. M. Wahl, "Low-level control of robot manipulators: A brief survey on sensor-guided control and on-line 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

[12] G. v. Rossum, "Glue It All Together With Python," in *Workshop on Compositional Software Architectures*, C. Thompson, Ed. Object Services and Consulting, Inc., Feb. 1998. [Online]. Available: http://www.objs.com/workshops/ws9801/papers/paper070.html

[13] X. Cai, H. P. Langtangen, and H. Moe, "On the performance of the Python programming language for serial and parallel scientific computations," *Scientific Programming*, vol. 13, no. 1, pp. 31–56, 2005. [Online]. Available: http://iospress.metapress.com/content/xawr0dx9xg61nb7q/

[14] J. Schrimpf, L. E. Wetterwald, and M. Lind, "Real-Time System Integration in a Multi-Robot Sewing Cell," in *International Conference on Intelligent Robots and Systems*. IEEE/RJS, Aug. 2012, accepted.

[15] M. Lind, "Automatic Robot Joint Offset Calibration," in *International Workshop of Advanced Manufacturing and Automation*, K. Wang, O. Strandhagen, R. Bjartnes, and D. Tu, Eds. Trondheim, Norway: Tapir Academic Press, June 2012.

[16] L. Tingelstad, A. Capellan, T. Thomessen, and T. K. Lien, "Multi-Robot Assembly of High-Performance Aerospace Components," in *IFAC Symposium on Robot Control*, 2012, accepted.

[17] J. Schrimpf, M. Lind, and G. Mathisen, "Time-Analysis of a Real-Time Sensor-Servoing System using Line-of-Sight Path Tracking," in *International Conference on Intelligent Robots and Systems*. IEEE/RJS, Sept. 2011, pp. 2861–2866.

[18] 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.

[19] M. Henning, "A New Approach To Object-Oriented Middleware," *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, Aug. 2004.

[20] M. Lind, J. Schrimpf, and T. Ulleberg, "Open Real-Time Robot Controller Framework," in *CIRP Conference on Assembly Technologies and Systems*, T. K. Lien, Ed. NO-7005, Trondheim, Norway: Tapir Academic Press, June 2010, pp. 13–18.

[21] J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, "Implementation Details of External Trajectory Generation for Industrial Robots," in *International Workshop of Advanced Manufacturing and Automation*, K. Wang, O. Strandhagen, R. Bjartnes, and D. Tu, Eds. Trondheim, Norway: Tapir Academic Press, June 2012.

[22] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The Best of Both Worlds," *Computing in Science Engineering*, vol. 13, no. 2, pp. 31 –39, Apr. 2011.

[23] 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.

[24] R. Volpe, "Task space velocity blending for real-time trajectory generation," in *International Conference on Robotics and Automation*, vol. 2. IEEE, May 1993, pp. 680–687.

6.8 Real-Time Analysis of a Multi-Robot Sewing Cell

J. Schrimpf, M. Lind, and G. Mathisen, *Real-time analysis of a multi-robot sewing cell*, IEEE International Conference on Industrial Technology (ICIT), 2013

Declaration of co-authorship

The sewing cell was designed by Johannes Schrimpf in close cooperations with Morten Lind and Lars Erik Wetterwald (not coauthor). The implementaton was done by Johannes Schrimpf and Morten Lind.

The control concept was designed and implemented by Johannes Schrimpf.

The real-time interface to the Universal Robots UR5 was programmed in close cooperation between Johannes Schrimpf and Morten Lind.

The real-time analysis was planned and designed by Johannes Schrimpf. The experiments were conducted by Johannes Schrimpf with occasional help of Morten Lind. The component diagram was prepared by Morten Lind based on discussions with Johannes Schrimpf. The analysis was mainly done by Johannes Schrimpf. Morten Lind contributed in ocassional discussions. The plots were prepared by Johannes Schrimpf and Morten Lind. The block diagram for the control system was prepared by Johannes Schrimpf.

The paper was written, prepared and submitted by Johannes Schrimpf. Geir Mathisen contributed during the planning phase of the paper. Morten Lind and Geir Mathisen contributed during the final review of the paper.

Comments

- In Section III, the maximum delay shown in the plots is from the force sensor to the trajectory generator 10.7 ms instead of the described 11.5 ms, and from the edge sensor to the trajectory generator 21.0 ms instead of the described 21.5 ms.

Real-Time Analysis of a Multi-Robot Sewing Cell

Johannes Schrimpf*, Morten Lind**, and Geir Mathisen*

Abstract—**This paper presents a sewing cell based on an industrial sewing machine and two robots to control the sewing operation in real-time. The software architecture and the communication structure are presented. Both software and hardware were chosen to build a flexible and highly available system suitable for prototyping. The focus is on analyzing the real-time characteristics of the control system. Experiments were conducted to measure the delays in different parts of the system in order to gain an understanding of the real-time performance and to show that the chosen system is capable to make use of the robot's high update frequency and low tracking delay.**

I. Introduction

Industrial sewing is a challenge for automation and is still dominated by manual labor, especially in low-volume productions. One challenge is the handling of the non-rigid material which is difficult to model due to high variations in the material characteristics and unpredictable behavior during handling and processing.

Our project focuses on automated sewing of recliner covers made of padded leather and textile parts. The parts are sewn into sub-assemblies which again are sewn into full covers. The difference in curvature of the matched edges of the parts deforms the sub-assemblies into shapes of intrinsic 3D curvature. Advanced strategies have to be utilized for identifying, handling and processing the sub-assemblies.

In a fully automated sewing cell, different operations have to be carried out for the complete process of sewing a sub-assembly:

- Detection and localization of the work pieces on a conveyor or in a stack.
- Pairing of the parts to be sewn.
- Handling and feeding of the paired work pieces into the sewing machine.
- Task control during the sewing operation.
- Handling after the sewing operation.

While all the operations in the process are important for an operative automated sewing cell, up until the current stage of the project, the focus has been on the sewing operation. The presented work uses a concept based on two independently working robots in order to be able to control parts with different shapes that may result in 3D-shaped sub-assemblies. Preliminary experiments and synchronization between two parts were discussed in [1] and the further control concepts and measurements are presented in [2].

*Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim, Norway. E-mail: johannes.schrimpf@itk.ntnu.no.

**SINTEF Raufoss Manufacturing AS, Trondheim, Norway.

Two Universal Robots UR5[1] were chosen for the demonstrator, shown in Fig. 1, due to the possibility of controlling the joints angles from an external PC at an update frequency of 125 Hz and with response time in the order of 10 ms.

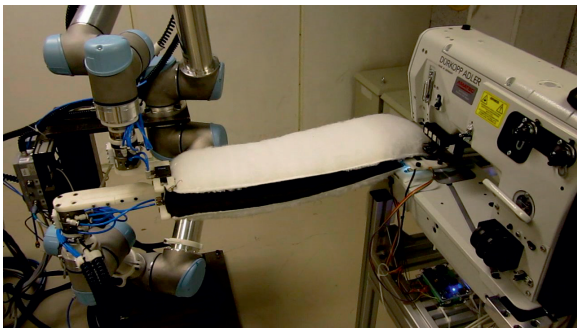


Fig. 1. The sewing cell with two robots and a sewing machine.

The resulting quality of the sewing operation is hard to quantify. This stems from the lack of a nominal sewing path and accurate tolerances since all quality control in the current production is based on human visual inspection. The most concrete numerical description of the seam path is that the seam has to be located within some 5 mm from the edge to be acceptable.

During the phase of choosing the software platform for the control PC, the main focus was on flexibility, wide availability, and systems that allow for rapid prototyping rather than industrial level reliability and stability. The main control platform is a standard PC. Ubuntu Linux was chosen as operating system due to maintainability, a wide spectrum of available software packages, and ease of use in general. The program code is mainly programmed in Python and in C++, and ROS is used for communication and organization of the different software components of the system.

Previous publications on the demonstrated sewing cell mainly focused on the quality of the sewing operation itself as well as on the choices of sensors and control strategies. The main focus of this paper is to analyze the real-time characteristics of the control system including the sensors and the robots. Another objective is to demonstrate that the chosen software and hardware solutions are suitable for utilizing the advantages of the robot's low cycle time and the low tracking delay.

[1]Previously known as UR-6-85-5-A

Related work in the field of automated sewing has been carried out by several other research groups. Gershon et al. [3], [4] presented a sewing demonstrator consisting of a single robot and a sewing machine. Sensor feedback is used to control the tension and the distance of the seam from the edge in real-time.

In Seliger and Stephan [5] an overview of the challenge of automated sewing is presented. The text focuses on the difficulties of the material handling and suggest adaptive control strategies that use measurements of the seam allowance and the feed rate during the process.

Another approach using adaptive control strategies is presented in Koustoumpardis et al. [6]. The authors control the tension in a single workpiece using a controller based on neural networks.

A sewing cell with servo-controlled feeding mechanism is presented in Winck et al. [7]. The authors control two parts independently that are separated by a thin plate. The system is based on open loop path control based on recognition of patterns on the work pieces. The authors emphasize the need for feedback control for the sewing process.

In Wetterwald et al. [8], a sewing cell that is able to attach fiber to leather parts is presented. A single robot is used together with sensors for edge detection and sewing speed.

II. System Description

A. Hardware Overview

The sewing cell consists of the following hardware components:

- Sewing machine
- Two robots
- Force sensors
- Edge sensors
- Control PC

This section describes the different hardware elements.

1) Sewing Machine: The main element of the sewing cell is a DA195 industrial sewing machine. A microcontroller board is used to access different functions on the sewing machine such as setting the drive frequency, controlling feed dog, thread cutter, and needle position. Ethernet is used for the connection to the control PC. The sewing machine has independent feeding mechanisms for the upper and the lower part with mechanically adjustable feed stitch length. The resulting feeding speed depends on the settings in the sewing machine as well as on the material thickness and characteristics.

2) Robots: Two Universal Robots UR5 6-axis robots were chosen as robot platform. They are arranged such that they can work in two planes, one plane for each workpiece. The UR5 comes with an controller cabinet based on a Debian GNU/Linux PC on which the native controller is deployed. Universal Robots offers a development library called UR_COMM_API to gain access to low-level functions on the controller. Using this C-library it is possible to replace the native motion controller by an user supplied controller, directly accessing the low-level controller with a control frequency of 125 Hz. In this frequency it is possible to read the current joint angles and command joint angles, velocities and accelerations for the next time step.

To control the UR5 from an external PC, a router application was programmed on the native controller PC, giving access to the low-level controller via Ethernet. A separate PC was installed running Ubuntu Linux. This PC is used for motion generation using an in-house developed trajectory generation framework[2] based on Orocos[3] [9]. Several motion controllers are available for different operations. The two trajectory generators used in the project are a *tool linear trajectory generator* to control the robot to the starting position and a *tool velocity trajectory generator* to move the robot in real time based on a given tool velocity (twist).

3) Force Sensors: To measure the tension in the work piece, ATI Mini45 force/torque sensors are mounted on the wrist of both of the robots. The sensors are connected to the control PC using Net F/T boxes that offer an Ethernet connection to the sensors. On the PC side, a driver stack is used to connect to the Net F/T using the RDT protocol over Ethernet[4]. The update frequency is variable, and for the sewing application set to 1 kHz

4) Edge Sensors: The control of the edge position in relation to the needle point of the sewing machine is important for the sewing task. To measure the edge position, two edge position sensors are mounted in front of the sewing foot, one for the upper and one for the lower part. The edge sensors consist of optical sensor arrays that output a one-dimensional image. To gain images of the two independent parts, a plate holding one of the sensors is placed between the two parts, while the other sensor is mounted directly on the sewing machine. The sensors are connected to a microcontroller board that sends the data via Ethernet to the control PC. The update frequency of the sensor system is set to 80 Hz

5) Control PC: A standard PC with an 8-core 2.8 GHz CPU and 8 GB RAM is used as control PC. Ubuntu Linux is used as operating system rather than a real-time operating system. This choice is in line with the focus on flexible systems that are widely available and that allow for rapid prototyping rather than on industrial level reliability and stability.

B. Control System Mechanisms

To control a sewing operation the workpieces have to be fed into the sewing machine such that the seam is located at a constant distance from the edge of both

[2] launchpad.net/python-urlibs
[3] www.orocos.org
[4] www.ros.org/wiki/netft

workpieces. Additionally, the tension between the workpieces and the robots has to be constant. Since the sewing machine has a feeding mechanism of its own, the robots have to adjust to the sewing speed in real-time. To control the edge and the tension separately, a coordinate system is defined such that corrections of the edge position result in a rotation around the needle while the tension is controlled by moving the robot towards or away from the needle, cf. Fig. 2.

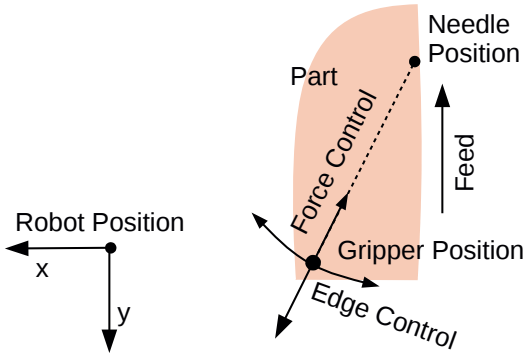


Fig. 2. The control coordinate system. Edge control results in a rotation around the needle while force control moves the robot towards or away from the needle. These movements are transformed into the robot base coordinate system.

The principle of the force controller is depicted in Fig. 3. Since the sewing force measured at the robot's tool is highly dependent on the position of the sewing foot, a running average filter is used to estimate the force over the past period of the sewing foot movement. This introduces a delay, but prevents the controller from following the periodical force change of the feed mechanism. After filtering, a PI-controller is used on the sewing force. A derivative feedback is neglected due to the delay in the control system.

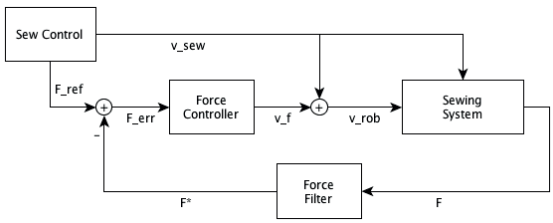


Fig. 3. Force controller for one robot.

The edge controller is shown in Fig. 4. A pure proportional controller is used. As was the case for the force controller, a derivative feedback is neglected also for the edge controller. The integral feedback is also neglected due to mechanical issues when the workpiece is sticking, which leads to an undesired accumulation of the integral contribution.

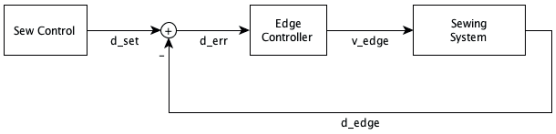


Fig. 4. Edge controller for one robot.

To gain an understanding of the whole control structure including the robots, Fig. 5 shows the whole control system and its control loops. The force and edge controllers are implemented as high-level controllers. The control output of the different controllers is added to the feed forward speed of the sewing machine and results in two velocity vectors as input for the correspondent robots.

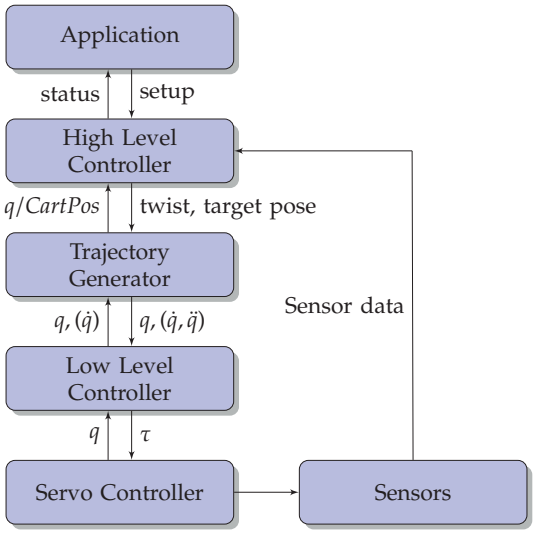


Fig. 5. The model of a real-time robot controller. CC BY-SA J. Schrimpf.

The force and edge error measurements for a typical sewing process is shown in Fig. 6. It can be seen that the controllers are able to hold both the edge position as well as the sewing force in the desired range. It is important to notice that the edge error is measured in front of the needle and changes occurring at the sensor have a smaller effect at needle position due to the rotational control strategy.

C. System Architecture

The ROS framework is used as middleware to connect the different sensors, controllers, and the robots [10]. ROS is a framework designed for robot controller systems that includes communication functionalities such as

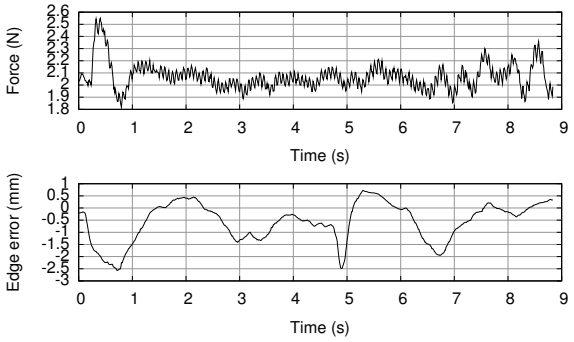


Fig. 6. Sewing force value and edge error for the lower robot for a typical sewing operation. The edge error is the error at sensor position, not at needle position.

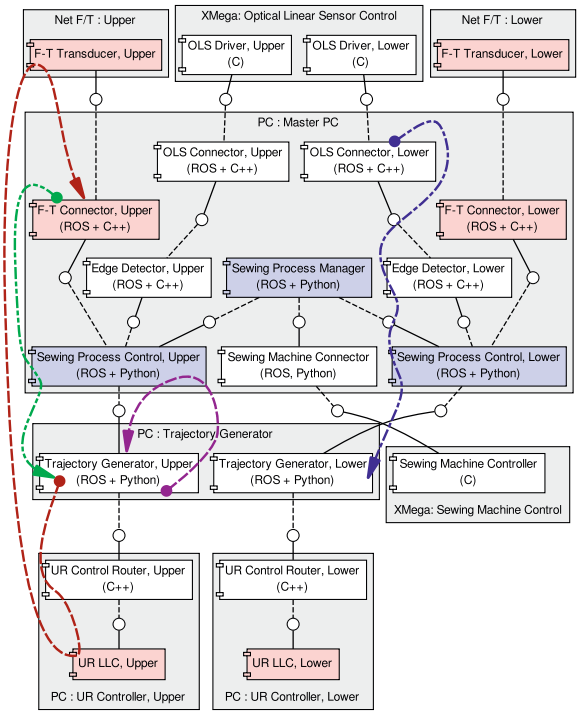


Fig. 7. Component diagram for the sewing setup with marked paths for time analysis among components.

publishing and receiving data via topics and providing services similar to remote procedure calls. Programs that integrate with the infrastructure in the ROS platform are called nodes. Additionally, ROS provides tools for visualization of data, and includes a package management system for 3rd party software.

The deployment of the different nodes is shown in Fig. 7. The functions of the nodes are as follows:

- ***F-T Connector*** – interface to the force/torque sensor
- ***OLS Connector*** – interface to the optical linear array
- ***Edge Detector*** – edge detection algorithm
- ***Trajectory Generator*** – trajectory generator for the robot
- ***Sewing Machine Connector*** – interface to the sewing machine
- ***Sewing Process Control*** – process controller including force and edge controller
- ***Sewing Process Manager*** – main application

III. Timing Experiments

A. Delays in the Robot System

In order to understand the behavior of the robot system, it is necessary to know the time it takes for the robot to process a new set of commanded joint angles. An experiment was conducted to examine this delay. The low-level interface to the robot operates with a frequency of 125 Hz. In this frequency the actual joint angles are communicated to the router program. New joint angles have to be set in a time window of 4 ms in order to be processed in the same cycle.

The arrival of a new joint update at the PC running the trajectory generator was chosen as reference point for this experiment. A ramp is commanded from the control PC and the time stamp at the reference point in the code is recorded when the movement is being processed. To measure the movement the force sensor on the tool was used running at a frequency of 1000 Hz, recording the force caused by the movement command. The time-stamped force data is then compared to the time stamp from the movement command. The measured path is the red path in Fig. 7. The result of the measurement for 1000 experiments is shown in Fig. 8.

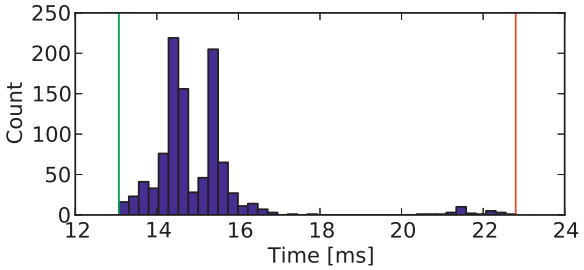


Fig. 8. Delay through the robot system from obtaining a new robot state in the trajectory generator until a movement can be measured at the force sensor.

The recorded delays vary between 13.1 ms and 22.8 ms with most measurements about 14 ms to 16 ms delay. The small peak at 22 ms indicates that there are a few cycles where the movement command is processed one cycle later. The delays include the transmission time of the data from the sensor to an external PC, which is stated in the data sheet as 0.288 ms.

B. Delay in the Sewing Controller

The time window for sending a joint update to the low-level controller after receiving the current joint state is 4 ms. This includes communication times and the calculation of the new joint set-points in the trajectory generator. Due to this hard timing constraint, the robot control loop does not include the *Sewing Process Controller*, but only the trajectory generator. This introduces a delay of one cycle in the control system since the new set-point that is calculated based on the current robot position, the force and the edge measurement is received in the trajectory generator after the new robot command is computed.

To ensure that the data is not further delayed due to long computation time in the *sewing Process Controller*, it is desirable to know the time that it takes from the computation is triggered by the new joint state update until a new twist is sent to trajectory generator. The measured path is the purple path in Fig. 7. A histogram showing this delay for 10000 measurements is shown in Fig. 9.

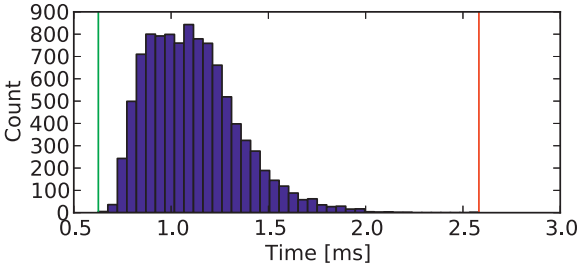


Fig. 9. Delay for the data from the trajectory generator to the sewing controller and back to the trajectory generator.

The histogram shows that all the measured delays are with a max value of 2.6 ms, which is below the deadline of 8 ms, so there is no further delay introduced due to long computation times.

C. Time Analysis for the Force Measurement

The total delay includes both the delay in the robot and sensor system as well as the computation time in the control system. Further delays are introduced by the asynchronous control loops. In the first experiment the duration was recorded between processing a motion command in the trajectory generator until the robot moves. In this experiment the remaining delay in the force control loop is measured. This is done by examining time stamps from the force measurements when the motion command is processed. The time stamps are set in the *F-T Connector* node. The measured path is the green path in Fig. 7.

Fig. 10 shows a histogram of the logged delays of 10000 measurements. The maximum delay in this measurement series, marked red in the Figure, was 11.5 ms. The average delay is just below 8 ms which is the cycle time of the robot. This delay is mainly caused by the time the set-point is stored in the trajectory generator until the next movement command is triggered by a new joint update from the low-level controller. This time would increase when using a robot with a lower update frequency.

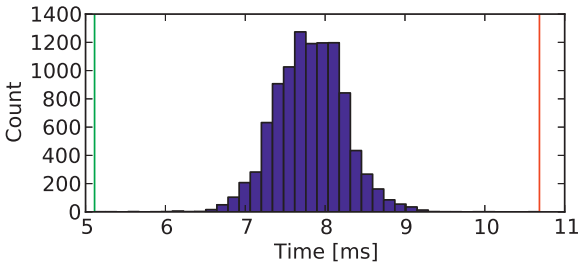


Fig. 10. Delays for the force measurements through the ROS system.

D. Time Analysis for the Edge Measurement

The last experiment was repeated for the edge loop. The time stamps are recorded in the OLS driver node, then sent together with the data through the ROS system, and evaluated when the motion is processed in the trajectory generator. The measured path is the blue path in Fig. 7.

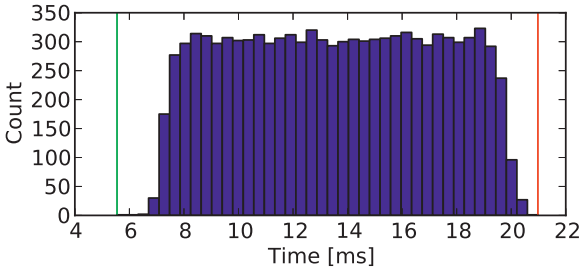


Fig. 11. Delays for the edge measurements through the ROS system in the active sewing system.

Fig. 11 shows a histogram of the logged delays of 10000 measurements. The maximum delay in this measurement series, marked red in the Figure, was 21.5 ms. The uniform distribution is due to the asynchronism between the sensor loop and the control loop. This time is also influenced by the update frequency of the robot and will increase with a lower update frequency.

IV. Performance Analysis

Table I summarizes the worst case delays for different system parts.

The most critical part of the system in respect to real-time requirements is the connection between the trajectory generator and the low-level controller of the robot.

TABLE I
WORST CASE DELAYS FOR DIFFERENT SYSTEM PARTS

System Part	Max Delay	Path
Tra. Generator (set) - Robot Movement	22.8	Red
Force Sensor - Tra. Generator (process)	11.5	Green
Edge Sensor - Tra. Generator (process)	21.5	Blue
Tra. Generator - Tra. Generator (set)	2.6	Purple

The trajectory generator has to send a new joint vector in a time window of 4 ms, otherwise the new command is ignored, or if further delayed the robot performs a security stop. Since the worst case delay from sending the data from the trajectory generator to the sewing controller and back to the trajectory generator is up to about 2.6 ms and the trajectory generator also needs time for calculations and communication, the command from the sewing controller is delayed one cycle until it has an effect on the commanded trajectory.

The worst case delay in the whole control loop, except the process, is found by adding the worst case delay in the robot to the worst case delay in the control system. This is 34.3 ms for the force loop and 44.3 ms for the edge control loop including the processing time in the sensor hardware. It can be seen that for the edge loop half of the total worst case delay is caused by communication, processing, and the asynchronous control structure, while the other half is caused by the robot system. For the force loop the delay in the control system is smaller due to the higher frequency of the sensor.

V. Conclusion and Further Work

A sewing cell was demonstrated that is able to sew two parts into a sub-assembly. The different delays in the control system were discussed. The focus was on the real-time characteristics of the system, not on the challenges of the sewing operation. Experiments were presented to determine the delays in the system. The results were discussed in respect to the requirements of the real-time interface of the robots and the different control loops in the system. The delays influenced by the update frequency of the robot are identified and these times will increase when using a robot with a lower update frequency.

There are several possibilities to tune the software and to decrease the delays in the system. One possibility is to move away from Python to C++ on the cost of the benefits Python is offering as a programming language for fast prototyping. It is questionable whether the effects would be significant since heavy calculations already now are implemented using C libraries, and a large amount of the delay is due to the asynchronous and distributed structure of the system.

Another possibility would be to implement the program monolithically to eliminate the communication delays which would be contrary to the main design decision to use a modular, distributed system.

In the current status the controller programs and the trajectory generators run on different PCs to distribute the load. This introduces a communication delay between the two parts of the program. It could be considered to distribute the program parts in another way, for example use one PC for each robot trajectory generator and the corresponding control program.

It was demonstrated, anyhow, that it is possible to build a sewing system using components that are formally not considered real-time capable while respecting the requirements of the robots low-level controller and using the benefits of real-time access to the robot. At the same time it is possible to use the benefits of these components, e.g. the high flexibility of the Python programming language in order to write code for prototyping and the communication platform offered by ROS.

Acknowledgements

The authors wish to thank the SFI Norman programme and the *Automated 3D Sewing* project founded by the Research Council of Norway. The authors also want to thank Ekornes ASA for close cooperation.

References

[1] J. Schrimpf and L. E. Wetterwald, "Experiments towards automated sewing with a multi-robot system," in *International Conference on Robotics and Automation (ICRA 2012)*, May 2012.
[2] J. Schrimpf, M. Lind, and L. E. Wetterwald, "Real-time system integration in a multi-robot sewing cell," in *International Conference on Intelligent Robots and Systems (IROS 2012) (accepted)*, 2012.
[3] D. Gershon and I. Porat, "Vision servo control of a robotic sewing system," in *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, apr 1988, pp. 1830 –1835 vol.3.
[4] D. Gershon, "Strategies for robotic handling of flexible sheet material," *Mechatronics*, vol. 3, no. 5, pp. 611 – 623, 1993.
[5] G. Seliger and J. Stephan, "Flexible garment handling with adaptive control strategies," in *Proceedings of the 29th ISR*, 1998, pp. 483 – 487.
[6] P. Koustoumpardis and N. Aspragathos, "Robotized sewing of fabrics based on a force neural network controller," in *Intelligent Robotics and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, vol. 7101, pp. 486–495.
[7] R. Winck, S. Dickerson, W. Book, and J. Huggins, "A novel approach to fabric control for automated sewing," in *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*, July 2009, pp. 53 – 58.
[8] L. E. Wetterwald, S. Dransfeld, H. Raabe, and T. Ulleberg, "Flexible robotic sewing with real time adaptive control," in *Proceedings 2nd Conference on Assembly Technologies and systems (CATS 2008)*, 2008.
[9] J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, "Implementation details of external trajectory generation for industrial robots." in *Proceedings of IWAMA 2012 - The Second International Workshop of Advanced Manufacturing and Automation*, 2012.
[10] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

Bibliography

[BBB+05] A. Blomdell, G. Bolmsjo, T. Brogardh, P. Cederberg, M. Isaksson, R. Johansson, M. Haage, K. Nilsson, M. Olsson, T. Olsson, et al., *Extending an industrial robot controller: implementation and applications of a fast open sensor interface*, Robotics & Automation Magazine, IEEE **12** (2005), no. 3, 85–94.

[Bør08] E. Børhaug, *Nonlinear control and synchronization of mechanical systems*, Ph.D. thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics, 2008.

[Bru01] H. Bruyninckx, *Open robot control software: the orocos project*, IEEE International Conference on Robotics and Automation, 2001. Proceedings, vol. 3, 2001, pp. 2523 – 2528 vol.3.

[Fos02] T. I. Fossen, *Marine Control Systems - Guidance, Navigation and Control of Ships, Rigs, and Underwater Vehicles*, Marine Cybernetics, Trondheim, 2002.

[GAMJ05] M. Graaf de, R. Aarts, J. Meijer, and J. Jonker, *Robot-sensor synchronization for real-time seamtracking in robotic laser welding*, Lasers in Manufacturing 2005 (München, Germany) (E. Beyer, F. Dausinger, A. Ostendorf, and A. Otto, eds.), AT-Fachverlag GmbH Stuttgart, 2005, pp. 419–424.

[Ger90] D. Gershon, *Parallel process decomposition of a dynamic manipulation task: robotic sewing*, IEEE Transactions on Robotics and Automation **6** (1990), no. 3, 357–367.

[Ger93] D. Gershon, *Strategies for robotic handling of flexible sheet material*, Mechatronics **3** (1993), no. 5, 611 – 623.

[GP88] D. Gershon and I. Porat, *Vision servo control of a robotic sewing system*, IEEE International Conference on Robotics and Automation, Proceedings, 1988, pp. 1830 –1835 vol.3.

[GS96] T. Gottschalk and G. Seliger, *Automated sewing of textiles with different contours*, CIRP Annals - Manufacturing Technology **45** (1996), no. 1, 23 – 26.

[HIA98] K. Hosoda, K. Igarashi, and M. Asada, *Adaptive hybrid control for visual and force servoing in an unknown environment*, Robotics & Automation Magazine, IEEE **5** (1998), no. 4, 39 – 43.

[KA11] P. Koustoumpardis and N. Aspragathos, *Robotized sewing of fabrics based on a force neural network controller*, Intelligent Robotics and Applications, Lecture Notes in Computer Science, vol. 7101, Springer Berlin / Heidelberg, 2011, pp. 486–495.

[KCL+08] C.-H. Kim, T.-Y. Choi, J.-J. Lee, J. Suh, K.-T. Park, H.-S. Kang, M.-Y. Lee, and S.-r. Kim, *Sensor guided laser welding robot system*, World Congress, vol. 17, 2008, pp. 4324–4329.

[KNMB00] M. Kudo, Y. Nasu, K. Mitobe, and B. Borovac, *Multi-arm robot control system for manipulation of flexible materials in sewing operation*, Mechatronics **10** (2000), no. 3, 371 – 402.

[Krö10] T. Kröger, *On-line trajectory generation in robotic systems: Basic concepts for instantaneous reactions to unforeseen (sensor) events*, vol. 58, Springer Verlag, 2010.

[Krö11] T. Kröger, *Opening the door to new sensor-based robot applications - the reflexxes motion libraries*, IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 1–4.

[KSK+10] D. Kubus, A. Sommerkorn, T. Kröger, J. Maaß, and F. M. Wahl, *Low-level control of robot manipulators: Distributed open real-time control architectures for stäubli rx and tx manipulators*, Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications - How to Modify and Enhance Commercial Controllers at the IEEE International Conference on Robotics and Automation (Anchorage, AK, USA), 2010, pp. 38–45.

[KW10] T. Kröger and F. Wahl, *Low-level control of robot manipulators: A brief survey on sensor-guided control and on-line trajectory generation*, 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.

[KZA06] P. Koustoumpardis, P. Zacharia, and N. Aspragathos, *Intelligent robotic handling of fabrics towards sewing*, Industrial Robotics: Programming, Simulation and Application (2006), 702.

[LH93] J. Lloyd and V. Hayward, *Real-time trajectory generation in multi-rccl*, Journal of robotic systems **10** (1993), no. 3, 369–390.

[LSU10] M. Lind, J. Schrimpf, and T. Ulleberg, *Open real-time robot controller framework*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010.

[LTS12] M. Lind, L. Tingelstad, and J. Schrimpf, *Real-time robot trajectory generation with python*, IROS2012 Workshop on Robot Motion Planning: Online, Reactive, and in Real-time, 2012.

[PFV+95] K. Paraschidis, N. Fahantidis, V. Vassiliadis, V. Petridis, Z. Doulgeri, L. Petrou, and G. Hasapis, *A robotic system for handling textile materials*, IEEE International Conference on Robotics and Automation, Proceedings, vol. 2, 1995, pp. 1769–1774 vol.2.

[rob] The International Federation of Robotics (IFR): World Robotics, *Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment,* IFR Statistical Department, Frankfurt 2012.

[SHV06] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*, John Wiley & Sons New York, 2006.

[SLM11] J. Schrimpf, M. Lind, and G. Mathisen, *Time-analysis of a real-time sensor-servoing system using line-of-sight path tracking*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 2861 –2866.

[SLM13] J. Schrimpf, M. Lind, and G. Mathisen, *Real-time analysis of a multi-robot sewing cell*, IEEE International Conference on Industrial Technology (ICIT), 2013.

[SLSM12] J. Schrimpf, M. Lind, A. Skavhaug, and G. Mathisen, *Implementation details of external trajectory generation for industrial robots*, Proceedings of IWAMA 2012 - The Second International Workshop of Advanced Manufacturing and Automation, 2012.

[SLU+10] 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*, Proceedings 3rd Conference on Assembly Technologies and Systems (CATS), 2010.

[SS98] G. Seliger and J. Stephan, *Flexible garment handling with adaptive control strategies*, Proceedings of the 29th ISR, 1998, pp. 483 – 487.

[SW12] J. Schrimpf and L. E. Wetterwald, *Experiments towards automated sewing with a multi-robot system*, International Conference on Robotics and Automation (ICRA), 2012.

[SWL12] J. Schrimpf, L. E. Wetterwald, and M. Lind, *Real-time system integration in a multi-robot sewing cell*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012.

[WDBH09] R. Winck, S. Dickerson, W. Book, and J. Huggins, *A novel approach to fabric control for automated sewing*, IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM, 2009, pp. 53 – 58.

[WDRU08] L. E. Wetterwald, S. Dransfeld, H. Raabe, and T. Ulleberg, *Flexible robotic sewing with real time adaptive control*, Proceedings 2nd Conference on Assembly Technologies and Systems (CATS), 2008.

[ZCX06] H. Zhang, H. Chen, and N. Xi, *Automated robot programming based on sensor fusion*, Industrial robot **33** (2006), no. 6, 451–459.

[ZLC06] L. Zhou, T. Lin, and S. Chen, *Autonomous acquisition of seam coordinates for arc welding robot based on visual servoing*, Journal of Intelligent and Robotic Systems **47** (2006), no. 3, 239–255 (English).