NTNU
Norwegian University of
Science and Technology

Self-Localization of Lego Trains in a Modular Framework

Henrik Heggelund Svendsen

Master of Science in Communication Technology
Submission date: June 2016
Supervisor: Peter Herrmann, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Self-Localization of Lego Trains in a Modular Framework

Student: Henrik Heggelund Svendsen

Problem description:

A specialization project was carried out at the Department of Telematics during the fall semester of 2015, focusing on the development of an autonomous, distributed control system for Lego trains operating on a model railroad located in an office at the department. The system relied on Lego Mindstorms components and hardware.

In cyber-physical systems, the self-localization and actuator control mechanisms are the portions that seal the gap between the abstract software and the real, physical world. As discovered during the mentioned project, the reliability and accuracy of these mechanisms have a strong influence on the safety properties of the system as a whole. The primary focus of the thesis will be to increase the reliability of these mechanisms, and to enhance the provisioning of environment descriptions, giving the control system a more precise and realistic foundation to base its decisions upon.

Various sensors and strategies should be tested for improving the self-localization mechanisms of the trains. If several techniques give promising results, combining different sensor inputs to improve location approximation between reliable beacon locations is of interest. Actuator control mechanisms should be made flexible so that e.g. self-adaptive mechanisms can be implemented. As the hardware used in the previously mentioned project forms a bottleneck both regarding performance and this kind of flexibility, a new hardware platform should be introduced.

To enhance modifiability aspects of the system, the mechanisms should be implemented using the modular framework OSGi.

Responsible professor: Peter Herrman, ITEM

Supervisor: Peter Herrman, ITEM

Abstract

The Department of Telematics of the Norwegian University of Science and Technology has an Intelligent Transportation Systems (ITS) lab where a model railroad layout built with Lego components is localized. The railroad model has been the center of attention in a series of projects and corresponding theses during the past year, where autonomous systems are developed and deployed on the model railroad. The prior systems have used a more or less common hardware platform based on the Lego Mindstorms EV3 smart brick. A re-emerging safety issue in these systems has been the self-localization systems, where characteristics related to a color sensor which counts passed railway sleepers acts as a bottleneck.

The focus of this thesis is the self-localization subsystem. To increase flexibility concerning peripheral connectivity, a new hardware platform based on the Raspberry Pi 2 is introduced. To improve the reliability and accuracy of the self-localization in the system, a variety of sensors are introduced, including an NFC reader; an accelerometer; a magnetometer and a color light sensor. Also, a motor controller under the direct control of the Raspberry Pi is used.

As the OSGi framework is central to the chosen modular approach, components from the previous systems are reused but is not adopted in its totality. Software modules for each of the peripherals are developed and tested, and their suitability as contributors to self-localization purposes are discussed. Summarized, each of the sensors' contributions to self-localization is listed below:

- The NFC reader, combined with Mifare tags encoded with location information placed underneath the railway surface functioning like beacons, stands out as the equipment supplying the most precise and reliable information.
- The color light sensor acts as a minor improvement compared to the one in the preceding projects' platform, thus it can aid in making approximated updates to the positional data in between the more precise beacons.
- The magnetometer provides data, which after processing reports the current heading of the train. The calculated heading seems accurate in the initial experiments, but as additional tests reveal it is subject to the fluctuating magnetic interference from the surroundings. The issue of interference may be diminished by physical isolation of the

model railroad. Suggestions for improvements are presented, but the heading data is at this point only applicable for coarse position approximations.

– The accelerometer output acceleration measurements, which are analyzed in real-time to calculate changes in velocity. The analysis turns out to be highly vulnerable for deviations in the acceleration readings, resulting in unpredictable outputs. Although efforts are made to migitate the deviations, satisfactory results are not achieved.

During the phase of testing the mentioned modules, the modular aspects of the architecture are implicitly tested. Sensor modules are easily interchangeable with simulated modules, and published events are used for logging experimental data.

Sammendrag

Institutt for telematikk (ITEM) ved Norges teknisk-naturvitenskapelige universitet (NTNU) har en lab for intelligente transportsystemer (ITS), hvor en modell-jernbane bygget av Lego-komponenter er huset. Denne modell-jernbanen har vært sentral i flere prosjekter og tilhørende avhandlinger de siste årene, hvor autonome systemer har blitt utviklet og anvendt på modellen. De foregående systemene har brukt mer eller mindre en felles fysisk plattform basert på smarte Lego Mindstorms EV3-enheter. Et gjentagende problem relatert til delsystemene for selv-lokalisering har vist seg å dukke opp, der karakteristikker relatert til en lys-fargesensor som teller passerte jernbanesviller oppfører seg som en flaskehals.

Fokuset i denne avhandlingen er delsystemet for selv-lokalisering. For å øke fleksibiliteten i forhold til tilkoblingsbarhet til perifere enheter som sensorer er en ny fysisk plattform introdusert, basert på mini-datamaskinen Raspberry Pi 2. For å forbedre påliteligheten og nøyaktigheten til selv-lokaliseringssystemet er en rekke nye sensorer innført, som omfatter en NFC-leser, et akselerometer, et magnetometer og en lys-fargesensor. I tillegg brukes en motorkontroller som blir direkte styrt av Raspberry Pi-en.

Siden OSGi-rammeverket står sentralt i den valgte modulære tilnærmingen blir enkelte del-komponenter gjenbrukt fra tidligere systemer, men komponentene kan ikke bli gjenbrukt i sin helhet. Programvaremoduler for hvert av de perifere enhetene blir utviklet og testet, og brukbarheten deres som bidragsytere til selv-lokaliseringssytemet blir diskutert. Hver enkelt av sensorenes bidrag til selv-lokalisering i systemet er kort oppsummert her:

- NFC-leseren, kombinert med Mifare-transpondere som er kodet med stedsinformasjon og plassert under selve jernbanemodellen som landemerker, skiller seg ut som utstyret som leverer mest presis og pålitelig posisjonsinformasjon.
- Lys-fargesensoren fungerer som en liten forbedring sammenlignet med den som er brukt i tidligere prosjekter, og kan dermed bistå i å levere anslagsvise oppdateringer av posisjonsdata i mellom de mer presise landemerkene.
- Magnetometeret leverer data som etter prosessering rapporterer om togets nåværende kurs. Den utregnede kursen virker korrekt i de innledende eksperimentene, men videre testing avslører at sensoren

er utsatt for varierende interferens fra magnetiske felter som stråler ut i omgivelsene. Problemet med interferens kan minskes ved å adskille modellbanen fysisk fra kildene, men i skrivende øyeblikk kan kompasskurs-data bare anbefales til å gjøre grove estimater for posisjonsoppdateringer.

– Akselerometeret avgir akselerasjonsmålinger, som blir analysert i sanntid for å regne ut endringer i hastighet. Analysen viser seg å være veldig sårbar for avvik i akselerasjonsmålingene, noe som resulterer i uforutsigbare utregninger. Selv om forsøk er gjort på å forminske avvikene fra akselerometeret blir ikke tilfredsstillende resultater oppnådd.

Under fasen med testing av de nevnte modulene blir også de modulære aspektene ved systemarkitekturen testet implisitt. Sensormoduler er ved enkelhet utbyttbare med simulerte moduler, og publiserte hendelser i systemet blir brukt for å loggføre eksperimentelle data.

Preface

This master's thesis was carried out by Henrik Heggelund Svendsen during the spring semester of 2016 as the final part of the 5-year Master of Science programme in Communication Technology at the Department of Telematics of the Norwegian University of Science and Technology. Peter Herrmann acted as both the responsible professor and the supervisor during the conduction of this project.

I would like to thank Peter for providing valuable input through discussions, Pål Sturla Sæther for supplying me with required material and tools, and Alvaro F. Fernandez for sharing prototype source code for the Xtrinsic sense board. Co-student Alexander Svae has been a great sparring partner, and several ideas has been a product of our chats. At last I would like to thank my girlfriend Caroline and our daugher Ella for the encouragement, the laughter, for giving me an airing when needed, and for bearing with me all the way.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AMQP Advanced Message Queuing Protocol.

API Application Program Interface.

DC direct current.

EEPROM Electrically Erasable Programmable Read-Only Memory.

ERTMS European Rail Traffic Management System.

ETCS European Train Control System.

GPIO General-Purpose Input/Output.

GSM-R Global System for Mobile Communications – Railway.

GUI Graphical User Interface.

I^2C Inter-Integrated Circuit.

IC integrated circuit.

ICT Information and Communications Technology.

IDE Integrated Development Environment.

ITS Intelligent Transportation Systems.

JDK Java Development Kit.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

LiPo Lithium Polymer.

LSB least significant bit.

NFC Near Field Communication.

PRT Private Rapid Transport.

PWM Pulse Width Modulation.

RFID Radio Frequency Identification.

RMS root mean square.

SASL Simple Authentication and Security Layer.

SCL Serial Clock Line.

SDA Serial Data line.

SOA Service-Oriented Architecture.

SPI Serial Peripheral Interface.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UART Universal Asynchronous Receiver/Transmitter.

UHF Ultra High Frequency.

UML Unified Modeling Language.

XML EXtensible Markup Language.

Chapter 1
Introduction

1.1 Intelligent Transportation Systems

Intelligent Transportation Systems (ITS) is, in general, a term used for all systems where Information and Communications Technology (ICT) is applied to, or alongside with, vehicular transport of personnel or goods, with an aim to increase safety, convenience and efficiency of the (existing) transportation systems, as well as to decrease socio-economic costs and environmental impact. ITS spans from automatic toll road systems to self-driving cars and collision avoidance systems. Although the idea of ITS was conceived in the 1980s, the boost in technological development and hence a dramatic decrease in hardware cost the latest years makes ITS now more relevant than ever. The debate concerning human-made climate changes combined with population growth also contributes to this fact, as factors like these express the need for improvements of the systems in use today.

1.1.1 Challenges

The appliance of ICT in physical systems such as moving vehicles results in what we define as a cyber-physical system. In a system of systems, e.g. the co-operation of 'smart' vehicles, strict spatiotemporal requirements apply as the sub-systems must perform self-localization and communicate their location to sub-systems in their proximity. Self-localization introduces challenges regarding the interpretation of the real world and real-time analysis of data, combined with communication delay or errors which add up to a complex whole.

1.2 Problem description and scope

A specialization project was carried out at the Department of Telematics during the fall semester of 2015, focusing on the development of an autonomous, distributed control system for Lego trains operating on a model railroad located in an office at the department. The system relied on Lego Mindstorms components and hardware.

In cyber-physical systems, the self-localization and actuator control mechanisms are the portions that seal the gap between the abstract software and the real, physical world. As discovered during the mentioned project, the reliability and accuracy of these mechanisms have a strong influence on the safety properties of the system as a whole. The primary focus of the thesis will be to increase the reliability of these mechanisms, and to enhance the provisioning of environmental descriptions, giving the control system a more precise and realistic foundation for decision-making.

Various sensors and strategies are to be tested for improving the self-localization mechanisms of the trains. If several techniques give promising results, combining different sensor inputs to improve location approximation between reliable balise locations is of interest. Actuator control mechanisms should be made flexible to prepare the implementation of e.g. self-adaptive mechanisms. As the hardware used in the previously mentioned project forms a bottleneck both regarding performance and this kind of flexibility, a new hardware platform should be introduced.

To enhance the modifiability aspects of the system, the mechanisms should be implemented using the modular framework OSGi.

1.3 Motivation

The motivation for this thesis is to establish a reliable self-localization subsystem for the model railroad system in the ITS lab. The subsystem should be of such a nature that integrating it in future systems should be a straight forward task. This requirement implies that a robust, modular architecture is needed. The overall aim is that the model railroad can act as a realistic testbed for developing, analyzing and deploying highly distributed, autonomous prototype technology that can, later on, be used in larger scale systems.

1.4 Methodology

As a modular framework will be utilized, modularity is an important element in the system architecture. A model-based approach is chosen for reactive modules subject to simultaneous inputs from multiple sources. Modules that are of a simpler nature can still be developed as pure Java/OSGi modules. Functional analysis of modules and connected hardware is to be carried out by collecting and comparing data samples comprising timing and measured physical metrics.

1.5 Structure of the report

In **Chapter 2**, background material that is required when studying the following chapters is presented. The background material is relatively coarse, and the reader is referred to specific material in case a more detailed presentation is of interest.

Chapter 3 presents the formal requirements set for the system, both of functional and non-functional nature.

Chapter 4 describes in closer detail how hardware components are combined to form the system's physical model.

Chapter 5 presents the software architecture of the system, describing both the use of software modules and the interactions between them.

Chapter 6 describes how data from the different sensors can be processed and utilized to improve the self-localization of trains. At the end of the chapter, a description of the integration with the distributed interlocking protocol introduced in [Sve15c, 7.1] is given.

Chapter 7 deals with the results and issues experienced while testing the individual components and the implemented software modules. An analysis of collected data is presented.

Chapter 8 contains discussion regarding the results from the previous chapter, and a conclusion sums up the thesis. Recommendations regarding further work are made at the end of the chapter.

Chapter 2
Background

2.1 European Rail Traffic Management System

In a historical perspective, European railway consisted of twenty-three different signaling systems distributed in fifteen countries. The diversity of solutions demanded international train sets to carry multiple technical systems, occupying valuable line capacity by requiring additional switching time. To counteract the pluralistic evolution of systems, the European Commission introduced the European Rail Traffic Management System (ERTMS), consisting of the two modules: the signaling system European Train Control System (ETCS) and the communication system Global System for Mobile Communications – Railway (GSM-R).

The ETCS specification defines four levels, from level 0 which concerns backward compatibility with older systems, to level 3 where equipment such as track circuits are deprecated. Typical for the three highest levels are the introduction of the "Eurobalise"[1]. The Eurobalise is a transponder device mounted between the rails on the ground, which transmits a datagram to passing trains. The passing train picks up the signaling with an underslung antenna, as seen in the picture in Figure 2.1. The datagram contains control data like speed restrictions, gradient, and positioning information [Blo06]. In levels 2/3, the registered data is made available for the centralized control center in real time by continuous transmission on the GSM-R radio link.

Figure 2.1: ETCS Eurobalise ground equipment and train antenna module, image from http://www.siemens.com/press/en/presspicture/

[1] A balise is an electronic beacon or transponder placed between the rails of a railway as part of an system. The word "balise" is used to distinguish these beacons from other kinds of beacons

The ERTMS system has gradually been installed in European countries during the last decade, and foreign stakeholders outside Europe has in the recent years begun to embrace the technology. By 2014, the system comprised more than 80.000 kilometers of track, and nearly 10.000 equipped vehicles worldwide [ERT14].

2.2 Technology

2.2.1 Publish/subscribe protocols

In distributed systems, the use of traditional *point-to-point* and *synchronous* communication often result in highly coupled, rigid systems[EFGK03]. The notion of a publish/subscribe interaction scheme is a collective term for protocols alleviating this burden by introducing a central component responsible for routing messages *from* the producers of data, the "publishers", to the consumers of data, the "subscribers". The central component is a software bus, an event manager. The publishers have no relation to whom the subscribers are, and vice versa – they are only acquainted with the event manager. Producers publish messages or events to the event manager, and consumers inform the event manager which types of messages or events of interest for the individual consumer. When publishing data, the event manager takes care of routing data to the subscribing consumers. This flow of data is depicted in the message sequence diagram in Figure 2.2.

The benefits of utilizing this kind of interaction scheme are decoupling concerning both time, space, and synchronization.

2.2.1.1 AMQP

Advanced Message Queuing Protocol (AMQP) is an open protocol specification, providing a variety of configurations. Among them, a publish/subscribe configuration is possible. The protocol is aimed to act as a middleware between networked applications across all platforms[OAS15a]. AMQP is a binary application-level protocol, and assumes to be running on a reliable transport protocol such as Transmission Control Protocol (TCP). Additional functionality provided is cryptographic security through Simple Authentication and Security Layer (SASL) and Transport Layer Security (TLS), message queuing, and message-delivery guarantees such as *at-most-once*, *at-least-once* or *exactly-once*, depending on configurable settings. A popular multi-platform implementation of the protocol is RabbitMQ [OAS15b], which provides both client APIs in various programming languages, as well as server daemon applications for multiple platforms.

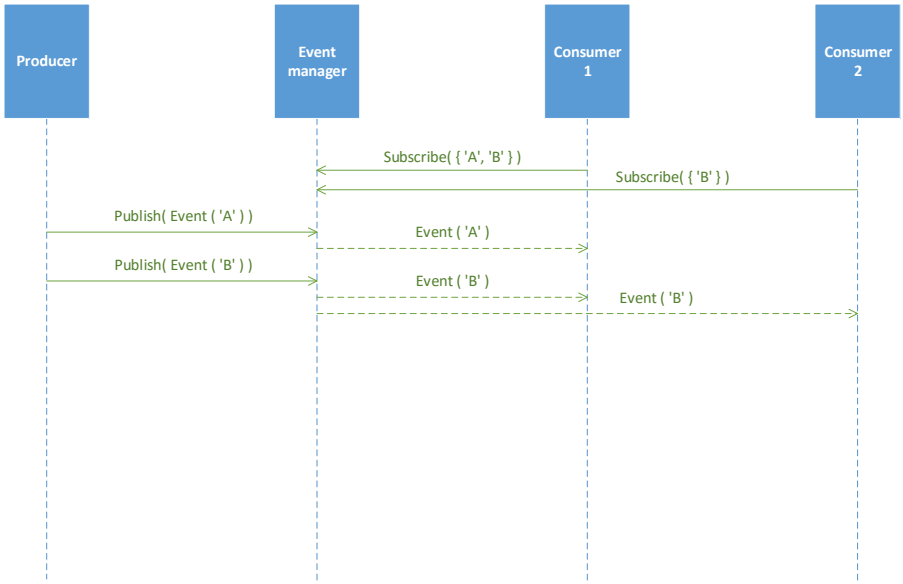


Figure 2.2: Actors and flow in a publish/subscribe interaction scheme

2.2.2 I²C protocol

I²C is a digital peripheral serial bus, supported by most digital peripheral devices like sensors. The bus uses two wires for connectivity; a Serial Data line (SDA) for data transfer and a Serial Clock Line (SCL) for clock synchronization. Data can be transferred on the bus at rates from 100kbit/s to 3.4Mbit/s, depending on mode. Equipment (slave) connected to a host (master) through an I²C bus has a 7-bit hardware address[2] represented by a two-digits wide hexadecimal number spanning from `0x08` to `0x77`. The excessive addresses are reserved according to the I²C specification. The I²C bus is limited to 111 devices, which is sufficient in most applications. Most equipment comes with the I²C address hard coded into the integrated circuit (IC), which in practice means that only one unit can be connected to each bus. The interested reader is referred to [Sem00] for an in-depth description of the protocol, though it is not required for further reading and understanding of this thesis.

[2]Both 7 and 10-bit address mode exists for I²C, though the latter one was added as an extension. 7-bit addressing is the standard mode.

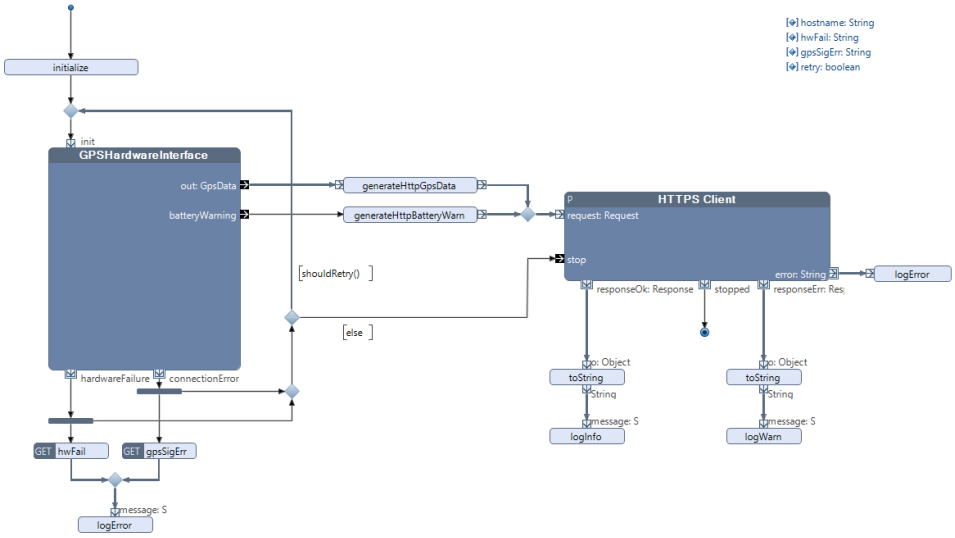


Figure 2.3: An example application in Reactive Blocks posting received GPS data updates to a HTTPS server

2.2.3 Reactive Blocks

Reactive Blocks is a model-based software development plug-in for the Eclipse Java Integrated Development Environment (IDE). It provides a Graphical User Interface (GUI) editor with drag-and-drop functionality where the developer can design the flow of the program. The program flow is generally composed in building blocks resembling an Unified Modeling Language (UML) activity diagram. Nesting of building blocks inside other blocks like depicted in Figure 2.3, creates a hierarchy of functionality – thus supporting the single responsibility principle[Mar03]. Blocks are connected to Java code through *operations* or activities as they are called in the UML context. Business logic can accordingly be written in pure Java code after the flow of the block/program is defined.

Reactive Blocks is, as the name suggests, a tool aimed at the development of reactive systems. The reason it's suitable for reacting to inputs/impulses is that the developer can focus more on the design of the application instead of details concerning e.g. concurrency handling. The framework automatically analyzes the system during the design phase, continuously supplying the developer with information and warnings if e.g. deadlocks, race conditions or synchronization errors. The automatic analysis uses formal verification techniques, relieving the developer from an entire phase of the software development cycle. After designing the system, connecting the underlying business logic and solving potential problems revealed by the analysis, Reactive Blocks builds the executable program by examining the design and generating executable code from it. Kraemer, being the originator of the Reactive Blocks project, presents an in-depth study of these techniques in [Kra08].

Reactive Blocks support various development platforms, including OSGi, which is described further in Section 2.2.4. With little extra effort, it can be used to develop OSGi bundles ready to deploy into new or existing systems. Like regular Java projects, the generated code outputted by Reactive blocks is accompanied by a manifest file which defines the OSGi policies for the bundle. The use of Reactive Blocks and the Model-based engineering method in relation to space-aware systems have been studied in, among other, [HBHS15] and [HBHS16].

2.2.4 OSGi

OSGi[3] is a technology acting as a modularity layer for Java based systems. The word *modularity* represents the logical decomposition of a complete system into smaller sub-systems – or *modules*, reducing the coupling and formalizing dependencies between the different modules [HPMS11].

2.2.4.1 Layers of the OSGi framework

The OSGi framework is a significant aspect of the OSGi technology. The framework is the execution environment of the application and provides a standardized API for the developer. As it is a multifaceted framework, it's often described as a layered architecture as shown in Figure 2.5. A short description of these layers is presented below.

The module layer is the lowest level layer in OSGi, concerned with modules, or bundles, and how code is shared between them. A bundle, or plug-in as it's sometimes referred to, is a single module with its own Application Program Interface (API). The

[3]OSGi was formerly an acronym for the Open Service Gateway Initiative, but the acronym was dropped after the third specification release. *OSGi* is now a trademark for the technology.

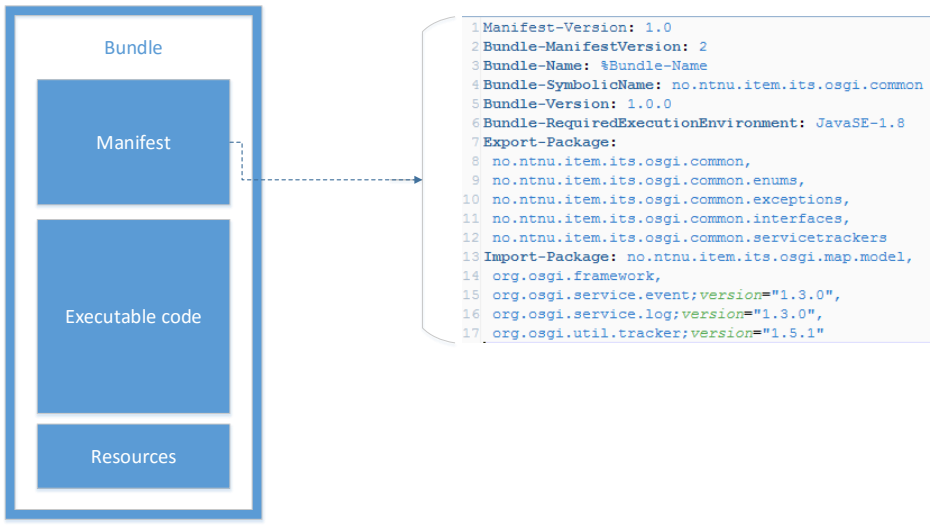


Figure 2.4: Illustration of an OSGi bundle

bundle is deployed as a *.jar*-file, which like regular Java JAR-files contains executable binaries and resources, and additionally, a metadata text file called the bundle's *manifest*. The manifest provides information regarding the respective bundle's fully qualified name, the bundle's version, which packages are exported, i.e. visible for other bundles (API), in addition to which dependencies the particular bundle has. A simple illustration of a bundle and it contents, along with an example of its manifest is shown in Figure 2.4. To clarify the bundle definition, it is no more than a regular Java project with it's compiled class files, resources (e.g. images), in addition to the mentioned manifest, compressed in a regular JAR file. In the OSGi context, the bundle is a *module*.

The lifecycle layer deals with installing new bundles to the framework's bundle cache, resolving the bundles' dependencies (i.e. verifying that the necessary bundles are available and resolved), starting and stopping bundles, i.e. like the name of the layer suggests: managing the lifecycle of the bundles.

The service layer is the part of the OSGi framework administering services. Before describing the service layer any further, a short definition of the term *service* is beneficial. A service is, in a software system context, nothing more than "work done

for another"[HPMS11, 4.1], i.e. in it simplest form nothing more than a method call from a caller to a callee; the callee is doing work for the caller. The Service-Oriented Architecture (SOA) is though something more than a simple method call. In OSGi, a service entails a contract – a regular Java interface which defines the API for the service. The service is registered to the framework's service registry, and consumers of the service need only to keep a reference to the framework component and have knowledge of the service contract in order to use the service. This pattern resembles the one mentioned in Section 2.2.1: a service is *published* (registered) to a *software bus* (service registry), making it amenable to incoming requests from service consumers. The consumers have no relation to who/where the service is, or how the service is implemented, other than the service contract itself. The service layer deals with service (de)registration, updates, and lookups.

As the security functions of OSGi are out of the scope of this paper, a description of the security layer which lies side-by-side to the other layers mentioned above is left out of this section. For more in-depth reading material, the reader is referred to [HPMS11].

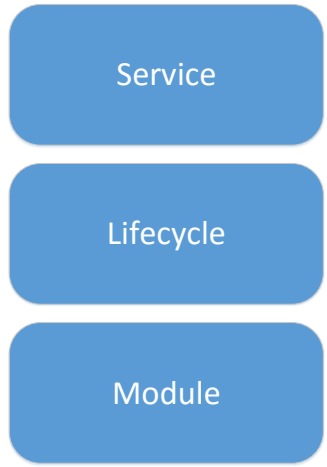


Figure 2.5: The layers of OSGi

2.2.5 Radio technologies

2.2.5.1 RFID

Radio Frequency Identification (RFID) is a communication technology operating on short range via radio signaling. The term is used for a mixed collection of radio frequency and techniques used to enable this communication. The typical employment of RFID has simple transponders (often called tags) on one end of the radio link, and more advanced devices (often called readers) on the other end[Lan05]. The tags are produced in the form of a microchip attached to an antenna, some so small that the term *RFID powder* has been introduced[Hor08].

Two sub-categories of tags exist; active and passive. The active tags have a built-in power source, typically a battery, powering the tag's operation. Power saving techniques exist to alleviate excessive drainage of the battery, such as requiring the reader to send wake-up signals, so the transponder only broadcasts its signal when it is within the range of the receiver, or by sending periodical broadcasts e.g. once an hour. Active tags usually operate at high frequencies and over relatively wide ranges up to 100 meters, and cost from $10 to $50,

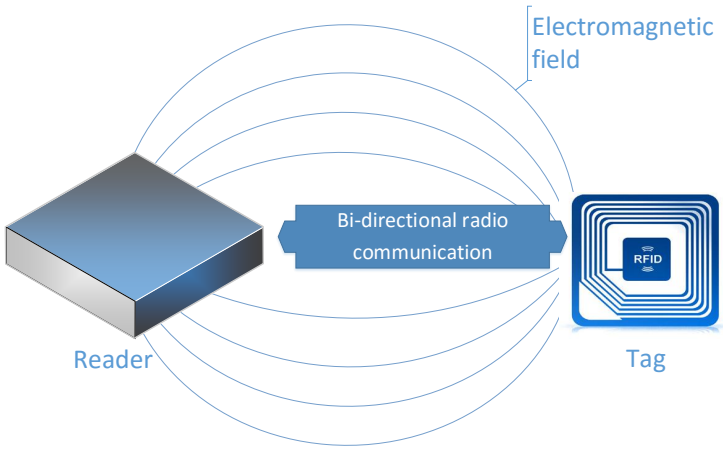


Figure 2.7: Passive RFID system illustration

depending on specifications. An example of a typical application of active tags is toll roads, e.g. the autoPASS system of Norway.

Passive tags, i.e. tags without an internal or connected power supply, are a less expensive alternative, with a cost of 20 cents to 40 cents in general. Frequencies and ranges are varying and tightly coupled, with 1/3 meters for low-frequency applications to 3 meters and more for Ultra High Frequency (UHF). The advantage of using low frequencies when considering low-power applications is that the radio waves penetrate materials more efficiently than higher frequencies, that tend to behave more like light waves. An example of a typical application to passive RFID tags is passports, where personal information like fingerprints, name, and address is encoded into the tag, to complicate modification and fraud. The communication between a reader and a passive tag is illustrated in Figure 2.7.

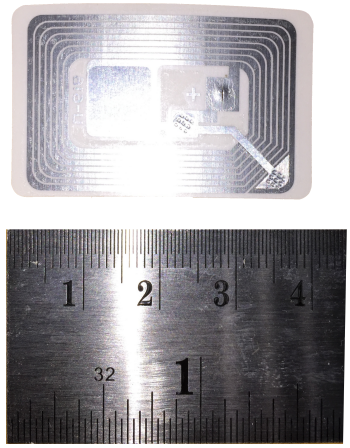


Figure 2.6: RFID tag embedded in a sticker

Passive systems utilizing low and high-frequency radio carriers use inductive coupling, i.e. a magnetic field powered by the reader is formed between the reader and the tag, inducing

a current in the tag. The range of the electromagnetic field contributes to the short ranges of these systems. Passive UHF systems use propagation coupling, or backscatter, where the tag reflects an altered signal back to the reader[Vio05]. Passive tags are generally quite compact, and are often mounted between a sheet of paper and an adhesive surface, resulting in an RFID sticker, as shown in Figure 2.6. The physical measurements of the depicted tag are 40mm * 25mm.

In short terms, the purpose of the communication is to transfer data from the tag to the reader. This data is typically associated with the identity of the object carrying the tag, hence radio frequency *identification*.

2.2.5.2 NFC

Near Field Communication (NFC) is a subset of the RFID technology, and is comprised by short range communication and passive tag usage only. Because of short range communication, secure data exchange is an inherited feature. NFC is deployed in many modern mobile phones and other electronic home appliances like network printers and Bluetooth speakers, and is used for identification purposes when establishing links between the participants in e.g. the Bluetooth protocol. The radio link in NFC applications operates at 13.56MHz[COO11, 3.4.1].

NFC is also enables contactless transactions with proximity smart cards, as defined in the ISO/IEC 14443A/B standard[ISO16]. Although several proximity smart card technologies exist, MIFARE is by far the most widely used system, developed and owned by NXP Semiconductors. By 2011, MIFARE was used in more than 80% of all contactless smart cards in the world[COO11, 3.4.1.2], spanning from public transport ticketing to e-payment. The MIFARE smart card IC supports a typical read/write range up to 10 centimeters, and generally includes an Electrically Erasable Programmable Read-Only Memory (EEPROM) with capacity from 1 to 4 kilobytes of memory which can be programmed wirelessly utilizing NFC.

2.2.6 Raspberry Pi

A Raspberry Pi 2 (from now on abbreviated RPi) is a pocket-sized, full fletched computer with relatively high performance. It offers standard interfaces like Ethernet, USB, and HDMI in addition to the pin header which provides access to power outputs, General-Purpose Input/Output (GPIO) connectors and peripheral buses like I^2C. The GPIO connectors and peripheral buses can be utilized to connect both analog and digital electronic equipment, including a variety of sensors and communication equipment. The pin header is depicted alongside with the other connectors in Figure 2.8.

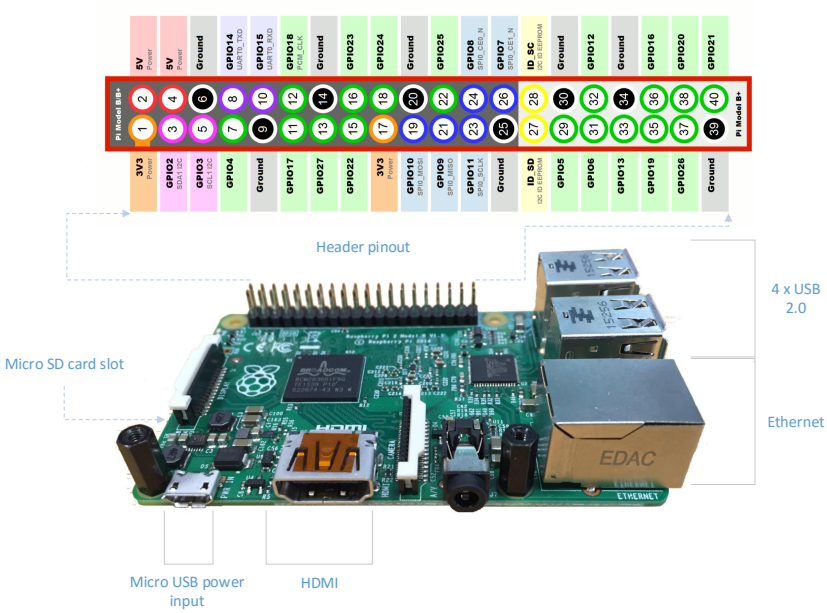


Figure 2.8: An overview of the Raspberry Pi 2 with its various interfaces

2.3 Related work

This thesis is the fourth in a line of reports written at the Department of Telematics, dealing with an autonomous model railroad system composed of Lego equipment. The model railroad itself has differed to some extent in the previous theses, regarding layout and supported behavior. Although the previous projects have used a similar physical architecture, the software architectures – and thus the implementations – has been developed from scratch each time, re-using only small parts from the prior systems. Although the system described in this thesis is independent of the earliest theses, the experience gained from those are valuable for this project. Also, the third of the preceding reports forms an essential foundation for the system in focus in this thesis. On these bases, the reports are summarized in short terms in this section. For more details concerning these systems, the reader is referred to the particular theses.

2.3.1 A PRT system

The first system is presented in Overskeid (2015)[Ove15]. Overskeid describes a model of a Private Rapid Transport (PRT) system and introduces the physical hardware architecture which is re-used in the following theses. The system is centralized

but distributed to the grade of having separate authorities controlling the vehicles traveling within their respective area of control. The physical Lego railroad is used as a model for a PRT guideway, which is uni-directional by nature. Self-localization is in this system based on counting passed railway sleepers, using a Lego Mindstorms color sensor directed towards the rail's surface.

2.3.2 An autonomous train control software

Hordvik and Øseth (2015)[HØ15] presents in their thesis an autonomous control software for trains. While sharing the hardware architecture and intelligence distribution with the mentioned system composed by Overskeid, Hordvik, and Øseth's implementation introduces a detailed color scheme used for self-localization of the trains. Based on which location the train had on the latest update, the current position is verified if the expected colored railway sleepers are detected by the color sensor. The localization data read by the trains are sent to centralized zone-controllers responsible for planning the movement of the trains, preventing any conflicts between intersecting trains. To reduce the complexity in the control system, this system is too restricted to uni-directional traffic.

2.3.3 A distributed, autonomous control system

The author of this thesis designed and developed a third system, which is described in [Sve15c]. Using the same hardware architecture as the two preceding projects, the project thesis introduces two significant improvements: a general interface for interpreting railroad layouts and allowing bi-directional traffic on the railway. Also, a striking difference in design regarding intelligence distribution is presented, as the control system is completely distributed with no centralized actors, besides the server acting as a messaging hub for the trains. The trains are i.e. entirely autonomous, each co-operating with the other trains as peers to prevent collisions.

In all three systems, WiFi is used for intercommunication between the trains, utilizing different publish/subscribe protocols in the application layer, on top of TCP in the transport layer. Publish/subscribe protocols are explained in greater detail in Section 2.2.1.

While all the mentioned projects illustrate different, and quite interesting aspects of autonomous transportation systems, they all point out the same components as the weakness regarding self-localization: the Lego Mindstorms color sensor combined with the performance of the on-board computer equipment of each train.

As some central parts of both design and implementation of the system in [Sve15c] is re-used to a great extent in this particular thesis, a rough overview of

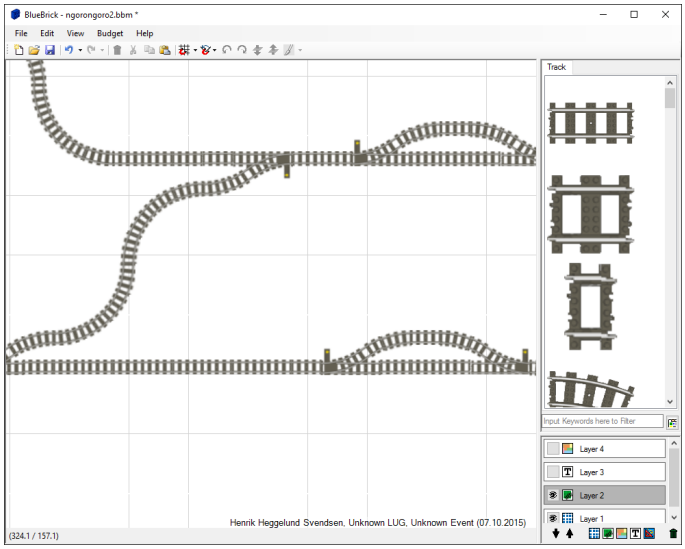


Figure 2.9: Screen capture from Bluebrick

the respective parts will be presented in the following paragraphs to provide the necessary background for this thesis.

2.3.3.1 Interpretation of railway design

As mentioned, Svendsen describes a general method for designing and interpreting railway layouts. An open-source developed tool called Bluebrick[MN15] is used for creating and altering railway layouts in a user-friendly fashion, as shown in Figure 2.9. Available parts are shown in the box to the right; the layout under construction is shown in the main canvas to the left. The tool outputs the designed layout as an EXtensible Markup Language (XML) structured file, which in turn is used as a basis for generating a data structure interpretable by the trains' control software. The derived map structure is used as both a state context for self-localization in each train, and as a contextual basis for the interactive train to train communication. The generated data structure and the procedures used for the translation from XML and Java object instances are described in greater detail in [Sve15c].

2.3.3.2 Collision avoidance

Svendsen proposes in [Sve15c] a protocol for distributed interlocking. It is based on the two-phase commit protocol used for atomic, distributed transactions, and its key functionality lies in guaranteed consensus regarding a shared interlocking table where reservations can be made without vulnerability to timing issues like e.g. network delay. The communication between trains utilizes AMQP via a designated

server acting as a message broker. Messages carry an object structure serialized as a JavaScript Object Notation (JSON) string. The AMQP messages are transported over the transport layer using TCP, and the link layer is composed of both IEEE 802.11 (WiFi) and Ethernet cabling.

The data structure mentioned in Section 2.3.3.1 is used as a basis for the shared model. Before a train is allowed to depart a station, the train must initiate a reservation procedure for the track it plans to cover during its next transfer, including the station track it plans to stop at. Not until the distributed interlocking protocol submit a positive confirmation, the train may start moving out of the station. Meeting collisions will never occur, neither will incidents of trains catching up and colliding with preceding trains, as long as the self-localization in trains reflects reality correctly at all times. Using this technique Svendsen's system maintain the necessary safety properties, though increased efficiency could be a subject for improvement.

2.3.4 A self-adaptive sensor system

In parallel with the work presented in this thesis, co-student Alexander Svae is designing and developing a self-adaptive system for the train model. Although his work is presented in an individual thesis and is considered to be independent, sharing the same physical framework and components has led to benefits in both camps. Svae's system focuses on how input from the surroundings and contextual data can be used to alter the behavior and functionality of the components in a dynamic manner. More specific, Svae's system is developed towards the same OSGi framework as this system, i.e. combining the two systems should be a relatively straightforward task.

Chapter 3
System Requirements

The system requirements sums up the functional and non-functional requirements the system is subject to.

3.1 Functional requirements

Functional requirements *what* the system should do, i.e. the explicit functionality. The following functional requirements are identified:

- Self-localization – Provide real-time data processing which can be utilized for improving accuracy and reliability of self-localization mechanisms
- Mobility – The system hardware must be of a compact nature, supporting the mobility needed in a moving, vehicular system.

3.2 Non-functional requirements

Non-functional requirements are of a more abstract nature than functional requirements, and describes *how* the system should e.g. perform. The following non-functional requirements are identified:

- Modularity – The system should be composed of modules which interact with each other in a dynamic manner. Modules should be interchangeable, and should provide valuable functions although the module is used in a different context.
- Modifiability – It should not be complicated to extend the functionality of the system.

3.3 Inherited requirements

As the self-localization subsystem must interact with e.g. an internal model of the railway defined in preceding projects, the requirements identified in [Sve15c, 3] are inherited. The inherited requirements are however not in the scope of this thesis but provides valuable background information insight. The inherited functional requirements are safety; reliability; bi-directionality, while the inherited non-functional requirements are autonomy; reconfigurability; and scalability.

Chapter 4
Physical Composition

As stated in Section 1.2 and Section 2.3, preceding projects related to the model railroad has used a common physical composition subject to little variation. As the physical components onboard the trains has proved to act as a bottleneck regarding self-localization[Sve15c, 8.1.2], a new hardware platform is introduced here. Note that the ground-based equipment from previous projects is re-used, as none of the contributors pointed out problems in these sub-systems.

This chapter is an overview of the chosen physical components of the system, and a presentation of the physical system composition is given. In the first section, a technical description of each component is given. The second section addresses how the different components are interconnected and assembled. At the end of the chapter, a summarized presentation of the ground-based components re-used from previous work[Ove15][HØ15]. How the raw data from physical components will be used in the scope of Section 6, and will not be treated in this chapter.

4.1 Components

4.1.1 Raspberry Pi 2

The pocket-sized computer Raspberry Pi 2 (RPi) is briefly introduced in Section 2.2.6. The RPi is chosen as the core component in the new hardware platform on the basis of its versatility. Below are some key points that stand out as important abilities in this system:

- **Connectivity** – As mentioned in Section 2.2.6, multiple standardized interfaces are available on the RPi. USB 2.0 connectors in combination with a regular WiFi-dongle, as depicted in Figure 4.1, allows for easy access to wireless networks with hardly any efforts involved. Serial buses like I^2C and SPI provides important communication with digital peripheral devices like e.g.

sensors. The GPIO connectors give further flexibility concerning connectivity of external components.

- **Mobility** – The physical size of the RPi, described in Table 4.1, makes it ideal for use in a small, mobile vehicular system. Also, it is powered through a micro USB connector, which means that it can be powered by e.g. portable power banks.
- **Performance** – Despite the size of the RPi, it has quite adequate performance specifications, as shown in Table 4.1. Compared to the Lego Mindstorms EV3 used in the system of [Sve15c], the RPi has 16 times more memory and a processor with multiple cores compared to a slower, single-core processor in the EV3.
- **Operating system** – With the official operating system of RPi, Raspbian OS[1], installing needed software is uncomplicated. The versatility of the operating system not only results in increased flexibility regarding e.g. middleware platforms but also allows the use of off the shelf applications for deployment of new executable code during runtime.

Many of the benefits of using the RPi is listed above, but there are also other alternatives. The Arduino[2] stands out as the most relevant alternative but is discarded for several reasons. The most obvious benefit of using an Arduino is that it offers a higher level of control regarding precise timing of sensors and switches than the RPi, as the operating system running on the RPi is not a real-time operating system[Goo13]. However, there are many disadvantages on selecting an Arduino to run this system on, among them the restrictions put on the developer regarding programming language; the Arduino only speaks C. The use of Java and OSGi is defined in the problem description of the thesis, and hence the Arduino is discarded.

Figure 4.1: USB WiFi dongle, image from http://www.edimax.com/

[1] Raspbian OS is a Linux Debian based operating system, made for the Raspberry Pi computers
[2] See https://www.arduino.cc/ for more information concerning the Arduino.

Table 4.1: Specifications for the Raspberry Pi 2 [Rpi16]

Processor	ARM Cortex-A7 900MHz quad-core
Primary memory (RAM)	1 GB
Secondary memory	micro SD card up to 32 GB
Operating system	A variety of Linux distributions Windows 10 IoT Core
Size	85 mm * 56 mm * 21 mm
Weight	45g

4.1.2 Power supply

As mentioned in the previous section, mobility is of highest importance. As the RPi use a micro USB connector for power input, a USB capable Lithium battery is chosen. The battery, a Skross Reload 5[SKR16] which is depicted in Figure 4.2, has a capacity of 5000 mAh and output voltage of 5 V (standard USB). If the RPi was to constantly drain its maximum current rating of 1A (which is highly unlikely), the battery would still last for five hours, which is acceptable for this system.

Figure 4.2: The Skross Reload 5 battery, image downloaded from http://www.skross.com/

4.1.3 Color light sensor

As described in preceding theses, the previous systems have used a color sensor for updating speed and position metrics. Although different sensors will be utilized in this system, using colored railway sleepers to some extent is still considered reasonable. The color light-to-digital converter TCS34725[aos16], is chosen as it is precise, fast and supports the I^2C protocol.

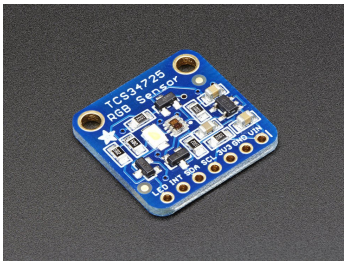


Figure 4.3: The TCS34725 breakout board, image from https://www.adafruit.com/product/1334

The TCS34725 provides color component resolutions from 8 to 16 bits precision, depending on chosen integration time, where the detected light is are divided into four component values: clear (unfiltered light); red-filtered; green-filtered; and blue-filtered light. The integration time can be set from 2.4 ms to 700 ms. The fastest integration time results in a color sampling rate of 417Hz, which is sufficient for a model of this size, as described in [Sve15c, 8.1.2]. A compact breakout board[3] with the TCS34725 chip, additional supporting circuitry including a bright is used in the system to reduce the manual labor required. The breakout, depicted in Figure 4.3, requires 3.3 V supply voltage, which is appropriately delivered by the RPi's pin header.

[3]The TCS34725 breakout can be viewed at https://www.adafruit.com/products/1334

4.1.4 NFC reader

In Section 2.2.5.2 the NFC technology is introduced, characterized by communication between two distinct counterparts; a reader and a tag/transponder. The PN532 integrated transceiver from NXP Semiconductor is a compact reader with a variety of operating modes, among other [Sema] an ISO/IEC 14443A (MIFARE) reader/writer mode. The PN532 also supports three communication protocols: I^2C bus; Serial Peripheral Interface (SPI) bus and Universal Asynchronous Receiver/Transmitter (UART), which is suitable since the RPi supports all three of them. As with the color light sensor, a breakout board[4] with support circuitry including an on-board, credit-card sized antenna is chosen. The breakout is depicted in Figure 4.4. Communication protocol is chosen by using a jumper configuration on the breakout. A 3.3V/5V supply voltage is required, depending on which protocol is being utilized. The PN532 has several features for power saving, as listed below.

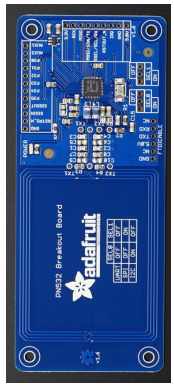


Figure 4.4: The PN532 breakout board, image from https://www.adafruit.com/product/364

- **LowVbat mode:** The chip is in virtual card mode[5] whenever an external electromagnetic field is present, and in power down mode otherwise[Sem10, 2.5]. This is the startup default mode. A state diagram showing the relations and transitions between operating modes is shown in Figure 4.5.
- **Interrupt mode:** The chip, and the breakout board, has a dedicated connector pin for an interrupt signal (IRQ). When the chip is configured to so, it sends an IRQ signal to the controller host whenever a tag enters its proximity, so that the host avoids polling the reader periodically to detect potential tags nearby.

[4]The PN532 breakout can be viewed at https://www.adafruit.com/products/364
[5]Virtual card mode means that the NFC reader emulates a tag

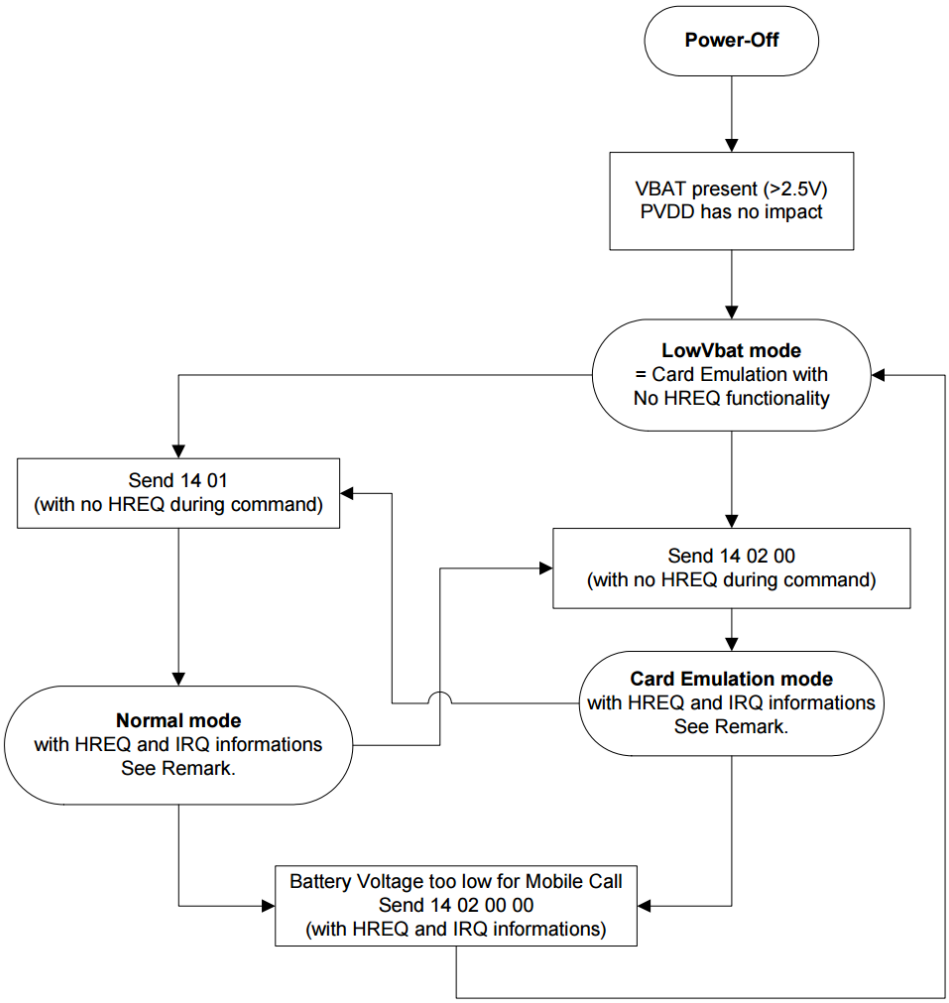


Figure 4.5: State diagram for the PN532, showing the relations and transitions between operating modes. The state diagram is presented in [Sem10, 2.5.2].

4.1.5 Xtrinsic sense board

An accelerometer and a magnetometer can be utilized either separately for measuring respectively proper acceleration[6] and magnetic vector field, or combined. A combination of the sensors outputs can be used for e.g. reading tilt compensated magnetic heading, regardless of the tilt the magnetic sensor may be subject to[7]. A breakout board containing multiple sensors, among other a digital accelerometer and a magnetometer, is used in this system. The Xtrinsic sense board, depicted in Figure 4.6, has a common connection for supply voltage, ground and the I^2C bus. As the sensors mounted on the board are implemented in separate ICs they can be addressed separately as if they did not have any common physical connections to the host. More information on the Xtrinsic sense board can be found in [Semd].

Figure 4.6: The Xtrinsic sense board, image from https://megapowertech.blogspot.com

4.1.5.1 Accelerometer

The accelerometer onboard the Xtrinsic sense board is the Xtrinsic MMA8491Q. The accelerometer has two modes; it can be configured as a simplified 45° tilt sensor or as a digital output accelerometer with $1/1000g$ sensitivity[Semc]. To maintain a low power consumption the sensor has an enable (EN) pin which controls the sampling. In Figure 4.7 a state diagram for the operational modes of the sensor is shown. When powered on, the sensor enters a 'shutdown' mode. It is now ready for sampling. When the EN pin is set high, the sensor acquires one sample from each of the three axis and stores them into a registry. The time elapsed from the EN pin is set high to when the data is available on the I^2C bus is referred to as T_{on} in the data sheet, and is stated to be 720µs in average. After the data has been transferred to the host, the EN pin should be set to logical 0 again. The procedure is repeated for each sample acquisition, but the time between a falling edge of EN and the subsequent rising edge needs to be separated by at least the reset time (T_{rst}) of 1000µs. This results in a maximum sampling rate of $f_{\max} = \dfrac{1}{720\mu s + 1000\mu s} = 581Hz$. Note that this calculation of the sampling rate does not take the time required for transferring the data from the sensor registry to the host into account, neither how much time the host needs for activating/deactivating the EN pin of the sensor. Thus, the real

[6]Proper acceleration is a term in relativity theory describing the physical acceleration experienced by an object relative to free fall, i.e. a stationary object will experience a proper acceleration $g = 9.81m/s^2$ *straight upwards* whilst an object in free fall would experience a proper acceleration of zero.

[7]Tiling a magnetometer alters the outputted data as the magnetic vector field[Ozy15]

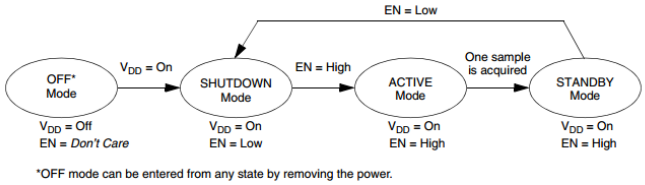


Figure 4.7: State diagram for the MMA8491Q, showing the relations and transitions between operating modes. The state diagram is presented in [Semc, 3].

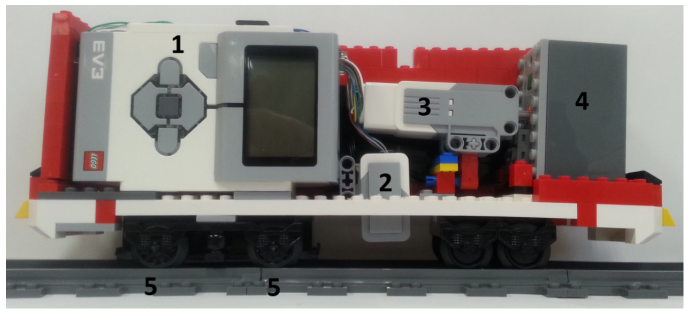


Figure 4.9: The physical composition of Overskeid's PRT pods, image from [Ove15, 4.2]. The controllable components are the EV3 smart brick (1); the color sensor (2); the speed control servo (3); the battery (4); the motorized wheel set (5)

maximum sampling rate would be lower, but probably still over 250Hz (assuming up to 2000µs for reading data and controlling the EN pin).

4.1.5.2 Magnetometer

The magnetometer located on the same board is the MAG3110. The sensor has a sensitivity of 0.1µT, noise is specified to a minimum of $0.25\mu T$ root mean square (RMS), and the full scale range is stated to be $\pm 1000\mu T$. The magnitude of the geomagnetic fields fluctuates from $25\mu T$ in South America to about $60\mu T$ over Northern China[Semb], i.e. well within limit values of the sensors. Unlike the accelerometer chip, the magnetometer does not require management of an EN pin in order to read sensor data.

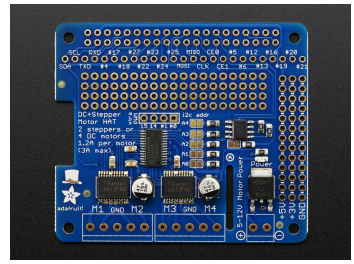


Figure 4.8: The PWM driver board, image from https://www.adafruit.com/products/2348

4.1.6 Motor controller

The preceding projects presented in Section 2.3 all utilized the same hardware platform, including the

propulsion control. To control the motor driving the
train forward, the EV3 brick controlled a servo, which turned an adjustment screw[8] on a Lego Lithium Polymer (LiPo) battery, which in turn outputted a subsequently variable voltage supplied to the motorized wheelset. The design is shown in Figure 4.9. As the RPi has no direct output compatible with the servo motor used, a different approach is necessary here. By using a dedicated PWM driver chip, the voltage from the battery in Figure 4.9 can be adjusted directly from executable code on the RPi.

The Adafruit direct current (DC) Motor HAT[9] depicted in Figure 4.8 is chosen as it's designed for the RPi. Besides hosting a PCA9685 PWM chip and support circuitry, it has a prototyping area facilitating connections to other peripheral devices. The outputted PWM voltage can be adjusted with 8-bit precision in a bi-directional manner. Further details on the Motor HAT can be found in [Ada].

Figure 4.10: A Lego Power Functions extension cable, image from http://shop.lego.com/

In addition to the supply voltage, the Motor HAT needs an external power source for driving the connected motors. This power source should not be the same that supplies power to the RPi, as electromagnetic noise can propagate from the connected DC motors. The battery used to power the motorized wheel set, and the wheel set itself, is re-used from preceding projects.

An extension cable, depicted in Figure 4.10, is used to simplify the physical connections to both the battery and the wheel set. The wire layout in the extension cable follows the diagram shown in Figure 4.11. The pins *PWM1* and *PWM2* is used for providing a modulated voltage signal to a motor.

[8]The adjustment screw of the Lego battery has 15 settings: 7 levels of positive and negative voltage, and an OFF setting. The output voltage is regulated by using Pulse Width Modulation (PWM)

[9]A 'HAT' (Hardware Attached on Top) is an add-on board for the RPi[Ada14]

4.2 Composition

4.2.1 Construction of the vehicle

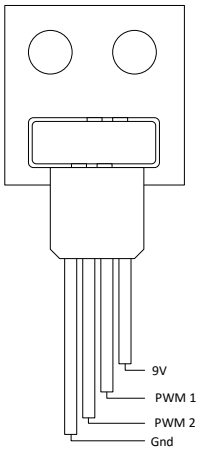


Figure 4.11: Wiring diagram for Lego PF extension cable

The electronic components presented in the previous section constitute the most central parts of the trains in this system. However, a vehicle – the train itself – is required to carry the hardware. Lego City train sets are used as building bricks to construct custom trains adapted to the on-board components.

As the color light sensor introduced in Section 4.1.3 has to be directed towards the railway surface, it is mounted to a Lego brick using a polymer-based adhesive. The assembly is shown in Figure 4.12. The remaining parts are placed either inside the vehicle body or mounted on top of it, and so no further adhesive fixtures are required.

The NFC reader described in Section 4.1.4 is placed horizontally inside the vehicle, i.e. so the antenna on the breakout board lies in parallel with the railway surface, to maximize the reading distance. Concerning the relatively low operating frequency of the reader, 13.56MHz[Sem10], the antenna on breakout board has a range of 10 centimeters. Mounted inside the vehicle, the reader has a distance of 5 cm to the surface lying underneath the railway, which is considered acceptable despite the Lego bricks obstructing a clear line of sight. The electromagnetic field should be strong enough to penetrate the relatively thin layers of plastic the Lego bricks are moulded of.

Other units placed inside the vehicle are the RPi, the attached Motor HAT and the Lego LiPo battery. As the environment has no direct influence on the inputs of these units, there is no need for further fixtures other than the containment of the units inside the vehicle body. On top of the vehicle sits the Xtrinsic sense board, clamped by Lego bricks. It is important that the board is as flush with the horizontal and vertical plane of the train as possible so that the three logical axes of the accelerometer coincides with the physical horizontal and vertical plane, and the train's direction of travel. The primary

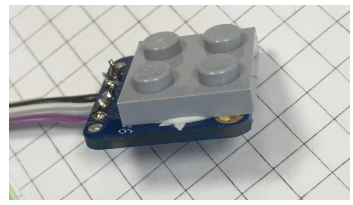


Figure 4.12: The TCS34725 breakout mounted to a 2*2 stud, flat Lego brick for interfacing

Table 4.2: Hardware addresses for I^2C enabled equipment

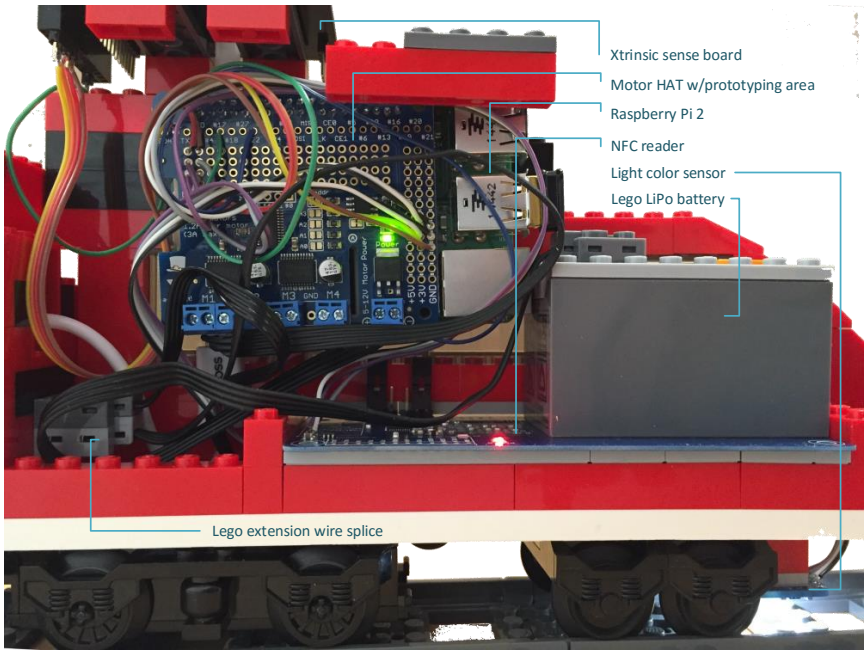
Device	Hardware address (hex)
PCA9685 chip set	`0x60`
PN532 chip set	`0x24`
TCS34725 chip set	`0x29`
MMA8491Q chip set	`0x55`
MAG3110 chip set	`0x0e`

power source for the RPi and connected peripheral
equipment is carried in a separate train car due to physical size of the battery. The resulting physical train construction is shown in Figure 4.13a and Figure 4.13b.

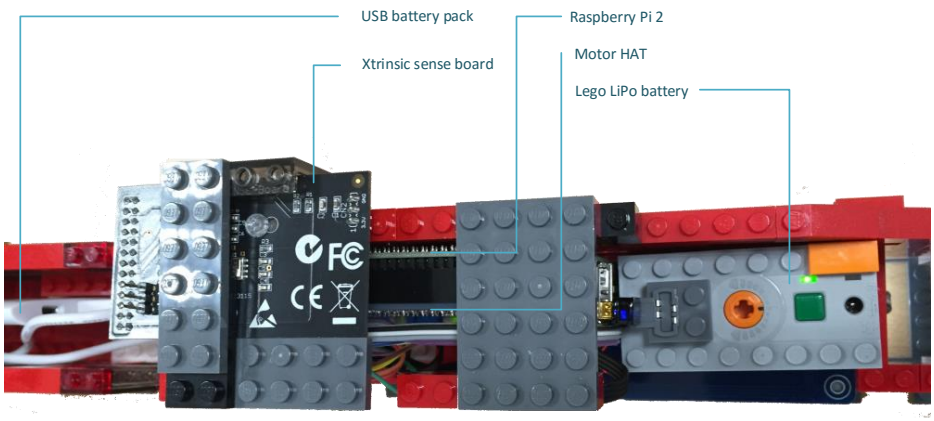
4.2.2 Schematic presentation

The peripheral equipment is connected to the RPi through the prototyping area on the attached Motor HAT. As all selected equipment supports the I^2C protocol, it is used as the primary communication bus. As mentioned in Section 2.2.2, I^2C enabled equipment often comes with a pre-coded hardware address, as is the case for all equipment listed in the previous sections. The addresses of the devices are listed in Table 4.2. Note that the Motor HAT has a 5-bit hardware address selection, which is not used here as multiple units are redundant for our system. More information on how to use the address selection jumpers can be found in [Ada].

The physical connections between the RPi and the peripheral boards are made according to the schematic logic diagram in Figure 4.14. As the Motor HAT has a suitable prototyping area, the physical connections are wired from the soldering points here. Wires run from the Motor HAT, which sits on top of the RPi, and to each of the separate breakouts. As the complete system is intended to be accommodated inside a moving vehicle, all connections are soldered to prevent weak couplings as a result of vibration.



(a) Train side view, components annotated



(b) Train top view, components annotated

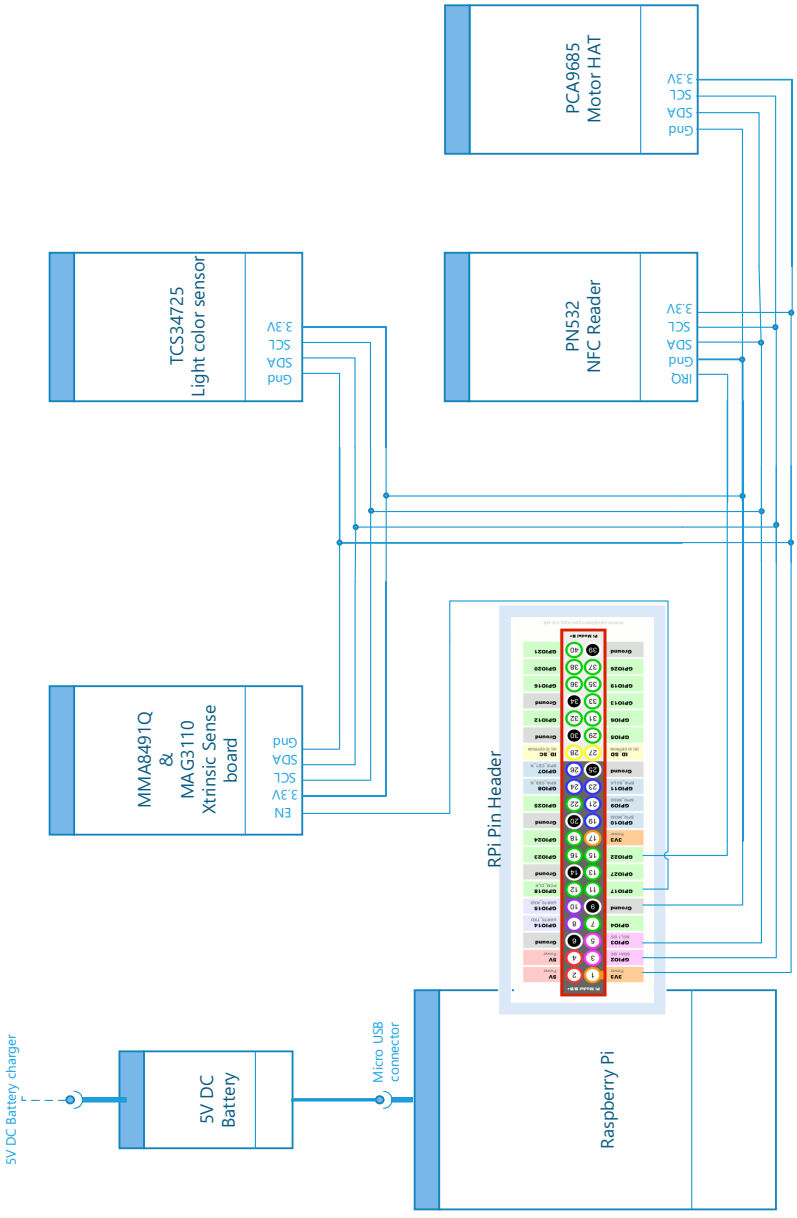


Figure 4.14: Schematic diagram showing the logic connections in the physical system.

Chapter 5
Software Architecture

All software systems have some kind of software architecture, whether it is by design or by chance. The software architecture is, in essence, an abstract way of defining the structure of a system, where each level of abstraction can be said to describe the software design in different ways, or in *architectural views* as coined by [BCK03]. The software architecture is a product of decisions made early in the design phase, i.e. when the *structure* of the system is planned.

This chapter will describe the software architecture for the system, as well as the underlying reasoning for the design decisions.

5.1 Architectural views

5.1.1 Modular view

As introduced in Section 2.2.4, OSGi is a modular framework for Java applications. The modular framework is chosen not only for the advantages given regarding modularity, but also for the robustness and agility provided by introducing lifecycle management and a service layer as well as automatic dependency resolution which is practical in a system where different modules may have a variety of different dependencies.

In Figure 5.1, a modular architectural view is presented, showing how modules in the system interact with each other through services and events. Each module, or bundle as it's coined in the OSGi context, is represented by a block in the diagram. The inter-module communication paths are represented by lines and arrows, showing the direction and type of communication.

A central aspect in Figure 5.1 is the implementation of the *whiteboard pattern*. As described in [KH04], the whiteboard pattern provides a decoupling of an event source and the respective event receiver(s). The event source, which is coined *publishers* in the diagram, gets a service reference to the Event Admin service[1], which is events are published to. The Event Admin can be viewed as a whiteboard, where event sources would write on the whiteboard and anyone with interests in the topic could choose to read what was written. Events posted to the Event Admin is redirected to those registered as EventHandlers, based on an event topic. EventHandlers registers, or subscribes to certain topics, based on what the target of interest is for the particular EventHandler, i.e. *what kind of events it would like to listen to*. Using this technique, the event source, and the event handler is completely decoupled, having no references to the types or instances of each other. What we have introduced with the whiteboard pattern is in other words an intra-Java Virtual Machine (JVM) publish/subscribe protocol recognizable from Section 2.2.1.

The event publishers generate events based on external input, and is categorized into two types:

- Event publishers associated with physical sensors and their corresponding hardware interface. These publishers create events based on data received from the sensor hardware interface, a procedure initiated by the publisher through a service call to the hardware interface.

- Event publishers based on derived events, i.e. events are created by analyzing received events published by other publishers. This category comprises both derived sensor event publishers, like the velocity publisher; and modules related to internal map and external communication like e.g. the Positioning module.

5.1.2 Pipeline view

While the previous section presented the modular structure, i.e. how modules relate to each other, this section is intended to describe how data flows through the system. As with the subsystem of [Sve15c, 5.2], the *pipe-and-filter* architecture presented in [BCK03, p. 215-216] is used in this system. The pipe-and-filter pattern is suitable for systems where streams of discrete data items are processed in a stepwise manner from input to output. The architectural view is illustrated in Figure 5.2 , showing an abstraction of how data flows in a pipe-and-filter manner through the system. The small, consecutive blocks represent data streams entering and exiting the system. The block denoted 'External systems' represents co-operating trains

[1]The Event Admin service is a compendium service in the OSGi specification, i.e. provided as an optional service by the framework provider.

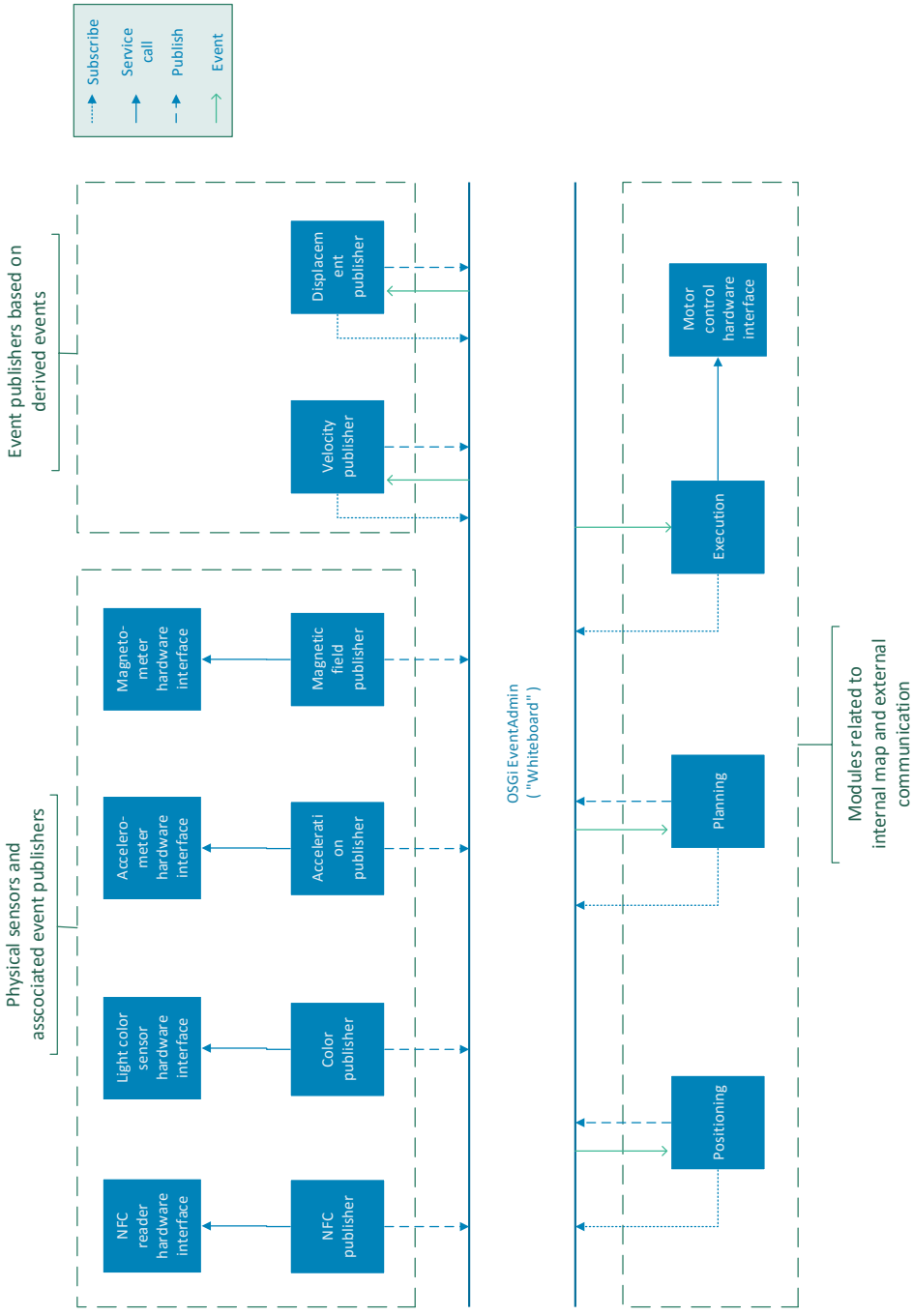


Figure 5.1: Architectural modular view

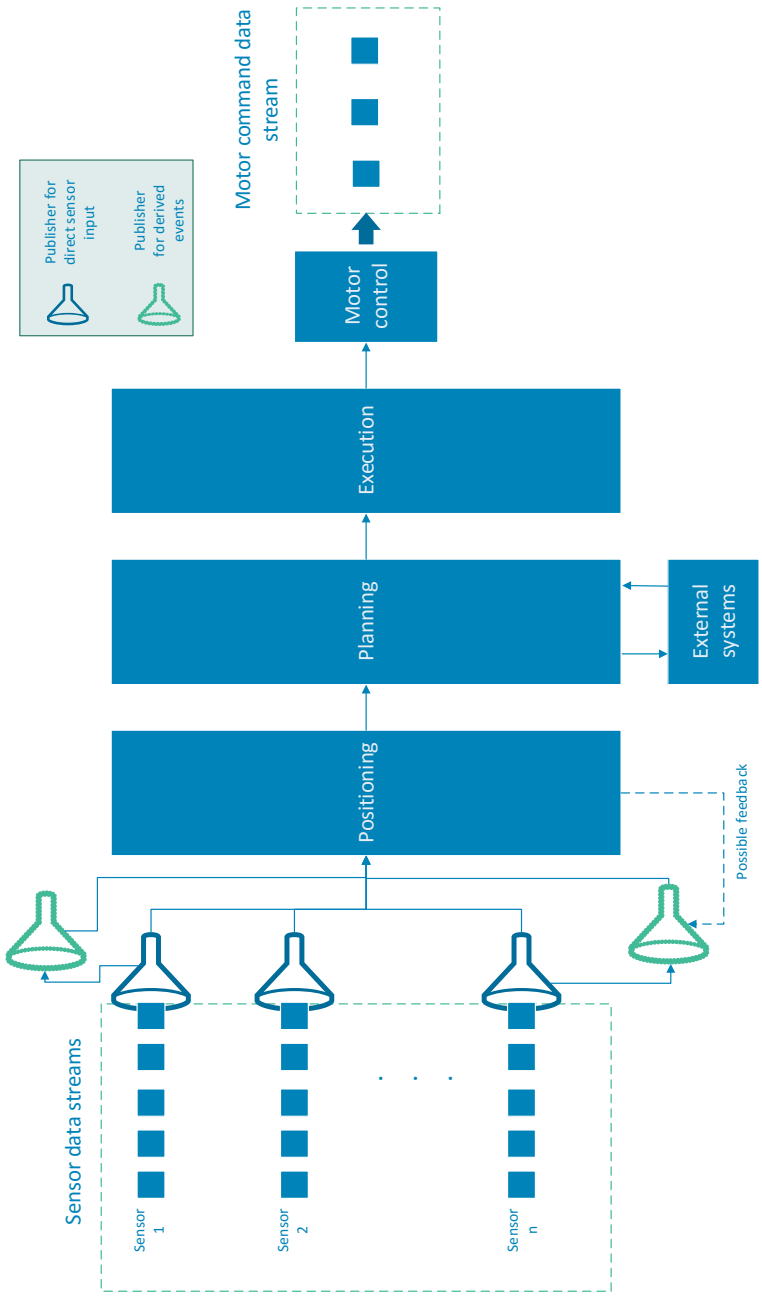


Figure 5.2: Architectural pipeline view

The stage where publishers receive and process sensor data to create and publish events is executed in a parallel manner, except for the publishers based on events derived from other publishers receive such events in a serialized manner. The events are fed through to the Positioning module, where the internal environment model is updated accordingly to the input. A feedback loop from the Positioning module to one or several publishers is feasible, considering the communication pattern presented in Section 5.1.1. Such a feedback loop is desirable if machine learning techniques are to be implemented at a later stage, e.g. to optimize sensor sample analysis.

The internal model update is tagged with a classification number, describing the accuracy of the update depending on which sensor source that triggered the update. The updated internal model is passed on to the Planning module, where the present position is analyzed and compared to the current route plan. If necessary, the Planning module communicates with the corresponding modules of other train systems traveling in the same railway system and passes on a movement plan to the subsequent module: the Execution module. At this stage, the movement plan's validity is guaranteed by the Planning module, and the Execution module decides which speed level should be passed on to the Motor Control module.

5.2 Sensor-publisher communication

As mentioned, the publishers retrieve raw data from the sensor hardware interfaces by service invocations. By letting the sensor hardware interfaces implement interfaces defined in a mutual, public 'common' module, no direct dependencies between modules are introduced. The argument for using services, which differ from the other inter-module communication in the system which relies on event propagation, is that the *timing* of sensor readings must be defined somewhere. By extracting timing implementations out of the sensor hardware interfaces, those modules become interchangeable and timing control is kept under the authority of a publisher.

The publishers use service trackers provided by the OSGi framework to retrieve temporary references to the service implementations of the sensor hardware interfaces. As an example, the UML diagram in Figure 5.3 is presented, showing the classes related to the MIFARE sensor (PN532) and the associated publisher. Note that there are none direct relations between the MifarePublisher and the PN532 module, which means that the sensor reading implementation is fully interchangeable. The outer boxes represent Java packages, which in this case all belongs to separate OSGi bundles except for the ones related to the OSGi framework.

Although some internal variations exist, all sensor/publisher pairs in the system follow this pattern. With such a high decoupling, replacing the sensor implementation with a sensor simulator can be managed solely by using the OSGi management console,

Table 5.1: Service contracts for peripheral equipment modules

Device	**Service contract**
PCA9685 chip set	ActuatorControllerService
PN532 chip set	MifareControllerService
TCS34725 chip set	ColorControllerService
MMA8491Q chip set	AccelerationControllerService
MAG3110 chip set	MagControllerService

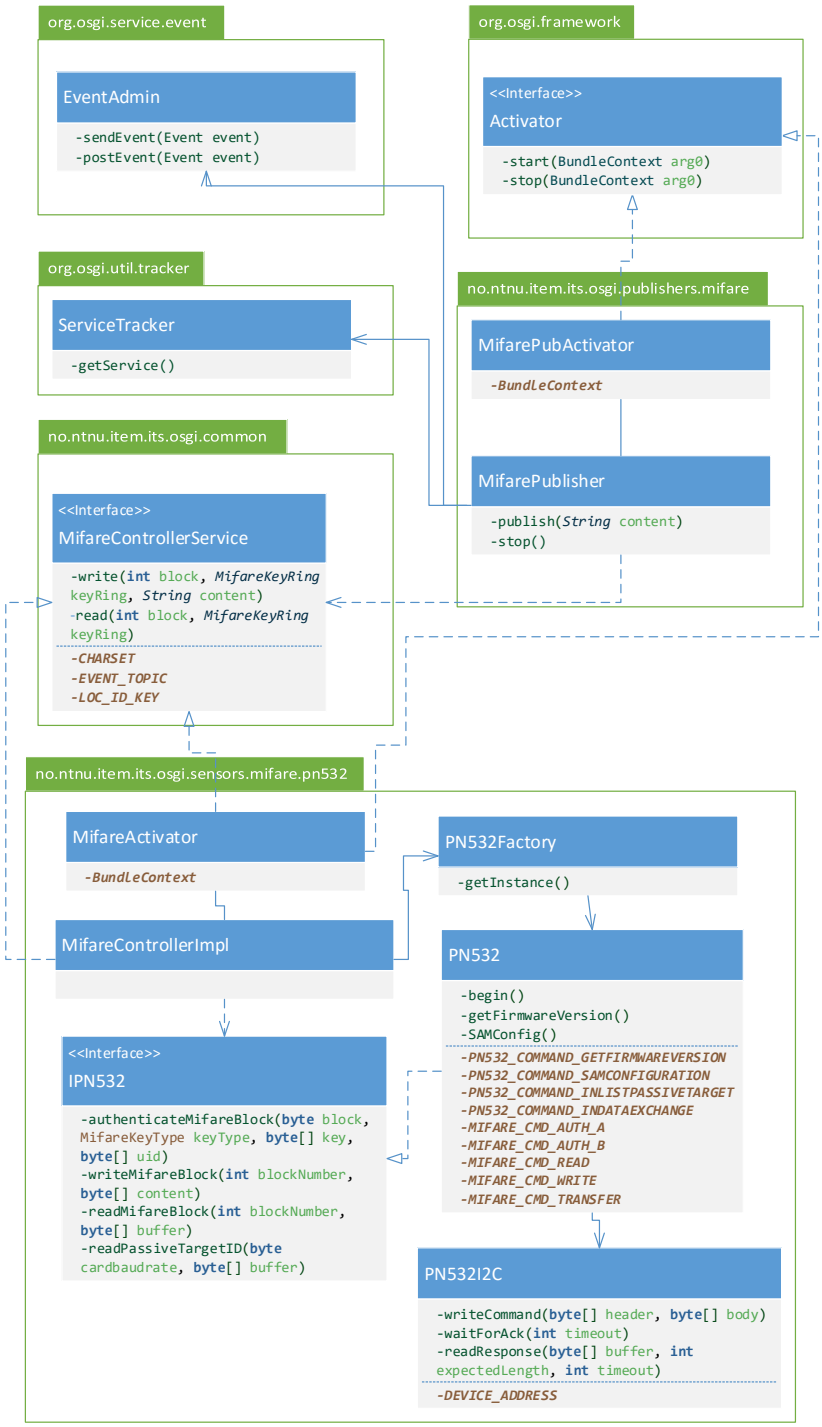
allowing for testing and debugging purposes. The only requirement on the sensor simulator bundle is that it contains a class implementing the respective service contract interface and that an instance of this class is registered as a service to the framework's service registry[2].

As mentioned above, the responsibility for timing implementation is imposed to the publisher. As multiple publishers have this requirement, a scheduling service is introduced. Using the same context as above, namely the MIFARE sensor/publisher pair, a UML class diagram is presented in Figure 5.4, showing how the MIFARE publisher relates to a scheduling service. Details not concerning timing, e.g. the implementation of the MifareControllerService and relations to the EventAdmin class is not presented here to enhance the readability of the diagram.

Like the previous pattern displayed in Figure 5.3, this pattern is recurring in the various sensor/publisher pairs. The scheduling service runs inputted sensor reading implementation at a configurable rate until it is removed, or if an uncaught exception occurs in the code.

The service contracts and the corresponding sensor hardware interfaces are listed below in Table 5.1. The definition for each of the service contract Java interfaces is listed in Table 5.2 All listed service contracts are contained in the *'common'-bundle*, and the package name `no.ntnu.item.its.osgi.common.interfaces` is skipped in the tables.

[2]Registering a service to the framework's service registry is possible through the static BundleContext property


org.osgi.service.event
EventAdmin
-sendEvent(Event event)
-postEvent(Event event)
org.osgi.framework
<<Interface>>
Activator
-start(BundleContext arg0)
-stop(BundleContext arg0)
org.osgi.util.tracker
ServiceTracker
-getService()
no.ntnu.item.its.osgi.publishers.mifare
MifarePubActivator
-BundleContext
MifarePublisher
-publish(String content)
-stop()
no.ntnu.item.its.osgi.common
<<Interface>>
MifareControllerService
-write(int block, MifareKeyRing keyRing, String content)
-read(int block, MifareKeyRing keyRing)
-CHARSET
-EVENT_TOPIC
-LOC_ID_KEY
no.ntnu.item.its.osgi.sensors.mifare.pn532
MifareActivator
-BundleContext
PN532Factory
-getInstance()
MifareControllerImpl
PN532
-begin()
-getFirmwareVersion()
-SAMConfig()
-PN532_COMMAND_GETFIRMWAREVERSION
-PN532_COMMAND_SAMCONFIGURATION
-PN532_COMMAND_INLISTPASSIVETARGET
-PN532_COMMAND_INDATAEXCHANGE
-MIFARE_CMD_AUTH_A
-MIFARE_CMD_AUTH_B
-MIFARE_CMD_READ
-MIFARE_CMD_WRITE
-MIFARE_CMD_TRANSFER
<<Interface>>
IPN532
-authenticateMifareBlock(byte block, MifareKeyType keyType, byte[] key, byte[] uid)
-writeMifareBlock(int blockNumber, byte[] content)
-readMifareBlock(int blockNumber, byte[] buffer)
-readPassiveTargetID(byte cardbaudrate, byte[] buffer)
PN532I2C
-writeCommand(byte[] header, byte[] body)
-waitForAck(int timeout)
-readResponse(byte[] buffer, int expectedLength, int timeout)
-DEVICE_ADDRESS

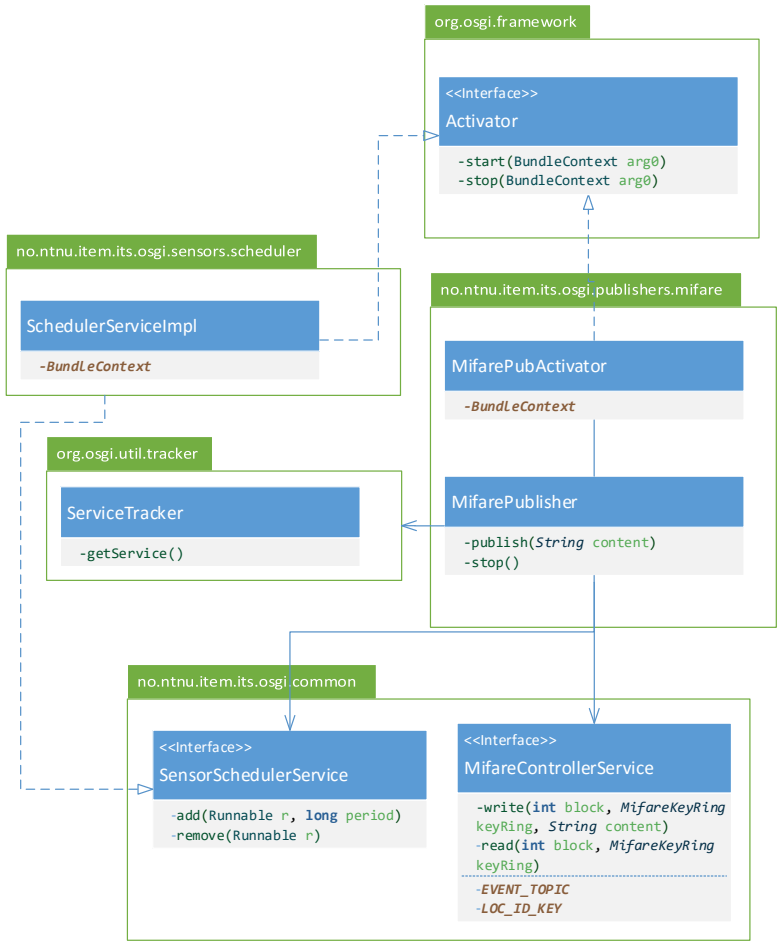


Figure 5.4: UML Class diagram for the periodic timing of MIFARE reader sampling

Table 5.2: Service contracts definitions

Service contract	**Declared methods**
ActuatorControllerService	`void send(MotorCommand c)` `void send(MotorCommand c, int speed)`

MifareControllerService	`void write(int block,`
	`MifareKeyRing keyRing, String content)`
	`String read(int block,`
	`MifareKeyRing keyRing)`

ColorControllerService	`int[] getRawData()`
AccelerationControllerService	`int[] getRawData()` `int[] getCalibratedData()`
MagControllerService	`double getRawData()`

Chapter 6
Data processing

As the title unveil, a central aspect in this thesis is to address problems concerning *self-localization* in the model Lego railroad system of the Department of Telematics. In an outdoors environment with full-scale vehicles, self-localization can be assisted by positioning systems like . However, when dealing with model scale equipment placed inside a building, such positioning systems offer neither the required signal intensity nor necessary accuracy and precision needed in a small scale environment. This chapter presents techniques to realize a self-localization mechanism, using raw data from the peripheral sensors and architecture introduced in the previous chapters. The sensor data processing presented in this chapter is logically located in the Publishers described in Chapter 5, except for the processing of MIFARE data.

As the source code for the modules which relates to the following sections is considered too comprehensive for inclusion in the thesis, the reader is referred to Section A.1.

6.1 Sensor data processing

6.1.1 Color light sensor data

The raw data delivered from the `ColorControllerService` is, as specified in Table 5.1 of the type `int[]`. More specific, the integer array is a collection of the component values of the perceived light, as described in Section 4.1.3. The order of the values are clear (index 0), red (1), green (2) and blue (3). A precision of 8 bits per component is considered sufficient to separate the various colors available in the Lego train context, as it results in 256 possible values per component. By managing with 8 bit precision, a maximum sampling frequency of 417Hz is possible[1].

[1] As other equipment use the same I^2C bus for communication, excessive sampling rates should be avoided

Table 6.1: Distances between color light sensor board and colored surface

Surface	**Distance [mm]**
Colored sleepers	7
Lego railroad bricks (gray)	11
Table	14

The raw color data is very detailed, but processing is necessary before publishing an event based on a sample further into the system. A color sample needs to be categorized, hence the introduction of discrete color constants, the `EColor enum` in the `common` bundle. The values of the EColor enum follows the colors of the flat Lego bricks placed on the railroad, and the Lego railroad bricks themselves. The colors are listed in Table 7.1. To classify the colors a two-stage procedure is presented, of which the first step involve sample collection and the second stage consist of constructing an algorithm.

6.1.1.1 Collecting samples

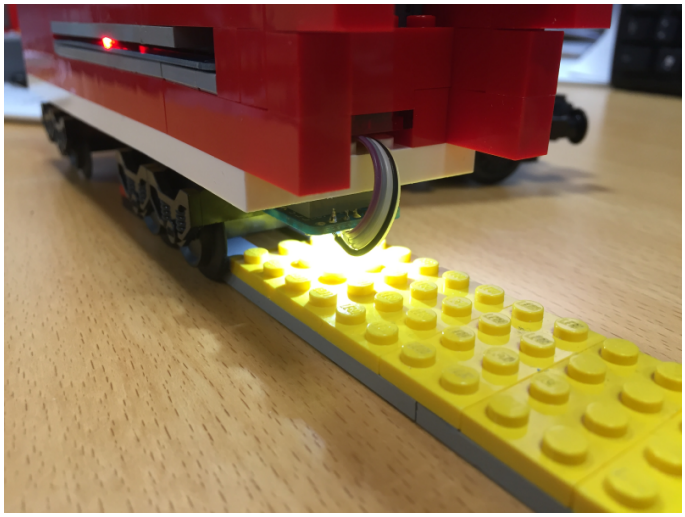
As the human interpretation of a color (e.g. 'green') is, in fact, a range of colors rather than a single unique dot in a color map, we need to define what colors to expect when reading the sensor data. As the multitude of colors in the system is limited, each single color should be mapped. A testbed is constructed to provide a continuous colored surface the color light sensor can travel back and forth over to simulate realistic movement. The testbed is depicted with a yellow surface in Figure 6.1, and the distances from the sensor board to the colored surfaces follows Table 6.1.

For each color in Table 7.1, 10.000 sensor samples is collected and stored while the vehicle is moved back and forth over the colored area of the testbed in a fluctuating rate, to cover scenarios with various speeds, etc. The sample set is analyzed using statistical methods to produce a result set consisting of mean values and standard deviations for the components, which forms a basis for a color classification algorithm.

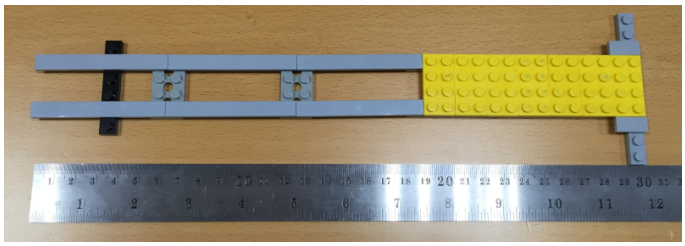
6.1.1.2 Color classification

Using the results from the color testbed experiment, an algorithm for color classification is presented in Algorithm 6.1. The distance between a color sample and each of the pre-collected colors from the previous stage is calculated. The pre-collected color that is evaluated to lie closest to the sampled color is considered to be the estimated color of the surface and is returned by the algorithm *unless* the distance is too large (above an appropriate boundary level), and the algorithm evaluates to `UNKNOWN`. A

Figure 6.1: Color mapping testbed



(a) Testbed constructed for color mapping, with vehicle on top. The distance between the sensor board and the colored surface is the same as when the vehicle travels on the model railroad.



(b) Testbed for color mapping, with measure. The colored surface is 14 Lego studs wide, i.e. 112 millimeters.

boundary level of 20 would correspond to a precision requirement of 2.6%, which is quite strict.

A different approach than calculating the color distance between sampled and pre-collected colors is to use the pre-collected color values as filters. For each component, if the raw color value lies within one standard deviation from the pre-collected mean, it passes the filter, else the filter is discarded. If a single filter remains when the procedure is over, the color is found. The process is presented in Algorithm 6.2. In a normal distribution, 68% of the values lies within one standard deviation from the mean.

The technique giving the most promising results is tested on a circular test track

Algorithm 6.1 Color classification distance algorithm

```
Sub classify(rawColor):
    Let ColorStatistics be List of FixedColors
  % where each fixed color is represented by an
  % EColor type enum and a List of
  % [clear, red, green, blue] component mean values

    Let minDiff = 10000
    Let minColor = Null

    For fixedColor in ColorStatistics:
      Let diff = compare(rawColor, fixedColor)
      If diff < minDiff:
        Let minDiff = diff
        Let minColor = fixedColor
      End If

    If minDiff > DiffBoundary:
      Return UNKNOWN % Do not approximate too much
    End If

    Return minColor.Type
End Sub

Sub compare(rawColor, fixedColor):
    Let redDiff = rawColor.Red*255/rawColor.Clear -
    fixedColor.Red*255/fixedColor.Clear
    Let greenDiff = rawColor.Green*255/rawColor.Clear -
    fixedColor.Green*255/fixedColor.Clear
    Let blueDiff = rawColor.Blue*255/rawColor.Clear -
    fixedColor.Blue*255/fixedColor.Clear

    Return Abs(redDiff) + Abs(greenDiff) + Abs(blueDiff)
End Sub
```

where a variety of colored sleepers are mounted to investigate if any sleepers are bypassed during operation.

6.1.1.3 Timing

Periodic readings of the sensor are useful when the vehicle is moving for continuous detection of the change in surface color, i.e. the passing of sleepers. According to

Algorithm 6.2 Color classification filtering algorithm

```
Sub filter(rawColor):
  Let ColorStatistics be List of FixedColors
  % where each fixed color is represented by an
  % EColor type enum and a List of
  % [clear, red, green, blue] component mean values,
  % and a List of
  % [clear, red, green, blue] component standard deviation values.
  Let Filters = ColorStatistics

  For component in rawColor:
    For fixedColor in Filters:
      If rawColor.component not within fixedColor.componentMean +-
        fixedColor.componentStdev:
          Filters.remove(fixedColor)
      End If

  If Filters.size = 1:
    Return Filters.getFirst.getType
  Else:
    Return UNKNOWN
  End If
End Sub
```

[Sve15c, 8.1.2], a timing period of $16ms$ would allow the model train to travel at speeds up to 1 $^{m}/_{s}$ and still register all passed sleepers, which is satisfactory.

6.1.2 MIFARE tag readings

As described in Section 2.2.5.2, Mifare-enabled tags has a built-in EEPROM with a capacity between 1 to 4 kilobytes of memory. A layout map of a small portion of the memory is depicted in Figure 6.2. When writing to MIFARE memory, operations are executed on block-level, which means that data of less than 16 bytes has to be padded with leading 0's before writing. Likewise, when reading from the MIFARE chip leading 0's should be trimmed before passing the data on to the recipient. As mentioned at the beginning of this chapter, this processing of byte array data does not take place in the publisher module, but rather in the `MifareControllerImpl` class of the `pn532` bundle. The reason for this design choice is that it makes the API for the PN532 bundle more universal and easier to use for others[2]. The implementation

[2]An experimental version of the bundle was provided to prof. Alexander Kraemer at the Department of Telematics for educational purposes

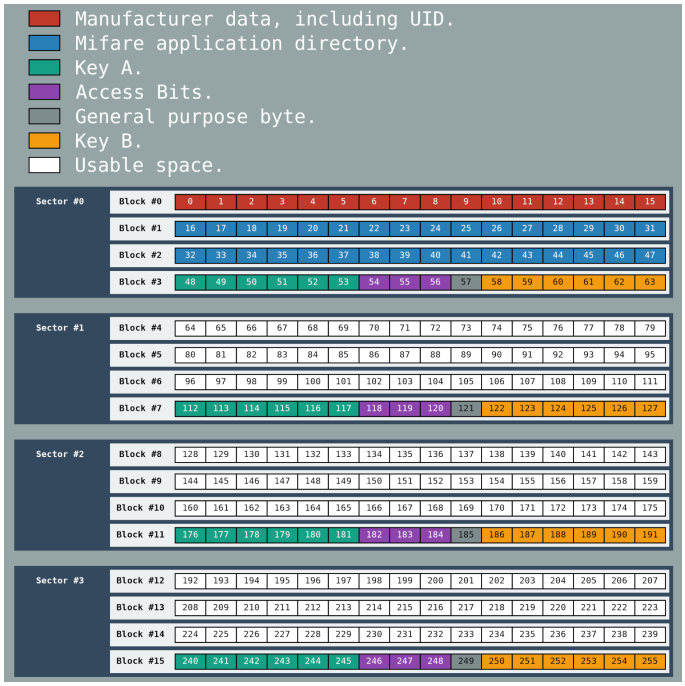


Figure 6.2: Layout of the first three sectors of a MIFARE tag's EEPROM memory. Each block consists of 16 bytes, each sector consists of four blocks, where the last block contains encryption keys for access to the preceding blocks.

allows the service consumers of the `MifareControllerService` to write and read Java `string`s in the UTF-8 character encoding. With UTF-8, all "regular" characters and digits are encoded with an 8-bit length encoding, i.e. a single MIFARE block can accommodate strings up to 16 characters length.

6.1.2.1 Timing

In Section 4.1.4 the interrupt signal of the PN532 chip was introduced. Ideally, one would use this signal to trigger a MIFARE reading, i.e. read a tag whenever it is within the range of the PN532 card. Other possibilities are periodical, continuous retries, or using other sensor input to trigger the reading. Periodical readings should be kept at half the pace required for the color sensor, taking the size of the antenna and thus, the proximity area for the NFC reader into account, i.e. reading at $32ms$ intervals.

6.1.3 Accelerometer data

The raw data read from the accelerometer is an array of binary values with one value per axis. The least significant bit (LSB) of each array element has a resolution of $1mg$. The accelerometer data can be useful for multiple purposes, e.g. emergency situation detection (tilt or sudden stop). To provide valuable data to other modules in the system, the `Accelerometer Publisher` is tasked to convert the raw data into SI-units before publishing the data wrapped in Events to the EventAdmin. The conversion to SI-units is presented in Algorithm 6.3.

Algorithm 6.3 Conversion of bit value acceleration in mg to SI-units

```
Sub convert(bitValue):
  Let g = bitValue / 1024.0
  Return g * 9.81 % meters per square second
End Sub
```

The Accelerometer Publisher wraps the data from each of the axes in an `Event` accompanied by a time stamp denoting at what time the data was read[3,4], and pass the event on to the EventAdmin, which notifies the potential modules subscribing to those events.

6.1.3.1 Derived event generation

In the case with the accelerometer, a processed event makes little sense by itself. However, along with the time stamp, a sequence of events can provide insight in the relative change of velocity and further on, the displacement of the vehicle. The relations between relative acceleration a, change in relative velocity, Δv, and change in relative displacement, Δs, is illustrated in Figure 6.3. The y-axis denotes both acceleration in $^m/_{s^2}$, velocity in $^m/_s$ and displacement in m.

$$v(t) = \int a(t)\mathrm{d}t \implies \Delta v = \int_{t_0}^{t_1} a(t)\mathrm{d}t \tag{6.1}$$

$$s(t) = \int v(t)\mathrm{d}t \implies \Delta s = \iint_{t_0}^{t_1} a(t)\mathrm{d}t \tag{6.2}$$

[3]The time stamp is generated by use of the static `System.nanoTime()` method, in the form of a `long`.

[4]As the passing and reception of events are executed in an asynchronous manner; events may arrive out-of-order or be delayed. Timestamps must, as a consequence, be appended as early as possible after the sensor data is read.

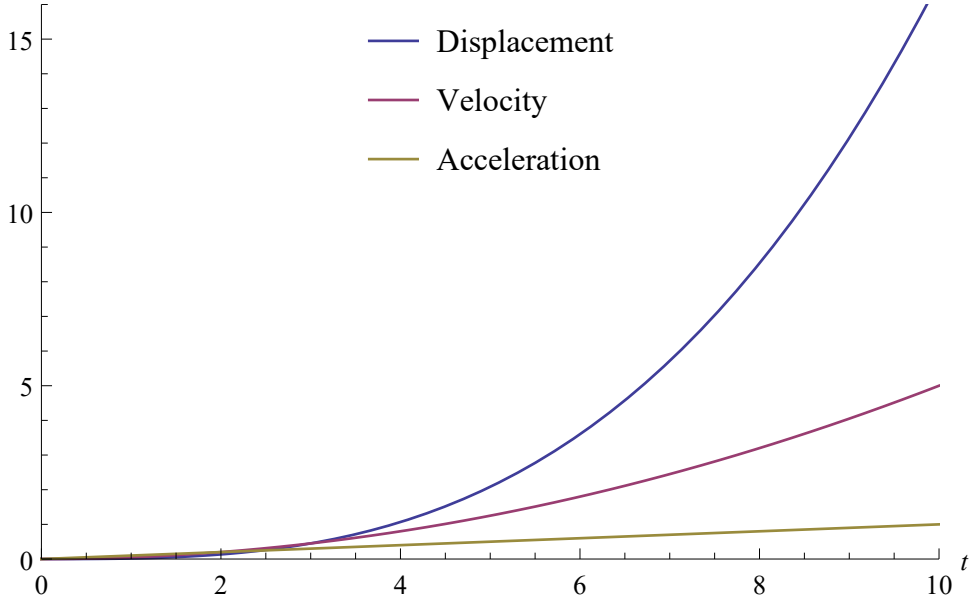


Figure 6.3: Relations between acceleration, velocity and displacement

The mathematical relations are given in equations (6.1) and (6.2). Both equations are based on the function for acceleration with respect to time, which is a function with unknown coefficients as a variable acceleration is expected. By looking at two consecutive events from the Accelerometer Publisher, both events containing axial, momentary acceleration and the timestamp for the event, a linear function can be found for the secant line intersecting both (a_0, t_0) and (a_1, t_1) by applying the two point formula solved for $a(t)$ in (6.3). Note that the following formulas must be applied to each axis to find the vectorial acceleration in three-dimensional space. However, as the vehicles movements are restricted to a single horizontal dimension[5], these calculations are only applied for the x-axis value, as it is this axis that is of interest when looking at normal train operation.

$$\begin{aligned} & f(x) - f(x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \\ \Rightarrow \quad & a(t) - a(t_0) = \frac{a(t_1) - a(t_0)}{t_1 - t_0}(t - t_0) \\ \Rightarrow \quad & a(t) = a_0 + (a_1 - a_0)\frac{t - t_0}{t_1 - t_0} \end{aligned} \tag{6.3}$$

[5]Although the trains move in three-dimensional space, in reality, the accelerometer chip sits in a fixed position relative to the vehicle body, thus will the x-axis of the sensor be the only effective axis

As (a_0, t_0) and (a_1, t_1) will differ for each event received, so will the coefficients of the function in (6.3), which means dynamically generating a new function for each iteration is required. The implementation of such a feature is quite straight forward using the functional programming features of the Java Development Kit (JDK) version 8, as shown in Source Code 6.4.

Source code 6.4 Generation of a linear function for a line intersecting two points

```
public Function<Double, Double> getAccelAsFuncOfTime(
         double a_0,
         double a_1,
         double t_0,
         double t_1)
{
  return new Function<Double, Double>() {
    @Override
    public Double apply(Double t) {
      return a_0 + (a_1 - a_0) * ((t - t_0) / (t_1 - t_0));
    }
  };
}
```

Using the `Function` dynamically generated in each iteration, the change in velocity during the time period between two samples can be found by numerical integration. The library `org.apache.commons.math.analysis.integration` is utilized to facilitate the numerical integration. More specific, the class `TrapeziodIntegrator` is chosen, as it implements the *trapeziodal rule* which suits the linear acceleration in this context. As the source code for this implementation is more comprehensive, the reader is referred to Section A.2.

As the calculation of the displacement delta, Δs, only requires an extra integration step as shown in (6.2), the details of an implementation is omitted here as it would be repetitive. Following the modular architecture, the derived event generation is

6.1.3.2 Timing

The accelerometer has to be frequently polled to record motions of significance. The maximum sampling rate mentioned in Section 4.1.5 is considered to be multiple times higher than what is needed to register necessary changes in acceleration. As the acceleration measurements aren't used directly, but as a basis for further processing, the time required for data processing has to be taken into account to prevent a situation where data is read faster than it can be processed, leading to a system congestion. The sampling rate is suggested to be set at $50ms$.

6.1.4 Magnetometer data

The magnetometer sensor provides, like the accelerometer, raw data in three axes. In the case for magnetic field readings, the unit used is *Tesla*, and in this particular case, the sensor has a precision of $0.1\mu T/LSB$. As mentioned in Section 4.1.5 the geomagnetic fields fluctuates within the microTesla-scale. Consequently, it is decided that the `MagPublisher` should provide events containing magnetic field data denominated in the μT scale, which is achieved by dividing the raw values by 10. As with the acceleration events, the magnetic field reading events should also be accompanied by a time stamp.

6.1.4.1 Calibration

The magnetometer may need to be calibrated before use. For simplicity, the assumption of a horizontal alignment of the magnetometer chip is made, which means that only the x and y-axis needs to be calibrated. The calibration procedure chosen is manual, but simple, as briefly described in [Zam11]. The procedure consists of iterating through two phases:

Periodically log magnetometer readings to a formatted file, with the readings' x and y values comma separated on each line, while turning the magnetometer 360° clockwise, in a horizontally stable position.

Plot the file to a point plot with the corresponding axis values. The points should form a circle. The center of the circle should be (approximately) positioned in the plot origin point $(0, 0)$. Add the necessary constant values to the magnetometer publisher to adjust the published values. Repeat until a satisfactory result is achieved.

6.1.4.2 Derived event generation

Based on the raw magnetic field readings, a magnetic heading can be calculated. As described in [Ozy15], the 23 equations for calculating a tilt-compensated compass heading using accelerometer data requires quite many mathematical operations, and is out of the scope of this paper. A simpler variant where a magnetic heading is calculated using only the x and y-axes components from the magnetometer is presented in (6.4), taken from [Hon], which is based on the magnetometer chip to be horizontally positioned. The result of the calculation is a magnetic heading denoted

in degrees, as a number $0 < \theta < 359$.

$$heading = \begin{cases} 90 - arcTan(\frac{x}{y})\frac{180}{\pi} & y > 0 \\ 270 - arcTan(\frac{x}{y})\frac{180}{\pi} & y < 0 \\ 180.0 & y = 0, x < 0 \\ 0.0 & y = 0, x > 0 \end{cases} \tag{6.4}$$

As the magnetometer picks up electromagnetic fields, it is interesting to investigate if there are any resulting interference when the motor unit is switched on and off, as electromagnetic motors are known to produce electromagnetic noise.

6.1.4.3 Timing

The heading of a vehicle will not be subject to rapid change, as the traveling speed is restricted. This implies that a low sampling rate of e.g. $5Hz$ is satisfactory for this sensor.

6.2 Actuator control

The Motor HAT with the on-board PWM chip has, as mentioned in Section 4.1.6, 8-bit precision on the output voltage adjustment. As a software service, the hardware interface module provides a `setSpeed` method which takes two arguments: a command describing the direction[6]; and an integer between 0 and 255 inclusive describing the actual speed.

6.3 Strategies for data utilization

In the following paragraphs, strategies for how the collected data is utilized to improve the self-localization function of the Lego train system. The functionalities described in this section is primarily the responsibility of the `Positioning` module.

6.3.1 Colored sleepers

As in related projects, information embedded in the coloring of the railway surface itself is utilized. In addition to registering the regular sleepers (gray color), signal colored sleepers are used. The information expressed by the various colors are listed in Table 6.2. Note that the information in a gray sleeper also applies to the other colors, as it too indicates movement.

[6]Direction commands can be either `Forward|Backward|Release`.
[7]The connector definitions are explained in [Sve15c, 4.2]

Table 6.2: Sleeper colors and their encoded information

Color	Encoding
Gray, *	Moved $32mm$ in travel direction
Blue	NFC tag in proximity
Red	Point section entry connector[7]
Green	Point section through connector[7]
Yellow	Point section divert connector[7]

6.3.2 MIFARE balises

By placing NFC tags beneath the railroad surface at designated positions, the passing vehicles can read location data out of the MIFARE memory of the tag. The tags are encoded with the unique identifier of the individual railway brick it's positioned under; an identifier searchable in the internal railroad model introduced in [Sve15c, 6]. A brick identifier is an eight-digit number represented as a `string` generated during the railroad layout design phase by the design tool Bluebrick[8], and subsequently stored in *block 42* in the memory of a tag. The tag is placed underneath the corresponding physical railroad brick in a centered fashion. The middlemost sleeper of the same brick is marked with a blue sleeper color by Table 6.2.

When the passing vehicle reads the location information stored in block 42 of the tag, a reliable position update can be made without the concerns of being 'out of sync' with counting sleepers, etc. This strategy has a resemblance to the Eurobalises of ERTMS introduced in Section 2.1.

6.3.3 Displacement based on acceleration readings

As described in Section 6.1.3, the accelerometer can be used to calculate both changes in velocity and displacement delta, i.e. how far the vehicle has moved in the interval $[t_0, t_1]$. The `VelocityPublisher` keeps a state concerning momentary velocity, which is synchronized and reset before starting the engine. It publishes then velocity each time it is updated, and so follows the periodic timing of the accelerometer readings. The velocity events can be used for publishing information relevant for other vehicles in proximity, as it may give an estimate on how long time this vehicle will use to finish the ongoing railroad track section.

Calculating displacement delta is the task of the `DisplacementPublisher`. The displacement delta calculation is as shown earlier the double integral of the accel-

[8]To alleviate this process, the source code of Bluebrick has been modified by adding a button to the context menu of a right mouse click on a selected brick. The identifier of the selected brick is consequently copied to the clipboard.

eration function of time, and can be derived from two subsequent velocity delta events. If a displacement delta with tolerable accuracy can be calculated from the acceleration readings, a position update can be executed based on the distance the vehicle has moved in the interval $[t_0, t_1]$. Object instances at a layer of the internal model coincide with the physical railroad bricks, so interpreting how far the vehicle has moved in the internal model based on a calculated metric is feasible as long as the accuracy of the calculation is relatively precise compared to the actual displacement.

As depicted in Figure 5.2 a feedback loop from the Positioning module is possible. In the case of a displacement delta module, the feedback loop can carry information on actual displacement whenever a reliable position update based on e.g. a MIFARE balise passing. The 'actual displacement'-information can, in turn, be analyzed and compared to the displacement calculations made in the same time frame, and optimization of the calculation can be evolved over time. This optimization is, however, a complex subject, and further investigation is thus precluded in this thesis.

6.3.4 Magnetic heading as a position approximation assistant

The magnetic heading can be used in several ways to improve the positioning of the vehicle. Below are two proposals to such strategies.

6.3.4.1 Using compass directions

In the design phase, the compass directions of the physical Lego layout may be embedded in the logical map. By comparing last known position and travel direction with a sampled magnetic heading and the information embedded in the map, the positioning system can estimate whether or not it has e.g. entered or left a turn, and thus what the position should be updated to. The structured map XML data provided by Bluebrick already contains some information regarding the orientation of bricks, as depicted in the XML snippet in Figure 6.4. To use this orientation directly, the layout must be placed so that north coincides with what is upward in the map design, or measurements must be made so that the deviation between relative north in the map and magnetic north is found and the information must be included in the logic map structure. As both the magnetic heading and the orientation of a rail

```
<Brick id="31352113">
  <DisplayArea>
    <X>91.92307</X>
    <Y>131.8762</Y>
    <Width>10.74999</Width>
    <Height>17.375</Height>
  </DisplayArea>
  <MyGroup />
  <PartNumber>2867.8</PartNumber>
  <Orientation>270</Orientation>
  <ActiveConnectionPointIndex>0
    </ActiveConnectionPointIndex>
  <Altitude>0</Altitude>
  <Connexions count="2">
    <Connexion id="c35137744">
      <LinkedTo>c66244779</LinkedTo>
    </Connexion>
    <Connexion id="c62848713">
      <LinkedTo>c43701031</LinkedTo>
    </Connexion>
  </Connexions>
</Brick>
```

Figure 6.4: XML snippet showing structure of rail brick element. In addition to type description (PartNumber) and connected elements (Connexions), the orientation is given in clockwise degrees from relative north in the map (upward).

brick is denoted in degrees, resolving the magnetic direction of a brick is a simple procedure as long as the mentioned deviation is found.

6.3.4.2 Continuous comparison

A simpler, but a less precise method is to compare incoming samples continuously to previous samples. In its most trivial form, this approach would involve only the previous and the current sample, and the comparison may conclude whether or not the vehicle is turning or driving straight forward. More sophisticated methods can be utilized, e.g. interpolating through a set of samples, let's say the ten most recent samples, and analyzing the result to conclude.

6.4 Merging sensor data streams

The sensor data streams flow in parallel in the system, and must at one point be merged in the *Positioning* module. As multiple events may be received by the module concurrently, the module is well suited to be subject to model-based design and is thus developed in Reactive Blocks. The OSGi project *no.ntnu.item.its.osgi.train.positioning*'s application block is depicted in Figure 6.5.

The module handles the published sensor events and converts them into map updates, which is returned to the event bus. Alongside with the map update is a number indicating the grade of estimation the particular map update is subject to, which is based on what kind of method was used to generate the update. This way the receiver of the map update, in this case, the *Planning* module, can compare an incoming update to previously received updates and resolve how to relate to the update. E.g. if an update with a high approximation grade is received two seconds after an update with significantly higher certainty, and the second update claims a position which is unrealistic concerning the recently received update, the second update can be viewed as excessive and may be rejected by the Planning module.

6.5 Integration with existing collision avoidance system

When the position is updated, using one or several of the techniques presented in the previous sections, the vehicular system needs to communicate its updated position to the other train in the system, at least the ones in its vicinity. As the same structure for describing the internal model of the map is used here as in [Sve15c], no breaking changes has to be introduced to the collision avoidance system. However, with a more precise and reliable positioning system, the collision avoidance system can be modified to take advantage of the mentioned improvements.

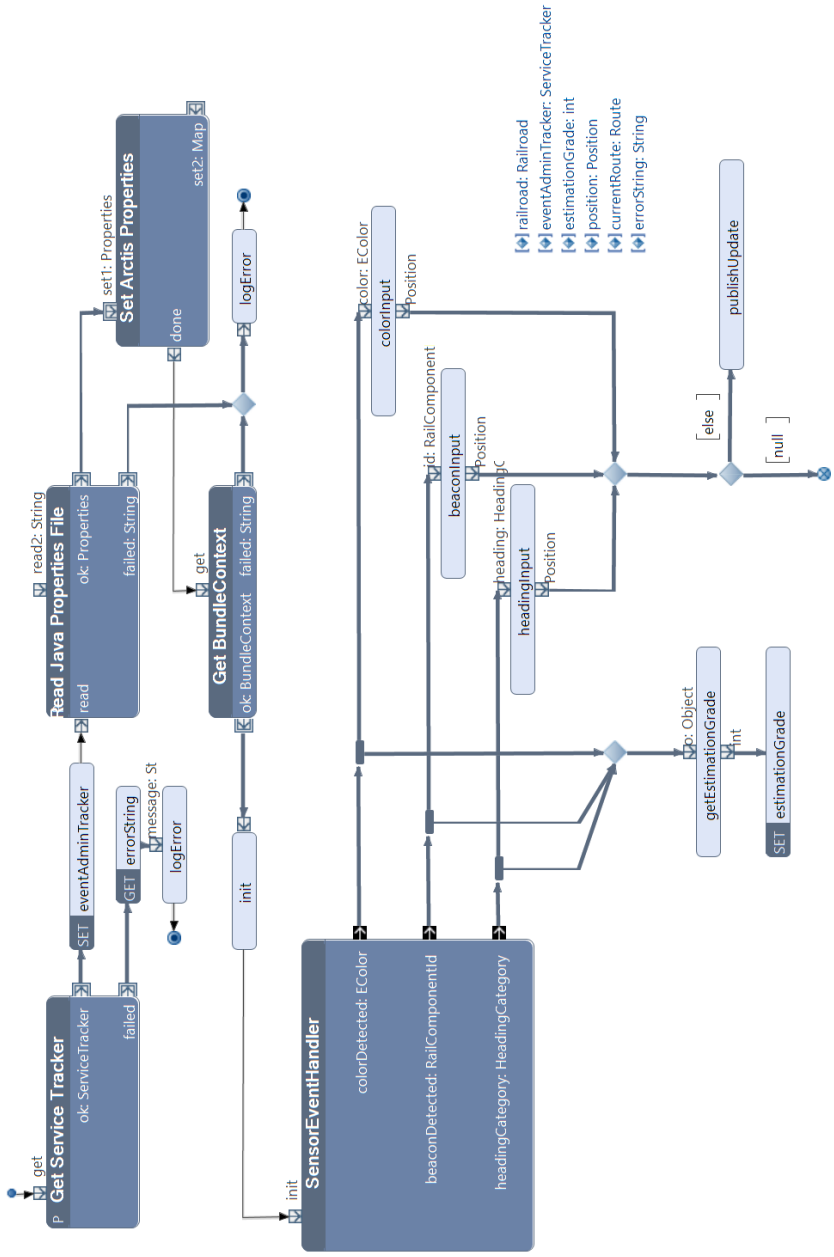


Figure 6.5: Positioning module's application block

In [Sve15c, 7.2], *Lockable resources* are introduced along with the implementation of a *LockHandler*. As the LockHandler is implemented in Reactive Blocks, re-using those same components in this system is trivial, integrating the LockHandler block into the *Planner* module of this system. The LockHandler takes Lockable Resources as inputs and provides output to the system when it has communicated with the corresponding modules of the other trains in the system. What the context of the *Lockable Resources* is, is up to the surrounding system to define.

Previously, the Lockable Resources has been intersections and entire stretches of railroad track in between intersections. The coarse partitioning resulted in a restriction of a single train per track section, even if waiting trains was traveling in the same direction and hence could follow the leading train. The waiting trains had to wait their turn at stations/intersections at the end of these tracks until the train occupying the track had finished.

By defining Lockable Resources to be a smaller building block than a complete section, i.e. a single rail brick, a more efficient system is within reach. Algorithms needs to be adapted in the *Planner* module for this behavior to cope with possible meeting collisions before such situations occur, but it is more a technicality than a real obstruction. The increased efficiency can be accounted for in a situation as depicted in Figure 6.6. If e.g. a train is waiting for another train to finish the track in front of it, both trains traveling in the same direction. The waiting train can commence on the first part of the track when it receives a release message on those track parts from the train in front, i.e. when the first train has e.g. passed a MIFARE tag along the track. This way tracks are sectioned into smaller pieces whereas safe transit is a fact as long as only one train is occupying a section at a time. Releasing segments can also be based on other sensor signals, e.g. the detection of a completed turn. Velocity readings from the first train can besides act as an upper limit for the speed allowed for the second train, as there's no point for the second train to ride faster than the first.

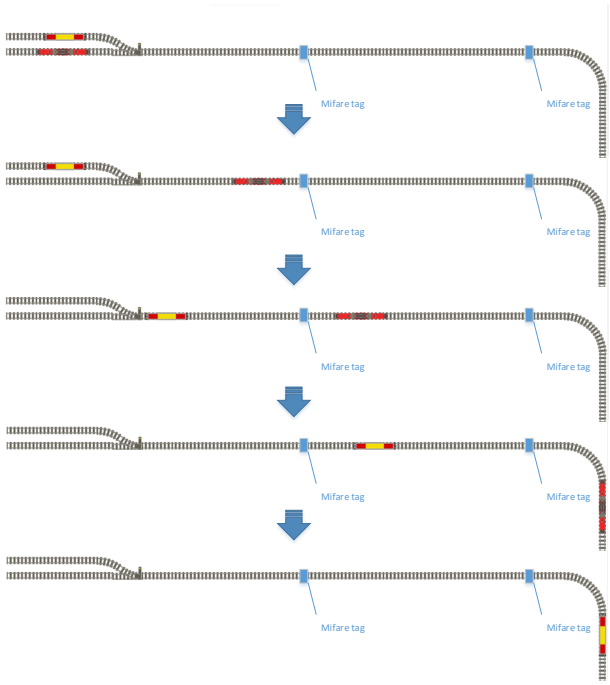


Figure 6.6: A case example for following trains

Chapter 7
Results

The sampled data that forms the statistical basis for this chapter is collected utilizing the architecture defined in Chapter 5 by registering an *EventHandler* to the framework, handling the events of interest. The EventHandler implementation takes care of logging the properties in focus for the specific experiment. Using this approach, not only the sensor hardware interfaces and publishers are tested, but the interaction with the framework as a whole as well.

In this chapter, experimental results are presented and explained, while the discussion regarding the results is carried out in Chapter 8.

7.1 Color light sensor

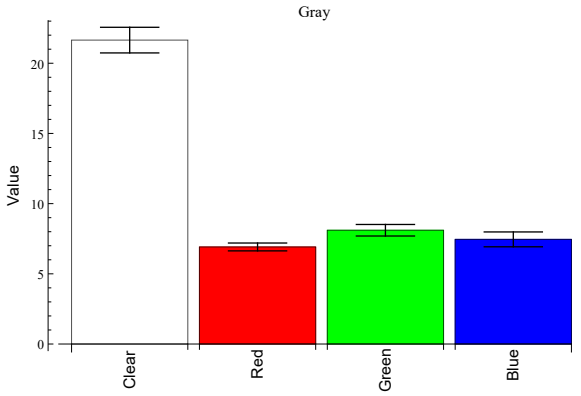
7.1.1 Pre-collected samples

As mentioned in Section 6.1.1.1, a sampling procedure on a testbed must be carried out prior to defining the color enumerations. The samples are collected using a sample interval of $10ms$, and a total of 10.000 samples are collected *for each* color. In Table 7.1, the results from the various testbed configurations are presented. Each testbed configuration result is listed with the respective color component sample means, $\bar{x}$, and standard deviations, σ.

Table 7.1: Statistics from color testbed sampling

Testbed	clear $[\bar{x}, \sigma]$	red $[\bar{x}, \sigma]$	green $[\bar{x}, \sigma]$	blue $[\bar{x}, \sigma]$
Gray	21.64, 0.91	6.91, 0.27	8.10, 0.41	7.45, 0.52
Yellow	138.16, 6.52	59.47, 2.60	55.81, 2.66	24.97, 1.23
Green	35.53, 1.84	7.18, 0.70	18.40, 0.77	11.21, 0.74
Blue	50.99, 2.29	6.48, 0.62	16.96, 0.81	29.67, 1.36
Red	35.52, 2.19	24.22, 1.19	7.37, 0.74	7.10, 0.80

Figure 7.1: Result set from color sampling on gray testbed.



(a) Each bar represents the mean values of a component, the error bars represent the respective standard deviation.

(b) Gray mean values as RGB color

In Figures 7.1, 7.2, 7.3, 7.4, 7.5, the statistics from Table 7.1 are presented. The data from each testbed color configuration is shown in it's own bar chart, where the bars represent the mean values of the respective color's components including error bars which shows the standard deviations. In addition, a colored circle is presented, where the color is obtained by converting the mean values into RGB colors[1].

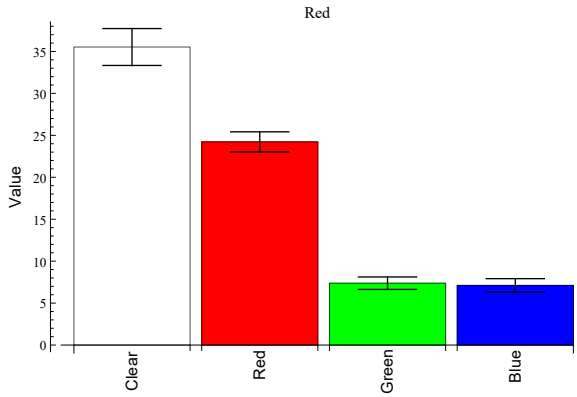
7.1.2 Testing color classification

Both techniques mentioned in Section 6.1.1.2 were applied at to find the most suitable one. The filtering technique resulted in very unstable color readings, returning e.g. `Yellow` when the concrete sleeper was green. The color distance calculation gave more promising results, and a test run was carried out on a circular track. The track consists of 16 curved Lego rail bricks with the following colored sleepers attached, on top of some of the 64 gray sleepers:

- Red: One sleeper, also marking the end of a completed round.
- Blue: Three sleepers.
- Green: Two sleepers.
- Yellow: Two sleepers.

[1]The RGB color code is obtained by multiplying each of the color components [red, green, blue] with 255, and dividing by the 'clear' value. The resulting numbers are rounded to the closest integer and converted to a hexadecimal value. The hex numbers are concatenated in the order given by the acronym: Red; Green; Blue, resulting in an RGB hex value.

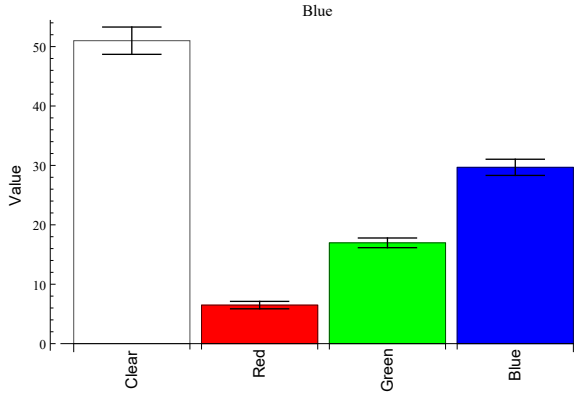
Figure 7.2: Result set from color sampling on red testbed.



(a) Each bar represents the mean values of a component, the error bars represent the respective standard deviation.

(b) Red mean values as RGB color

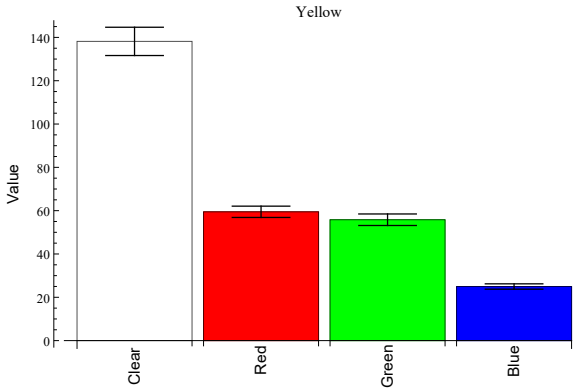
Figure 7.3: Result set from color sampling on blue testbed.



(a) Each bar represents the mean values of a component, the error bars represent the respective standard deviation.

(b) Blue mean values as RGB color

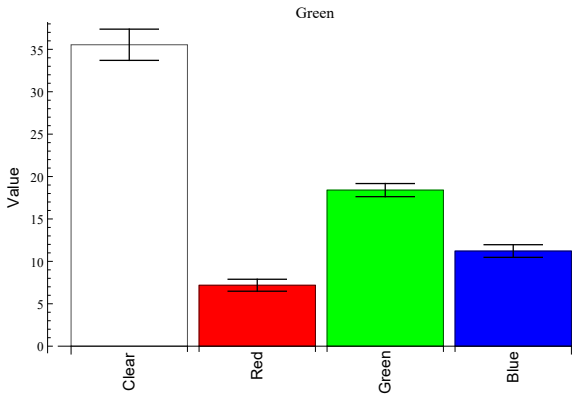
Figure 7.4: Result set from color sampling on yellow testbed.



(a) Each bar represents the mean values of a component, the error bars represent the respective standard deviation.

(b) Yellow mean values as RGB color

Figure 7.5: Result set from color sampling on green testbed.



(a) Each bar represents the mean values of a component, the error bars represent the respective standard deviation.

(b) Green mean values as RGB color

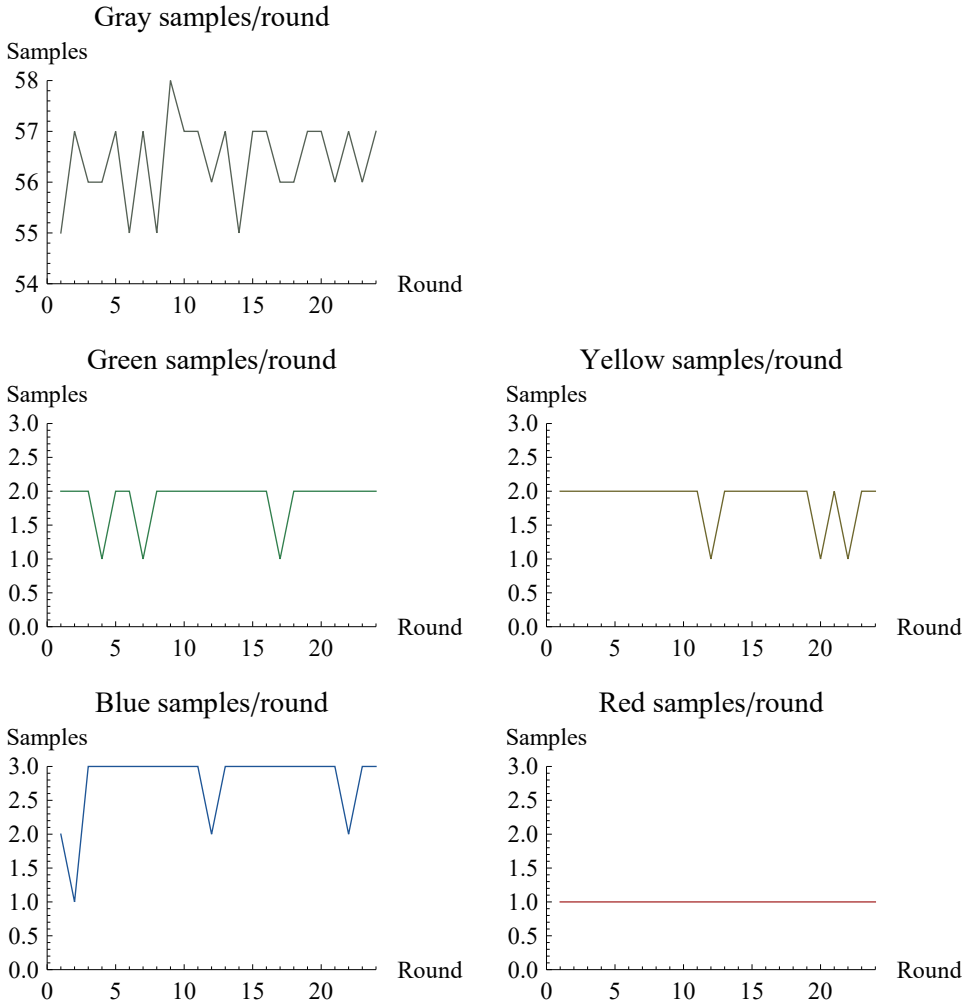


Figure 7.6: Result plots from color sensor test run on a circular track

The results of the test run on the circular track are shown in the plots in Figure 7.6. The plots show how many sleepers the color sensor/publisher pair detected in each round on the circular track. Each color is represented in it a separate plot. As it appears from the plots, only the `Red` sleepers are detected without any deviations during the test run. All in all, 63 sleepers are recognized each round in average, though this mean value is somewhat drawn up by the fact that gray sleepers are over-reported in some rounds.

Table 7.2: Statistical data for the color sensor sampling rate

Property	**Value** [ms]
Mean	15.999
Variance	0.261
Standard deviation	0.511
99.9 percentile	21.003

7.1.3 Testing color sensor timing

During the test run mentioned above, the time delta between sensor samples was logged to document the relative performance of the application. The results of the timing test can be seen in the plot and survival count histogram of Figure 7.7. The plot shows the time delta between color samples over time, while the histogram shows the survival count, i.e. how large quantities of the samples are less than a given value on the x-axis. As can be observed in the histogram, less than ten samples are delayed more than $5ms$ after the planned $16ms$ periodic delay, and the worst case is $32.5ms$. The statistical properties of the sample set are seen in Table 7.2. The phenomenon in the plot where a delayed sample is followed by an equivalent reduction in delay of the next sample is caused by the implementation of the `SchedulerService`.

7.2 MIFARE tag readings

As with the color sensor/publisher pair, a circular test track is used to test the NFC-reader/publisher pair. Three Mifare-capable tags are encoded with different ID's using the `pn532` bundle, and placed with equal distances from each other under the test track. The *Location IDs* used are (99999995;99999996;99999997). The train is set to run for 26 rounds, resulting in 77 read *Location IDs*. As the observant reader may notice, 77 readings is one reading short when there are three readings per round for 26 rounds. The readings are plotted in Figure 7.8. The values on the y-axis represent different Location IDs. As evident in the plot, one tag reading out of the total of 78 possible readings were omitted.

7.3 Acceleration based metrics

As described in the previous chapter, the accelerometer can potentially function as a basis to calculate changes in velocity and displacement in a real time manner. To test the accuracy and identify the potential of the accelerometer, several tests are conducted. The test procedures and results are described in the following sections.

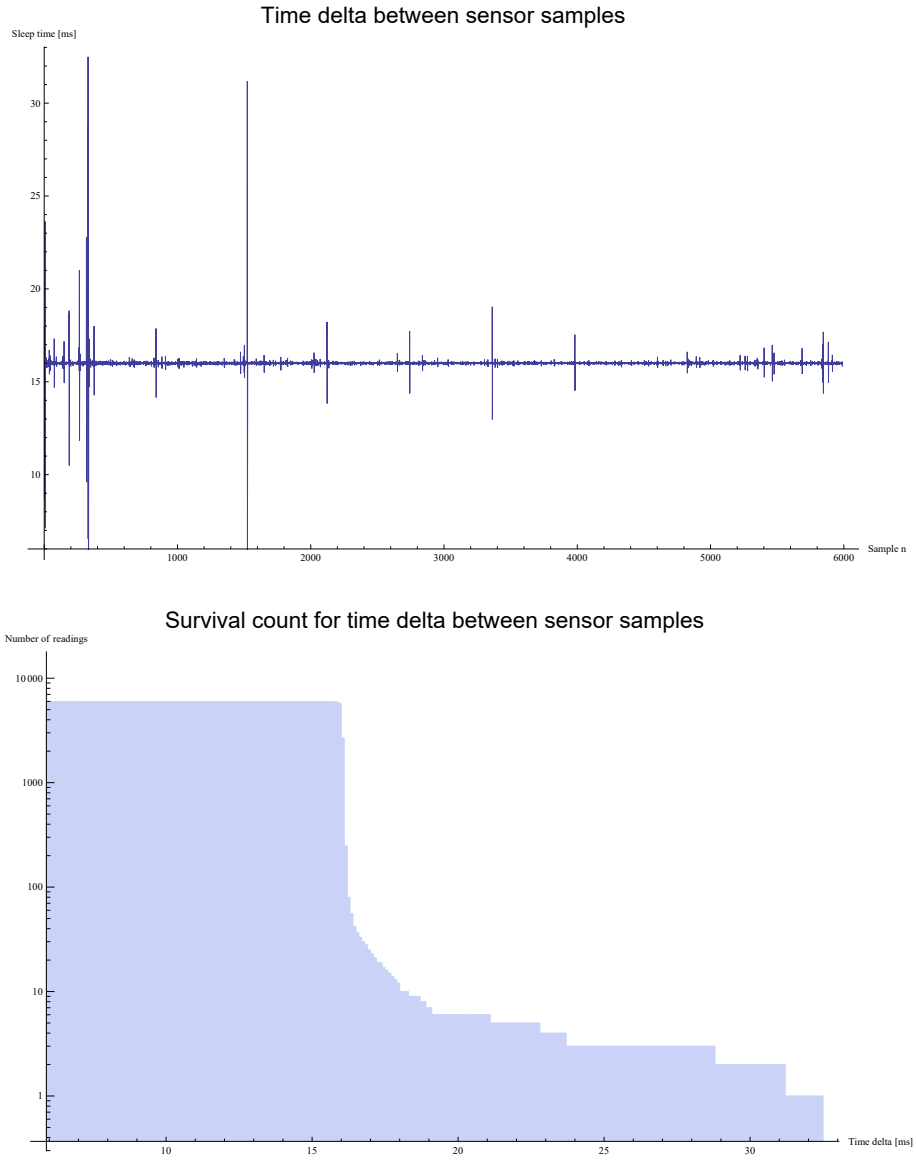


Figure 7.7: Timing measurements results of the color sensor/publisher pair

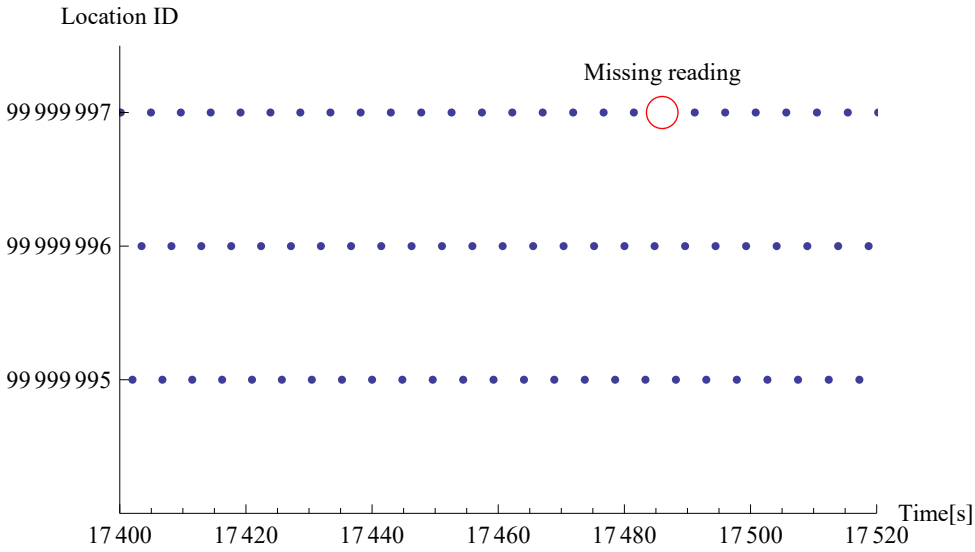


Figure 7.8: MIFARE tag readings during test run on circular track

7.3.1 Initial linear movement mapping

For this test a 40*cm* long, straight track is used. The train is set to movement from a stationary position and stopped after approximately one second. After a pause of one second, the movement is reversed, bringing the train back to the initial position one second later. This procedure is repeated once. The result from this test is shown in Figure 7.9. The green circles indicate the time frame where the train is accelerated forward and then decelerated to a stationary position. The red circles indicate the time frame where the opposite (reverse) movement occurs. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

As the result presented in Figure 7.9 shows, the acceleration measurements can be interpreted to describe the movements of the train. The calculated velocity does however never return to zero, which in reality is the case during the pause after all four movements when the train is stationary. During the reverse movements marked with red circles in the figure, the velocity should be negative as the train accelerates in a negative x-axis direction.

7.3.2 Accelerometer simulator

To identify the source of the problem detected in the previous section, a simulator is developed. The physical accelerometer is replaced by a simulated accelerometer simply by substituting the `MMA8491` bundle with the simulator bundle, having the simulator bundle registering the same service as the initial sensor bundle, the

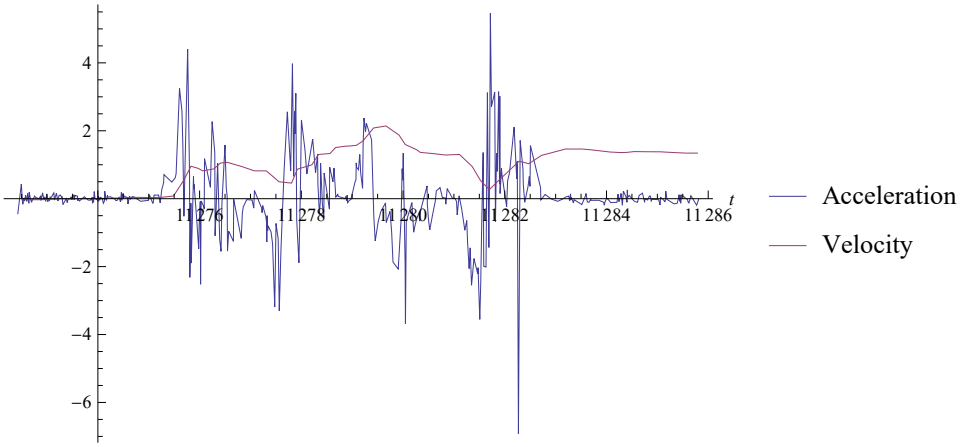


Figure 7.9: Acceleration measurements along with calculated velocity delta

AccelerometerControllerService. For each time the *AccelerationControllerMocker*'s `getRawData()` method is called, a new integer array is returned. All three axis are simulated similarly, but it is the x-axis that is in focus. The values returned by the simulated service forms a triangular pulse signal which means the simulator imitates the initial, forward directed movement made in the previous experiment. By inputting the triangular pulse signal to the calculation logic in the *VelocityPublisher*, verification of the mathematical functions can be done. Note that no changes are made to the VelocityPublisher, i.e. the sampling rate is unchanged.

The output from the *VelocityPublisher* is shown along with the input to the algorithm, i.e. the simulator output, in Figure 7.10. Note that there is no change in velocity as long as the acceleration is zero. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

Analyzing the resulting velocity delta, the correctness of the speed calculation is verified. The velocity rises to $1^{m}/_{s}$ before it stabilizes due to a positive triangular peak acceleration of $1^{m}/_{s^2}$ with two seconds duration, and then decreases back to $0^{m}/_{s}$ and stabilizes due to a negative triangular peak of the same size.

7.3.3 Noise damping

As the erroneous velocity delta is confirmed not to originate in the velocity calculation algorithm, the deviations are suspected to be a result of sensor noise and inaccuracy. As inaccuracy is a static factor that follows the sensor hardware, sample noise will be the center of attention in the following attempts. Below are the attempts to cancel the instabilities seen in Figure 7.9 described in detail.

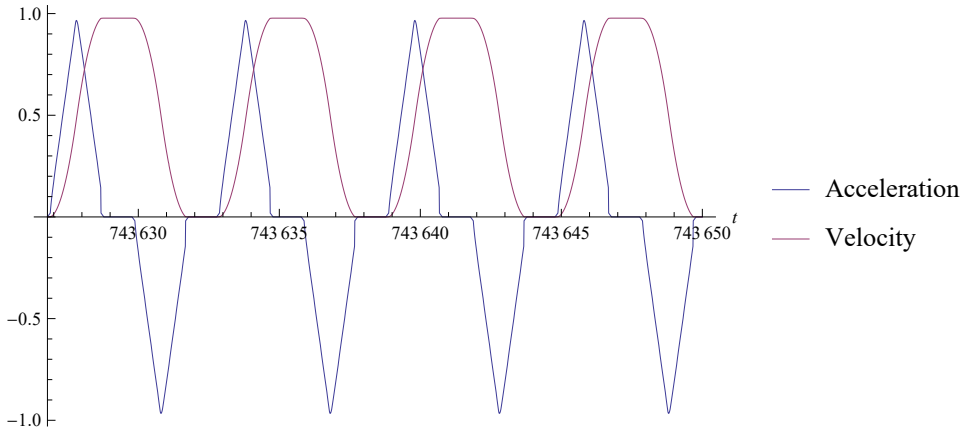


Figure 7.10: Simulated acceleration measurement along with calculated velocity delta

7.3.3.1 Low-pass filtering

To stabilize the output of the AccelerationPublisher, thus avoiding publishing possibly erroneous events, the low-pass filter in (7.1) is implemented in the AccelerationPublisher. The coefficient α determines to which extent the output should be smoothened; an $\alpha = 0$ entails no smoothing while an $\alpha = 1$ results in a static output[Nic11].

$$output_n = a_n + \alpha_n(a_{n-1} - a_n) \tag{7.1}$$

7.3.3.2 Static low-pass filtering

An experiment is carried out using a fixed value of $\alpha = 0.3$ while moving the train forward from a stationary position for approximately six seconds before bringing it to a halt again, in which the resulting outputted acceleration measurements and calculated velocity delta is presented in Figure 7.11. The green and the red circle shows a drift in calculated velocity when the train is stationary, caused by a bias of the measured acceleration. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

As the results indicate, the calculated velocity drifts in the highlighted circles, while the physical train is stationary. This is caused by an apparent bias of the measured acceleration in the same time periods, which in Figure 7.9 was more stable. The intermediate period does, however, show a rise and consecutive fall in calculated velocity, which resembles the actual movement of the train during the experiment.

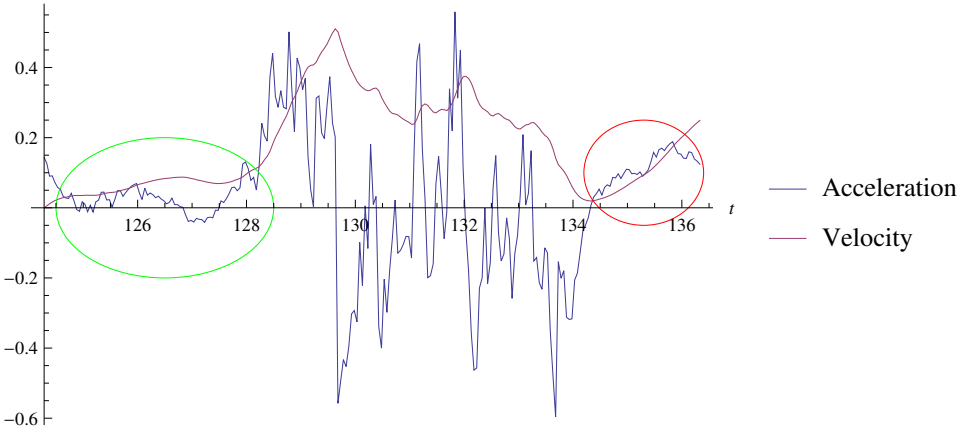


Figure 7.11: Acceleration measurements and calculated velocity with static low-pass filtering, $\alpha = 0.3$.

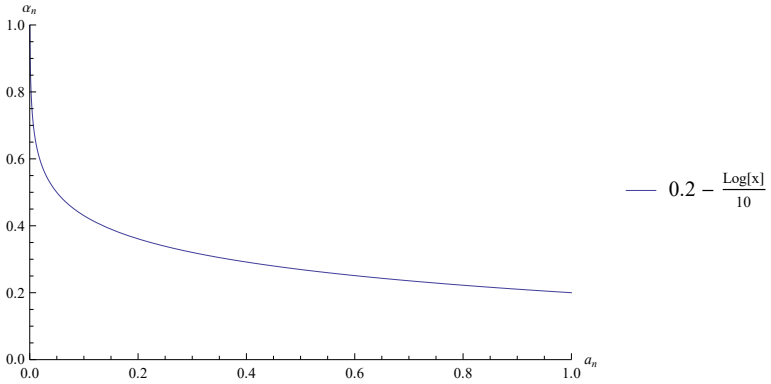


Figure 7.12: Plot of alpha function of (7.2) for the range $< 0, 1 >$

7.3.3.3 Dynamic low-pass filtering

Further pursuing the smoothing approach to achieve reliable acceleration measurements, the fixed alpha value is substituted with the function in (7.2). The variable range of the function is plotted in Figure 7.12.

$$\alpha_n = \begin{cases} 1 & |a_n| < 0.001 \\ 0.2 - \frac{1}{10} \log a_n & 0.001 < |a_n| < 1 \\ 0.2 & |a_n| >= 1 \end{cases} \tag{7.2}$$

The purpose of an adapted alpha-value is to stabilize the outputted acceleration

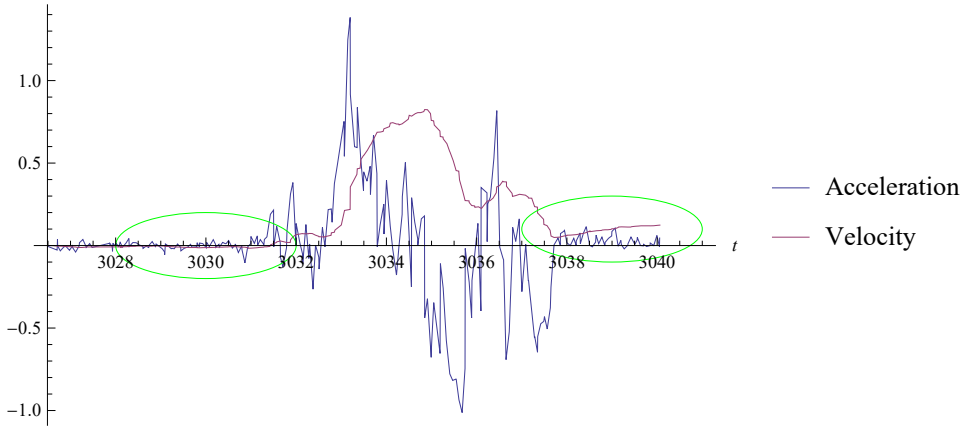


Figure 7.13: Acceleration measurements and calculated velocity with dynamic low-pass filtering

when the actual momentum acceleration is close to zero. An experiment similar to the one last described is conducted, and the results are presented in Figure 7.13. The green circles show a slight drift in calculated velocity when the train is stationary. The calculated velocity has some variations relative to the actual movement. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

As the plot in Figure 7.13 shows, the calculated velocity is in this case quite near the actual velocity in the physical experiment. Some variations are present, e.g. the minor drop in velocity while moving forward, but the calculations represent the overall movement in general.

7.3.3.4 Investigating dynamic low-pass filtering further

As the latter result seems promising, a similar experiment is carried out to determine whether or not the results are representative of the chosen approach of dynamic low-pass filtering. The results from this experiment can be seen in Figure 7.14. The green circle shows a slight drift in calculated velocity when the train is stationary. As the train is halted, the impact on the acceleration output is so small that the calculated velocity delta experiences a minor drop. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

As can be seen in Figure 7.14, the calculated velocity delta lies slightly lower initially than in the previous test run but rises towards the end. When the train is brought to a halt, the calculation of the velocity delta falls through as the outputted acceleration measurement indicate a too short negative pulse. This shows that the apparent reliability observed in the previous experiment was a more or less random

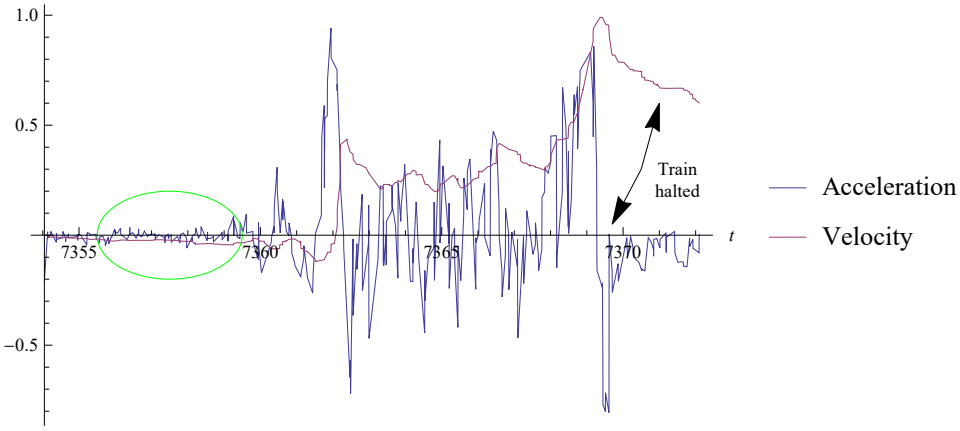


Figure 7.14: Acceleration measurements and calculated velocity with dynamic low-pass filtering, second run

outcome.

7.3.3.5 Simulating white noise

To provide a comparative basis, the simulator of Section 7.3.2 is modified to include white noise and is passed through the dynamic low-pass filter with the same characteristics as in the latter experiments. The output of this simulation can be viewed in Figure 7.15. The orange, dotted line represents a function $f(t) = 1.6 \sin \frac{1}{37} t$. The y-axis denotes both acceleration in $^{m}/_{s^2}$ and velocity in $^{m}/_{s}$.

As the results from the simulated test run with white noise generation on the simulated signal shows, the calculated velocity delta drifts, approximately following the sine function $f(t)$. This result emphasizes the observations from Figure 7.14, i.e. that the implementation of the dynamic low-pass filter is insufficient to achieve reliable velocity calculations.

7.3.4 Accelerometer issues

Due to the discovered intricacies concerning the acceleration measurements pointed out in the sections above, the approach of velocity calculation is not pursued any further. This implies that calculations of displacement delta are too out of the question in this thesis.

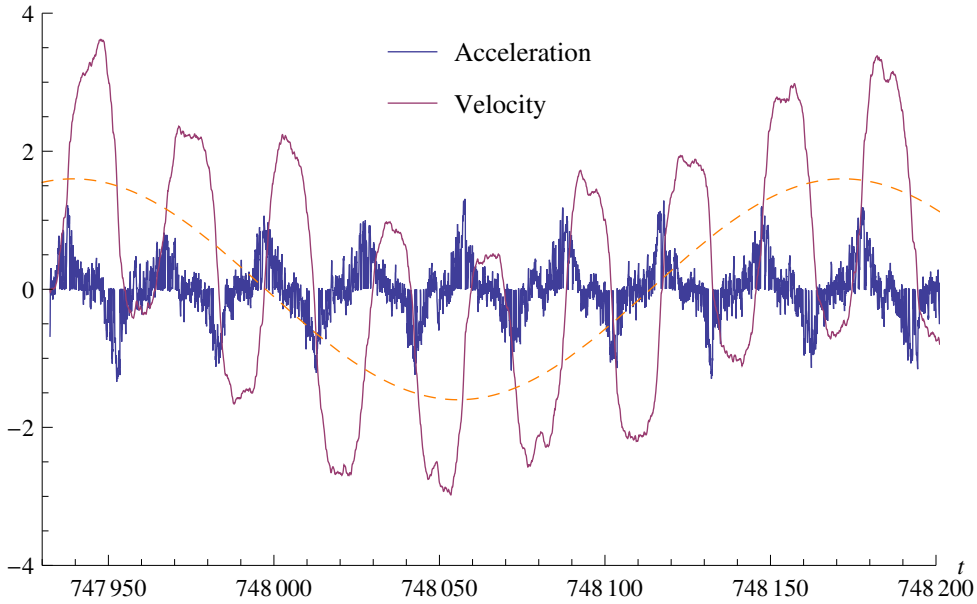


Figure 7.15: Simulated acceleration measurement passed through dynamic low-pass filter, along with calculated velocity delta

Table 7.3: Magnetometer calibration constant values

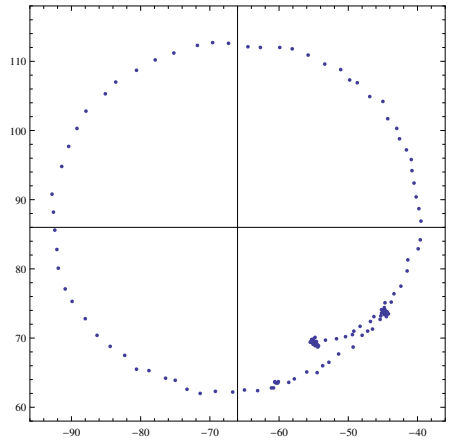
Axis	Value
x	65
y	-85

7.4 Magnetic heading measurements

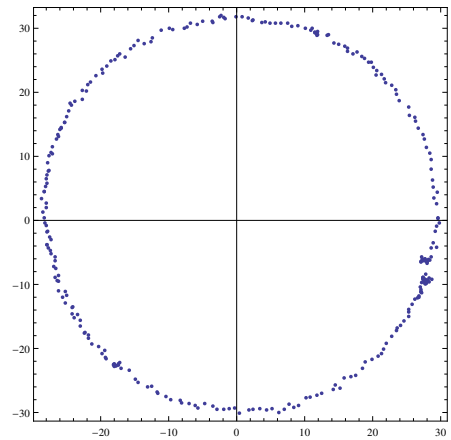
7.4.1 Calibration data and adjustments

As introduced in Section 6.1.4.1, magnetometers may need to undergo a calibration procedure before reliable operation can be initiated. The outputs of the two-step procedure is presented in Figure 7.16a,7.16b. In Figure 7.16a the sampled values forms an approximate circle far away from the plot origin point $(0, 0)$, whilst in Figure 7.16b the origin point is the approximate center of the circle formed by the sampled values. The calibration constant values required to achieve the result in Figure 7.16b for the Mag3110 chip on-board the prototype train is shown in Table 7.3.

Figure 7.16: Magnetometer calibration plots. Note the values on the x/y-axes



(a) Magnetometer calibration output prior to adjustments



(b) Adjusted magnetometer calibration output

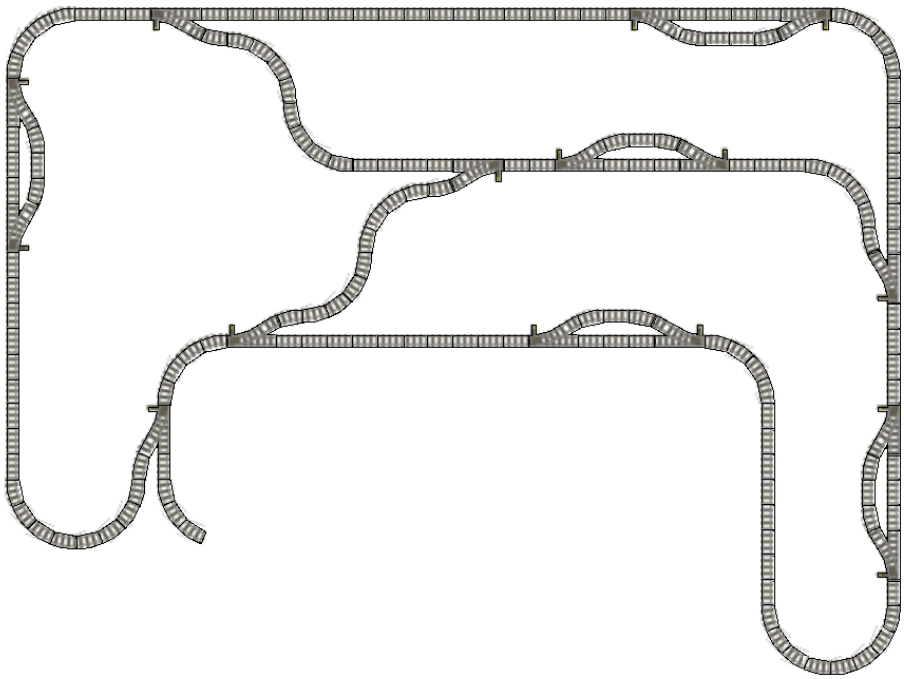


Figure 7.17: Railroad map from previous project, image from [Sve15c]

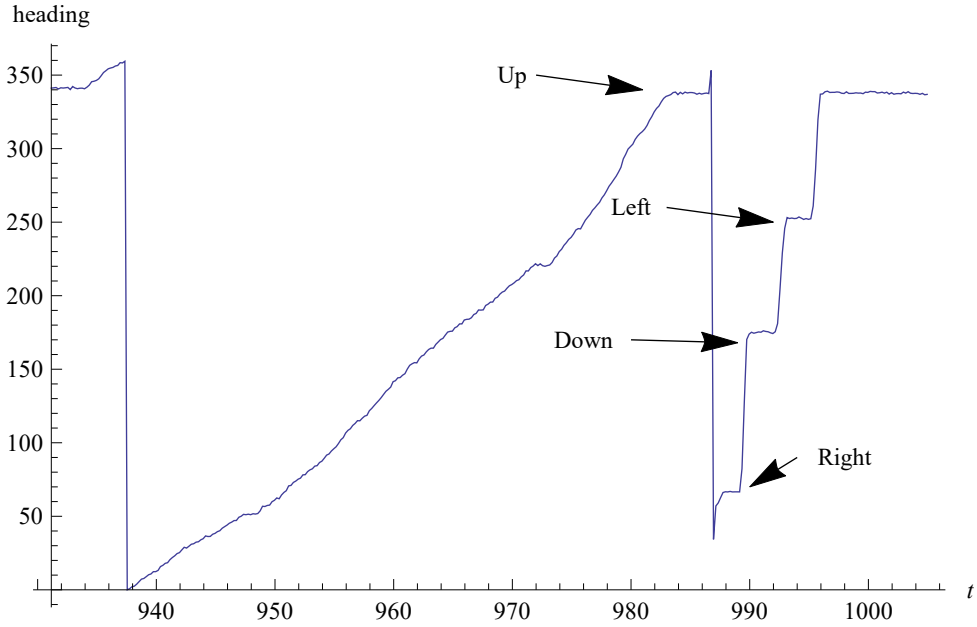


Figure 7.18: Mapping directions to heading values

7.4.2 Magnetic heading of the track layout

The Lego track layout used in [Sve15c], depicted in Figure 7.17, has four general directions if curved track pieces are not accounted for. These directions are described by 'Up'; 'Right'; 'Down'; and 'Left'. To map values for these four directions and to verify proper functionality of the magnetometer sensor/publisher pair, an experiment is carried out where the train is first rotated slowly 360° clockwise, starting at an 'Up'-bound position. After the rotation is completed, the train is rotated 90° clockwise for four iterations, holding the position for approximately three seconds in each post to get stable readings in each position ('Up';'Right';'Down';'Left').

As the results in Figure 7.18 shows, stable output values are present at all four direction headings. The magnetic heading level of each direction is indicated in the plot. The vertical shifts at approx. $t = 938$ and $t = 987$ is a result of the train rotating past the magnetic north, which lies at $heading = 0$. The slow, complete rotation results in a smooth, rising curve in the plot.

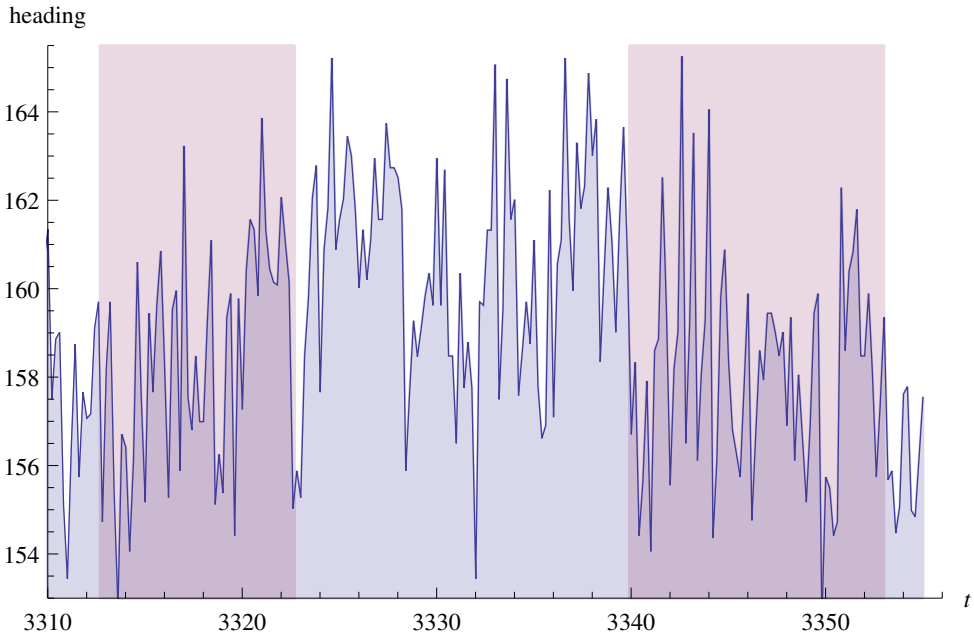


Figure 7.19: Motor interference from the on-board motor unit

7.4.3 Magnetic interference

7.4.3.1 From the motor

An experiment is carried out to investigate whether or not the electromotor onboard the train introduces any magnetic interference to the magnetic heading calculation. The train is placed on top of blocks to remain stationary throughout the experiment. The motor unit is turned on and off twice, running in about 10 seconds each time with an intermediate pause of approximately 18 seconds. The result from this experiment is presented in Figure 7.19. The red, shaded areas indicate that the motor unit is running, while the other regions show that it is turned off.

As can be seen in Figure 7.19, the magnetic noise the sensor is subject to does not change dramatically while the motor is switched on. If anything, the overall calculated heading seems to lie a bit lower during engine operation contrary to when the motor is turned off, observing the area under the curve in Figure 7.19, but this may be a coincidence.

7.4.3.2 From the environment

In everyday life, the environment one is surrounded by is packed with magnetic fields related to electric current, which became evident during a test run on the circular

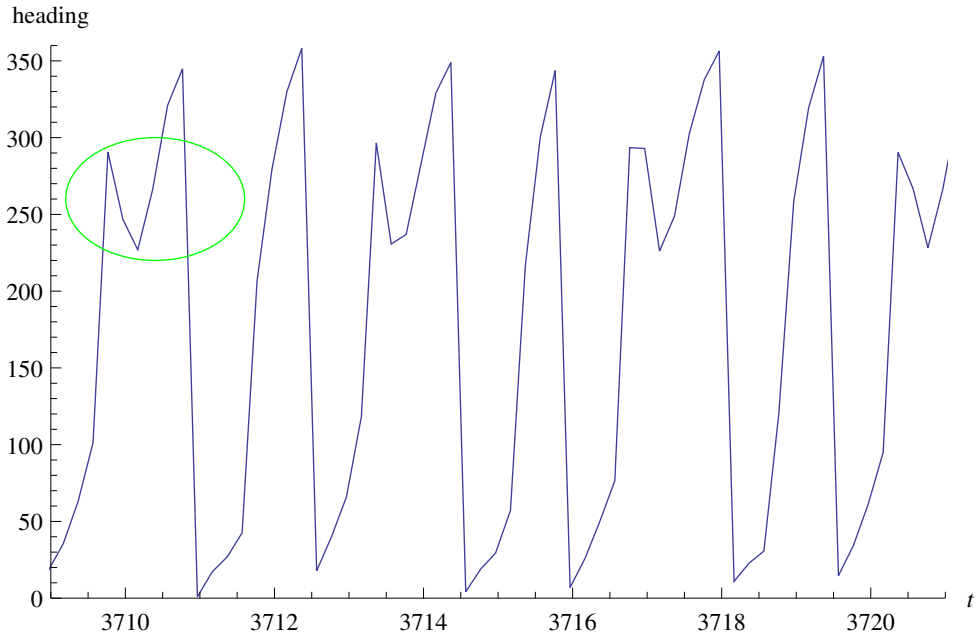


Figure 7.20: Magnetic heading calculation during circular test track run, first position

track mentioned in Section 7.1.2. As can be seen in Figure 7.20, interference is visibly recognizable in the data set. The green circle indicates the point where magnetic interference causes a negative spike in the calculated heading.

The event repeatedly occurs for each round the train travels around the circular track. More particular, the event occurs whenever the train has an approximately westbound heading. A closer look at the surroundings of the circular track reveals a cable trunking running a few centimeters away from where the train passes in a westbound position, in the exact height as the magnetometer sensor is mounted. The experiment is repeated after re-positioning the circular track so that the train passes with approximately 25 centimeters clearance to the cable trunking while headed westbound. The results of this second test run can be seen in Figure 7.21. The green circle indicates where magnetic interference by power cables are smaller than before, but can still be identified by visual inspection of the data plot.

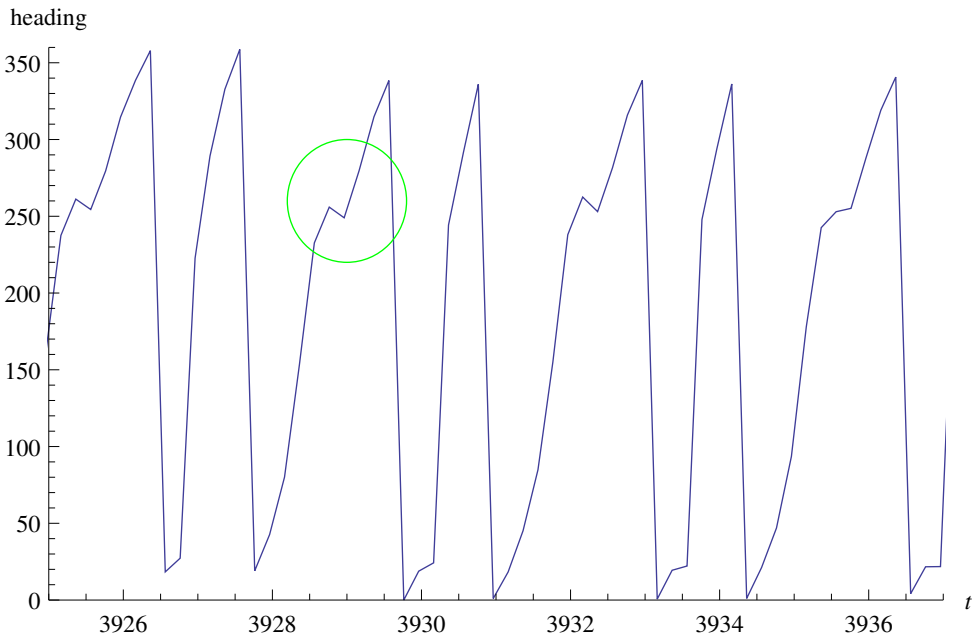


Figure 7.21: Magnetic heading calculation during circular test track run, second position

Chapter 8
Discussion

In this chapter, the results presented in Chapter 7 is discussed. Each sensor is dedicated its own section in Section 8.1, and a conclusion with regard to the problem description is drawn in Section 8.2. In the end, Section 8.3 presents proposals for further work related to the system.

8.1 Sensor strategy feasibility

8.1.1 Colored sleepers and the color light sensor

The pre-collection of color samples forms a basis for categorizing future samples made by the color light sensor. As can be observed in Table 7.1, most of the sample sets has a relatively low standard deviation, i.e. the variation in the measurements from the test bed is low. The color with the highest variation is yellow, which is also the color which has the least resemblance to the actual color when the mean values are converted to an RGB color code, as depicted in Figure 7.4b.

As seen in the previous chapter, an average of 63 sleepers is detected for each round on the circular track, i.e. one less than the actual number. Also, the number of gray sleepers are frequently over-reported, so that the actual number of missing sleepers are greater than 1/64. Comparing this result with the time delta between color samples in Figure 7.7, the absent sleeper detections may be partially explained by reading delay.

Compared to the results in [Sve15c, 8], the ten worst samples were delayed with at least $100ms$, with a worst case at almost $700ms$, while the numbers, in this situation, are in contrast $5ms$ and $32.5ms$. This shows that the newly introduced hardware platform is more suitable than the previous, at least when only regarding the color sensor timing issues experienced in the prior project.

Experiments show that the use of colored sleepers can not be fully trusted as

a sole basis for self-localization, though it can be a useful aid to make movement approximations.

8.1.2 MIFARE balises and the NFC reader

8.1.2.1 Omitted reading

The output from the MIFARE related modules in the proposed system constitute the most reliable position data, as the information embedded in the MIFARE memory reports a unique location identity to the passing train. The implementation of the sensor/publisher pair is though not entirely reliable, as we observed a single absent tag reading during the 26 test rounds. As the empirical data set is quite small in this context, it's hard to say whether the failure is of a repetitive nature or not, or if the failure could occur even more frequently in long-lasting operation.

What is certain, though, is that there is a possibility that a balise might be neglected while the train passes it. There are two probable causes of this malfunction: the reading is delayed as in the case of the color sensor; the timing of the NFC reader is too infrequent. Either way, this problem could most likely be solved by implementing handling of the interrupt signal from the pn532 chip. An attempt was made to implement the interrupt processing, but the time frame for the thesis did not allow for further efforts. The implementation was impeded by timing issues as the interrupt signal is affected not only by having tags entering the readers proximity but also when sending commands to it through the I^2C interface.

8.1.2.2 MIFARE classic attack

The security of the MIFARE classic cards are based on symmetric-key cryptography, and a non-public security scheme that goes under the term 'security by obscurity'. History has shown that such an approach to implementing security functions will be broken sooner or later.

In 2008, the security features of the MIFARE classic cards were reverse engineered [GdKGM+08] and a practical attack on it was published [dKGHG08], making the onboard memory susceptible to e.g. manipulation by unauthorized parties. Choosing this technology to enhance the self-localization function of trains opens up for potential unwanted, undesirable modification, thus introducing vulnerability towards sabotage. However, this being only a model system, restricted physical access to the system in addition to the small consequences in the case of sabotage compensates for the security issue.

8.1.3 Accelerometer measurements and derived calculation

As seen in the previous chapter, the accelerometer delivers quite unstable measurements. Although suitable to measure horizontal tilt, the experiments reveal that the sensor's erroneous behavior propagates, and is amplified through calculations, which results in unpredictable velocity readings.

Initiatives to mitigate the mentioned problems are taken, but with mixed results. Some experiments have more promising outputs than others, and the experiment which output presented in Figure 7.13 is by far the most successful one. The result is though not representative for the technique, it shows that it might be *possible* to achieve satisfactory results using the chosen approach, requiring a deeper investigation than provided in this thesis.

As the velocity calculations from the simulated data experiment in Section 7.3.3.5 seems to drift at a rate similar to a sine function, the possibility of exploiting this phenomenon in real input operations has been assessed. The systematic fluctuation is likely to be caused by the simulated signal's triangular waveform, and the pattern has probably no relation to the actual calculations or dampening implementations.

Based on the demonstrations presented in this thesis, the accelerometer is assumed to be a far too unreliable source to calculate velocity and displacement metrics from, even if these parameters are used only for approximations.

8.1.4 Magnetic heading utilization

The magnetometer provides magnetic heading information to the train and is in the initial experiment quite precise. Relatively steady levels are outputted when the direction is fixed, and the resulting plot in Figure 7.18 shows smooth, rising curves as the magnetometer is subject to clockwise rotation.

The motor in the train prototype has seems to have little impact on the calculated magnetic heading. The general level of the heading appears to have a marginal decrease when the motor is on versus off if looking at the area under the curve in Figure 7.19. The increase is so small that it could as well be a result of the random variation in the outputted value from the magnetometer.

A source of magnetic interference that *has* an impact on the heading levels is close proximity power cables. It is conceivable that cables with larger cross sections and with a higher current load would have an even greater influence as the magnetic field surrounding it would be more powerful. Other appliances, such as a transformer, would probably also be a problem.

The results from the previous chapter show that the magnetometer may, in fact,

be used as a basis for deriving representative magnetic heading metrics, which in turn could be utilized in relation to the abstract model of the railroad. Full utilization depends on an interference-free environment for the Lego railway, which is considered unlikely. The information could be used to make assumptions regarding whether or not the train is moving straight forward or if it's turning.

8.2 Conclusion

Based on the results previously presented, it is evident that even though the Raspberry Pi (RPi) is a relatively cheap, lightweight computer, the introduced hardware platform performs better than the one used in previous projects. As the RPi has more generic interfaces than the Lego Mindstorms EV3 bricks used earlier, connectivity versatility is highly increased.

As discussed in the previous sections, the variety of sensors leads to multiple challenges and experienced issues. The NFC reader combined with MIFARE balises encoded with location information stands out as the most precise and reliable alternative but provides little movement approximations in between balises. The magnetometer and the color sensor is applicable for such approximations, while the accelerometer's lack of accuracy makes it unsuitable for even making coarse estimates in this system. The Mifare-based approach has as mentioned earlier resemblance with the Eurobalises in the real-world system ERTMS. Even though the implementation of the NFC reader module is of experimental nature, the functioning strategy opens up for encoding other kinds of control data into the balises. Thus, the model system obtains similar characteristics as the ERTMS system, making it applicable to a prototyping and testing platform for e.g. future expansions of the ETCS. This is mentioned as an overall motivation in Chapter 1.

During sensor measurement experiments the framework and architecture in Chapter 5 has implicitly been put to a test. Although quantifying non-functional aspects of a software architecture is hard, the seamlessness in developing and deploying sensor simulators to replace the original sensor implementations shows that the system's architecture really offers modifiability. This has also been the experience when sharing modules across projects in the collaboration with co-student Svae.

The modules presented in this thesis acts as a sound basis for self-localization in an autonomous train system.

8.3 Further work

All of the tested sensors should be investigated further to improve the data processing succeeding each sensor sample. In addition to the individual elements described below, there are common factors of the sensors that should be taken into closer account. More particular, the dependability of the sensor related implementations in interaction with the hardware should be analyzed and, if feasible, quantified. Concurrent execution of the different modules should be studied more carefully. The fusion of sensor data, which is solved in a quite simplistic manner here as it is not the main scope of the thesis, could also be of interest to increase data utilization.

The NFC interrupt signal could provide additional stability improvements of the MIFARE tag readings. The schematic description in Section 4.2.2 shows how the physical connection should be rendered, the challenge is mere to analyze the timing issues more thoroughly to compose a functioning implementation.

Acceleration measurement processing should be subject to deeper analysis before it can be turned to account. Different approaches can be used, e.g. experimenting with other kinds of accelerometer chips or using different techniques for processing the accelerometer data. A different, more sophisticated approach is to use machine-learning to 'teach' the application categories of movements so that it can classify an experienced movement *a posteriori*. As the accelerometer measurement plots can be studied, deducing the movements causing the acceleration 'by hand', it should be possible to achieve by using machine-learning as well.

The magnetic interference in the railroad's surrounding environment can be mapped by carrying through experimental runs where the train stops at frequent, known positions[1] to log the measured heading relative to the known position in the abstract model. By learning the interference pattern along the track, the pattern could be used to recognize areas and positions.

[1] Positions can be known through utilizing e.g. MIFARE readings

References

[Ada] Adafruit. Adafruit dc stepper motor hat for raspberry pi - mini kit. Adafruit, https://www.adafruit.com/products/2348, (accessed May 10, 2016).

[Ada14] James Adams. Introducing raspberry pi hats. RaspberryPi.org, https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/, (accessed May 10, 2016), July 2014.

[aos16] Texas advanced optoelectronic solutions. Tcs34725 color light-to-digital converter with ir-filter. Adafruit.com, https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf, (accessed May 13, 2016), 2016.

[BCK03] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Pearson Education, 2003.

[Blo06] Robin Bloomfield. Fundamentals of european rail traffic management system (ertms). In *Railway Signalling and Control Systems, 2006. The 11th IET Professional Development Course on*, pages 165–184. IET, 2006.

[COO11] Vedat Coskun, Kerem Ok, and Busra Ozdenizci. *Near Field Communication (NFC): From Theory to Practice*. John Wiley & Sons, 2011.

[dKGHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D Garcia. *A practical attack on the MIFARE Classic*. Springer, 2008.

[EFGK03] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[ERT14] ERTMS. Ertms deployment statistics - overview. ERTMS.net, http://www.ertms.net/?page_id=58, (accessed April 21, 2016), 2014.

[GdKGM+08] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling mifare classic. In *Computer Security-ESORICS 2008*, pages 97–114. Springer, 2008.

[Goo13] Steven Goodwin. *Smart home automation with Linux and Raspberry Pi*. Apress, 2013.

[HBHS15] Fenglin Han, Jan Olaf Blech, Peter Herrmann, and Heinz Schmidt. *Model-based Engineering and Analysis of Space-aware Systems Communicating via IEEE 802.11*, volume 2. 2015.

[HBHS16] Peter Herrmann, Jan Olaf Blech, Fenglin Han, and Heinz Schmidt. A model-based toolchain to verify spatial behavior of cyber-physical systems. pages 40–52, 2016.

[HØ15] Simon Hordvik and Kristoffer Øseth. Control software for an autonomous cyber-physical train system. Master's thesis, Norwegian University of Science and Technology, Department of Telematics, June 2015.

[Hon]

[Hor08] Tim Hornyak. Rfid powder. *Scientific American*, 298(2):68–71, 2008.

[HPMS11] Richard Hall, Karl Pauls, Stuart McCulloch, and David Savage. *OSGi in action: Creating modular applications in Java*. Manning Publications Co., 2011.

[ISO16] ISO/IEC). 14443-1/4:2016 identification cards – contactless integrated circuit cards – proximity cards. ISO.org, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=70170, (accessed May 10, 2016), 2016.

[KH04] Peter Kriens and B Hargrave. Listeners considered harmful: The "whiteboard" pattern. *Technical whitepaper, OSGi Alliance*, 2004.

[Kra08] Frank Kraemer. Engineering reactive systems, July 2008.

[Lan05] Jeremy Landt. The history of rfid. *Potentials, IEEE*, 24(4):8–11, 2005.

[Mar03] Robert Cecil Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[MN15] Alexander McKenna and Alban Nanty. "bluebrick documentation". Bluebrick.lswproject.com, http://bluebrick.lswproject.com/help_en.html, 2015.

[Nic11] Tom Nichols. Smoothing sensor data with a low-pass filter. ThomNicols.org, http://blog.thomnichols.org/2011/08/smoothing-sensor-data-with-a-low-pass-filter, (accessed May 13, 2016), August 2011.

[OAS15a] OASIS. "AMQP is the internet protocol for business messaging". Amqp.org, https://www.amqp.org/about/what, (accessed November 5), 2015.

[OAS15b] OASIS. "RabbitMQ - what can RabbitMQ do for you?". RabbitMQ.org, https://www.rabbitmq.com/features.html, (accessed November 5), 2015.

[Ove15] Kristian Overskeid. ITS using Lego Mindstorm. Master's thesis, Norwegian University of Science and Technology, Department of Telematics, March 2015.

[Ozy15] Talat Ozyagcilar. Implementing a tilt-compensated ecompass using accelerometer and magnetometer sensors. November 2015.

[Rpi16] "raspberry pi faqs - frequently asked questions". RaspberryPi.org, https://www.raspberrypi.org/help/faqs/, (accessed May 13, 2016), 2016.

[Sema] NXP Semiconductors. Pn532/c1 near field communication (nfc) controller data sheet. NXP.com, http://cache.nxp.com/documents/short_data_sheet/PN532_C1_SDS.pdf, (accessed May 13, 2016).

[Semb] NXP Semiconductors. Xtrinsic mag3110 three-axis, digital magnetometer. NXP.com, https://www.nxp.com/files/sensors/doc/data_sheet/MAG3110.pdf, (accessed May 13, 2016).

[Semc] NXP Semiconductors. Xtrinsic mma8491q 3-axis multifunction digital accelerometer. NXP.com, http://cache.nxp.com/files/sensors/doc/data_sheet/MMA8491Q.pdf, (accessed May 13, 2016).

[Semd] NXP Semiconductors. Xtrinsic-sense board. Element14.com, https://www.element14.com/community/servlet/JiveServlet/downloadBody/65488-102-1-287915/XTRINSIC-SENSE%20User%20Manual%20v0%205.pdf, (accessed May 13, 2016).

[Sem00] Philips Semiconductors. The i2c-bus specification. *Philips Semiconductors*, 9397(750):00954, 2000.

[Sem10] NXP Semiconductors. Pn532 c106 application note. Adafruit.com, https://cdn-shop.adafruit.com/datasheets/PN532C106_Application+Note_v1.2.pdf, (accessed May 13, 2016), January 2010.

[SKR16] SKROSS. "skross reload 5 datasheet". Farnell.com, http://www.farnell.com/datasheets/1935853.pdf, (accessed May 13, 2016), 2016.

[Sve15a] Henrik Heggelund Svendsen. "henrihs/bluebrick-id-fork - a fork of alban nanty's bluebrick". Github.com, https://github.com/henrihs/BlueBrick-id-fork, (accessed December 18, 2015), 2015.

[Sve15b] Henrik Heggelund Svendsen. "henrihs/bluebrick4j - a library for reading bluebrick files in java". Github.com, https://github.com/henrihs/BlueBrick4J, (accessed December 18, 2015), 2015.

[Sve15c] Henrik Heggelund Svendsen. Model-based engineering of a distributed, autonomous control system for interacting trains, deployed on a lego mindstorms platform. Master's thesis, Norwegian University of Science and Technology, Department of Telematics, December 2015.

[Vio05] Bob Violino. The basics of RFID technology. RFIDJournal.com, http://www.rfidjournal.com/articles/view?1337, (accessed May 10), 2005.

[Zam11] Tim Zaman. Easy magnetometer calibration. TimZaman.nl, http://www.timzaman.nl/?p=994&lang=en, (accessed May 13, 2016), April 2011.

Appendix A
Source code

A.1 Git repository

The complete source code for the modules developed in the context of this thesis is comprehensive, counting over 5.000 lines of code if the parts re-used from preceding projects are not accounted for. The source code for each of the modules is co-located in a single git repository at https://github.com/henrihs/osgi-train. Other repositories related to in the thesis is [Sve15a] and [Sve15b].

A.2 Velocity delta

```
public class VelocityData<T extends UnivariateRealIntegratorImpl> implements
    Comparable<VelocityData<?>> {

       private long timestamp;
       private final double a_x;
       double v_delta = 0;
       private T integrator;
       // The integrator used in the implementation using this class is
       // the TrapeziodIntegrator, but other integrators like
       // the SimpsonIntegrator are also supported

       public VelocityData(double a_x, long timestamp, T integrator) {
              this.timestamp = timestamp/1000; // microsecond accuracy
              this.a_x = a_x;
              this.integrator = integrator;
       }

       public long getTimestamp() {
              return timestamp;
       }
```

```
public synchronized void calculateVelocityDelta(VelocityData<?>
    priorEvent) {
       try {
              double a_0 = priorEvent.a_x;
              double a_1 = this.a_x;

              // If a_0 and a_1 are polar opposites,
              // the change in speed is always equal to zero
              // In addition, integration over such an interval
                  fails,
              // iff a_0 is negative and a_1 = -(a_0)
              if (a_1 == -(a_0)) {
                     v_delta = 0;
                     return;
              }

              double t_0 = priorEvent.getTimestamp()*1E-6; // second
                  resolution (SU-unit)
              double t_1 = getTimestamp()*1E-6; // second resolution
                  (SU-unit)

              AccelFunction a = new AccelFunction(a_0, a_1, t_0,
                  t_1);
              v_delta = integrator.integrate(a, t_0, t_1);
       } catch (FunctionEvaluationException |
           IllegalArgumentException | ConvergenceException e) {
              ((LogService)VelocityPubActivator.logServiceTracker.getService())
              .log(LogService.LOG_ERROR, "Could not integrate
                  acceleration expression: ", e);
       } catch (Exception e) {
              ((LogService)VelocityPubActivator.logServiceTracker.getService())
              .log(LogService.LOG_ERROR, "Unknown error occured: ",
                  e);
       }

}

public String toString() {
       return String.format("Time: %d ms, Acceleration: %f m/s^2,
           delta V: %f m/s", timestamp, a_x, v_delta);
}

@Override
public int compareTo(VelocityData<?> o) {
       return Double.compare(this.timestamp, o.timestamp);
```

```
        }
}

public class AccelFunction implements UnivariateRealFunction {

        private Function<Double, Double> accelAsFuncOfTime;

        public AccelFunction(double a_0, double a_1, double t_0, double t_1)
            {
               accelAsFuncOfTime = getAccelAsFuncOfTime(a_0, a_1, t_0, t_1);
        }

        @Override
        public double value(double t) throws FunctionEvaluationException {
               return accelAsFuncOfTime.apply(t);
        }

        public Function<Double, Double> getAccelAsFuncOfTime(double a_0,
            double a_1, double t_0, double t_1) {
               return new Function<Double, Double>() {
                      @Override
                      public Double apply(Double t) {
                             return a_0 + (a_1 - a_0) * ((t - t_0) / (t_1 -
                                 t_0));
                      }
               };
        }

}
```