



Norwegian University of
Science and Technology

System for Self-Navigating Autonomous Robots

**Thor Eivind Svergja
Andersen
Mats Gjerset Rødseth**

Master of Science in Cybernetics and Robotics

Submission date: June 2016

Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Summary

The purpose of the thesis was to build an Arduino-based robot, whose intended use was to map unknown areas, as well as to develop a server application that controls several robots and uses the gathered information to form a map of the area. Additionally, the wireless communication in the existing solution was to be updated using state-of-the-art technology.

An Arduino-robot was designed and built using materials acquired from Sparkfun, Elfa Distrelec and the Cybernetic Workshops at NTNU. The robot provides the same functionality as existing robots, but due to an encoder that got broken during the final week of the thesis, the position estimates does not work.

The communication protocol was updated from Bluetooth to Bluetooth Smart, using nRF51-dongles developed by Nordic Semiconductor. The software running on the server-dongle was developed in the C programming language.

By studying the existing server application, written in MATLAB, the group got an insight into the system functionality. Most of the same functionality was implemented in a new server application using the Java programming language. The system architecture was designed using modules that took care of one area of responsibility each.

The group developed a Graphical User Interface (GUI) for the system during a design process. Using well-known principles and guidelines during the design phase led to a GUI that has the human perception in mind. The user interface was developed using the intuitive GUI-builder in NetBeans, and the goal was to inherit most of the functionality from the existing solution but provide the information in a more user-friendly way.

Sammendrag

Prosjektets gjennomføring var delt inn i tre temaer. En Arduino-basert robot skulle bygges for bruk i kartlegging av ukjente områder. Videre skulle en serverapplikasjon utvikles med mulighet for å kontrollere flere roboter i sanntid, samt å kunne bruke informasjonen robotene samlet til å danne et felles kart av dataene. I tillegg skulle den trådløse kommunikasjonen fra den eksisterende løsningen oppdateres til å bruke nyere teknologi.

En Arduino-robot ble designet og bygd ved å bruke materialer og deler fra SparkFun, Elfa Distrelec og de kybernetiske vekstedene ved NTNU. Roboten tilbyr den samme funksjonaliteten som de eksisterende robotene, men da den ene hjul-enkoderen ble ødelagt iløpet av den siste uken av prosjektet, fungerer ikke posisjonsestimering i roboten.

Kommunikasjonsstandarden ble oppdatert fra Bluetooth til Bluetooth Smart, ved å bruke nRF51-dongler utviklet av Nordic Semiconductor. Programvaren som kjører på serverdongelen ble utviklet i programmeringsspråket C.

Ved å studere den eksisterende serverapplikasjonen, skrevet i MATLAB, tilegnet gruppen seg en innsikt i systemets funksjonalitet. Mesteparten av den samme funksjonaliteten ble implementert i en ny serverapplikasjon utviklet i Java. Systemarkitekturen ble designet ved hjelp av moduler, og hver modul hadde hvert sitt unike ansvarsområde.

Gruppen foretok en omfattende designprosess for å utvikle det grafiske brukergrensesnittet til systemet. Ved å bruke velkjente prinsipper og retningslinjer under utformingen, ble brukergrensesnittet designet med menneskets oppfattelseevne i fokus. Brukergrensesnittet ble utviklet ved hjelp av en GUI-builder i utviklingsverktøyet NetBeans, og målet var å implementere all funksjonaliteten fra den eksisterende løsningen, men presentere denne på en mer brukervennlig måte.

Conclusion

During the Master Thesis, the group has developed a server application and a Graphical User Interface in Java, software for an nRF51-dongle in C, as well as built an Arduino-robot. The different parts of the project seems to be good solutions to the problem description the group made at project start.

Choosing Java as programming platform facilitated a clear structure of the code and provided functionality for dividing the system into modules. The server application is developed in terms of the guidelines in “Code Complete”, and is structured in a way that makes it easy for future projects to understand and further develop the software by adding new functionality. The code is generalised, so that it is possible to add robots with different specifications than the ones already existing, e.g. a robot with more than four infrared sensors. The server application inherits most of the functionality implemented in previous solutions, however, in our implementation the system is processing data and controlling the connected robots in real-time.

The user interface is developed during a design process where the human perception has been in focus. The process has helped the group focus on the aspects that are important when designing a graphical user interface, and the final product inherits most of the functionality provided in the previous solutions, but presents them in a more user-friendly way.

Choosing Bluetooth Smart as communication protocol enabled fast, short-range and low power consuming communication. The protocol was not known to the groups at beforehand, but by studying the standard, the groups managed to implement it into the system. The server-dongle receives and processes five messages per second from each robot connected, and delivers update and status messages to the robot dependent on the server application state. The communication range suited for the system is less than 15 meters between the server and the connected robots, this to ensure that all messages are received.

The group has built a new robot, increasing the existing collection of robots. The robot uses an Arduino as the microcomputer and it is built using the same wheels, gears, infrared sensors, tower and inertial measurement unit as the AVR-robot. Using AVR-dude enabled the Arduino-robot to use the same real-time operating system as running on the AVR-robot, without having to rewrite too much of the code. One of the wheel

encoders got damaged during the last week of the project, and therefore it cannot estimate its position at this point in time. Other than the broken encoder, the robot works as intended and is therefore successfully built and implemented.

During the project period, the group has gained useful experience regarding project management and implementation. As “System for Self-Navigating Autonomous Robots” is a part of a system consisting of three dependent projects, the key to success has been communication and cooperation. Regular conversations and meetings with both the employer and the other groups working with the system have enabled effective solving of any changes to the project tasks. Using Git as version control tool for the developing of the software made it possible for several groups to work on the same application.

Compared to the scope of the project, the group is pleased with the technical result. The solution meets all the requirements of technical functionality and the solution seems to be a modernised solution compared to the prior implementations of the system. Updating the communication standard from Bluetooth to Bluetooth Smart, rebuilding the server application from MATLAB to Java, and building a new robot based on an Arduino microcomputer can all be said to extend the system in a provident way.

Acknowledgements

This master thesis ends a two-year Master's degree programme for Cybernetics and Robotics at The Norwegian University of Science and Technology, Trondheim. The thesis constitutes the basis for evaluation of the subject "TTK4900 Teknisk Kybernetikk, Masteroppgave", and it counts 30p for each student.

A special thanks to our supervisor and employer Tor Engebret Onshus, professor at The Department of Engineering Cybernetics. Onshus has facilitated a good dialogue throughout the project period and has laid a solid foundation for the implementation of this project.

The group would like to thank the two other cooperating parties, Erlend Ese and Eirik Thon, for great teamwork. Without our close cooperation during the project period, the resulting product would not have been as good as it became.

Further, a thank goes out to the guys at both the Cybernetics Mechanical Workshop and the Cybernetics Electronic Workshop. The Mechanical Workshop has provided us with mechanical parts, while the Electronic Workshop has provided a safe environment for building and soldering the robot, as well as electronic parts needed throughout the project period.

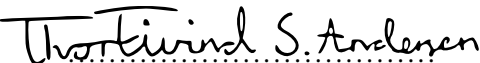
Finally we would like to thank the guys at office B117, for providing a social and engaging working environment.

"Do not seek to follow in the footsteps of others, instead, seek what they sought."

- Matsuo Basho


.....

Mats Gjerstet Rødseth
matsroedseth@hotmail.no


.....

Thor Eivind Svergja Andersen
te@svergja.com

Contents

Summary	i
Conclusion	v
Acknowledgements	vii
I INTRODUCTION AND THEORETICAL BASIS	1
1 Introduction	3
1.1 Background Information	3
1.2 Objective	4
1.3 Content of the Report	5
1.4 Previous work	6
2 Theoretical Basis	7
2.1 Physical	7
2.1.1 Electronics	7
2.1.2 Microcomputers	9
2.1.3 Inertial Measuring Unit	10
2.2 Communication	11
2.2.1 Serial Communication	11
2.2.2 Wireless Communication	12
2.3 Programming Platforms	13
2.3.1 Java	13
2.3.2 Arduino	14
2.3.3 C	14
2.3.4 MATLAB	15
2.4 Human-Computer Interaction	15
2.4.1 Design Process	15
2.4.2 The Gestalt Principles	16
2.4.3 Schneidermans 8 Golden Rules	17
2.5 System Architecture	19
2.5.1 Concurrent Programming	19
2.5.2 Code Complete	20

II	MATERIAL, METHODS AND RESULTS	23
3	Material	25
3.1	Software	25
3.2	Hardware	27
4	Project Management	31
5	Arduino-Robot	33
5.1	Designing the Robot	33
5.1.1	Acquiring Materials	33
5.1.2	Assembly	34
5.1.3	Wiring	35
5.2	Programming the Arduino-Robot	36
5.2.1	Requirements	37
5.2.2	Configuring Software	38
5.3	Improving the Robot	40
5.3.1	Designing the Motor and Axle Plastic Housings	40
5.3.2	Designing the Printed Circuit Board	41
5.4	The Final Product	43
6	Wireless Communication	45
6.1	Programming the nRF51-dongle	45
6.2	Developing the nRF51-server Software	46
6.3	Creating the Message Protocol	47
6.4	Distance and Integrity Tests	49
7	Server Application	53
7.1	Designing the System Architecture	53
7.2	Designing the Real-Time System Flow	55
7.3	Structure	55
7.4	System Flow	57
7.5	Functionality	59
7.6	Limitations	60
8	Graphical User Interface	61
8.1	Design Process	61
8.1.1	Sense the Gap	61
8.1.2	Understand and Refine the Problem	61

8.1.3	Explore	62
8.1.4	Evaluation and Feedback	65
8.1.5	Select Plan	66
8.2	Programming the Graphical User Interface	66
8.3	The Final Product	66
9	Testing the System	71
III	DISCUSSION, CONCLUSION AND FUTURE WORK	73
10	Discussion	75
10.1	Arduino-Robot	75
10.2	Using Bluetooth as Communication Protocol	75
10.3	Using nRF51 as Communication Unit	76
10.4	The Message Protocol	77
10.5	Graphical User Interface	77
10.6	Selecting the Programming Platform	78
11	Conclusion	81
12	Future Work	83
	Acronyms	85
	Bibliography	87
	Appendices	93

List of Figures

1.1	The three theses that form the system	5
2.1	Infrared triangulation [40]	8
2.2	Arduino Mega 2560 pin diagram [4]	9
2.3	Simplified UART Interface [18]	11
2.4	SPI Interface [19]	12
2.5	Java compiling process [38]	14
2.6	The four main steps in a design process [56]	16
2.7	Activities that form software construction [33]	20
2.8	The five levels of design in a program [33]	22
3.1	Arduino Mega 2560 [17]	27
3.2	Motorcontroller [17]	27
3.3	IMU [17]	27
3.4	Infrared Sensor [17]	28
3.5	nRF51-dongle [49]	28
3.6	Robot Chassis [17]	28
3.7	Motor and encoder kit [17]	29
5.1	Bottom part of the robot (prototype)	35
5.2	Top part of the robot (prototype)	35
5.3	Wiring diagram for the Arduino-robot, as built	36
5.4	Blinking led example in C	40
5.5	Wheels leaning towards the chassis before improvements	41
5.6	Eagle Schematic drawing	42
5.7	Eagle Board drawing	43
5.8	The finished Arduino-robot	44
6.1	Flow of the different message types between robot and server	49
7.1	Dividing the system into subsystems and packages during project planning phase	54
7.2	Structure of the system modules	56
7.3	Simple class diagram of the Java application	58

- 8.1 Graphical User Interface, suggestion 1 63
- 8.2 Graphical User Interface, suggestion 2 64
- 8.3 Graphical User Interface, suggestion 3 65
- 8.4 GUI, Main window 67
- 8.5 GUI Flow 69

- 9.1 Maze used during the testing of the system 71
- 9.2 Four map plots of the same maze 72

- 10.1 Start-Stop panel from GUI, Main window 78

List of Tables

3.1	Extra Material	30
6.1	Sketched list over messages from the Robot	48
6.2	Sketched list over messages from the Server	48
6.3	Message Integrity Test in Corridor	50
6.4	Message Integrity Test in Open Landscape	51
8.1	Criteria for functionality in the GUI	63
8.2	User feedback regarding GUI Design	65

PART I:

INTRODUCTION AND THEORETICAL BASIS

1 | Introduction

1.1 Background Information

It has always been important for human to record their surroundings, and the history of mapping can be traced back to more than 3000 years B.C. The first maps consisted of trade routes, hunting grounds and small areas that included places of interest. The maps were not very accurate, with bad relationships and few details included, but the maker was able to show the features that the maker wished to record [54].

The Greeks and the Romans redefined the art of mapping, and when Claudius Ptolemaeus published his work “Geographia” in year 150 A.D, the European geographic thinking was revolutionised [54]. As the years have passed, modern technology has expanded the ways people can map areas, from sundials and compass to the present GPS.

Today’s technology enables humans to automate the way areas are mapped, using swarms of cooperating robots that use sensors to “perceive” their surroundings. In situations where massive destructions have happened due to natural disasters, it is crucial to track survivors soon after the incident. Rescue teams cannot look further than 18-20 feet into a destroyed building using traditional probes and boroscopes [44], but modern technology allows search & rescue teams to send in several cooperating autonomous robots that can localise the survivors, and let the rescue team understand the areas based on the robot’s recordings [48]. The use of robots in search & rescue is not limited to natural disasters, but can also be applicable in man-made situations such as warfare areas and factory explosions.

Mapping unknown areas using cooperating robots has been studied and implemented through projects at NTNU during the last 12 years. In some projects the students have built robots that use different kinds of sensors, while others have designed/improved server software that exploits the data gathered by the robots. An important aspect of the projects has been to exploit the functionality of cheap parts to do advanced tasks, and this has also been the focus of this thesis.

1.2 Objective

The objective of this thesis is to re-implement the functionality of the existing solution in a new server application using a more suitable programming language, to build a robot based on a new type of microcomputer, as well as to update the wireless communication between the robots and the server using state-of-the-art technology.

Server Application

The new server application should provide real-time features in addition to being able to control several robots at the same time. Further, the application should:

- Implement the functionality from the existing solution.
- Have a system structure that is formed by modules, thus facilitating future work.
- Include a Graphical User Interface that presents the system's functionality in a more user-friendly way.

Robot

The new robot should add to the existing collection of robots, and should implement the following:

- The same sensors as existing robots.
- Arduino as microcomputer.
- Ese (2016) real-time Operating System (OS).
- The new wireless communication standard.

Wireless communication

In addition to developing a new server application and building a new robot, the group will select a new standard for the system's wireless communication, and develop the server-side software for the new communication.

1.3 Content of the Report

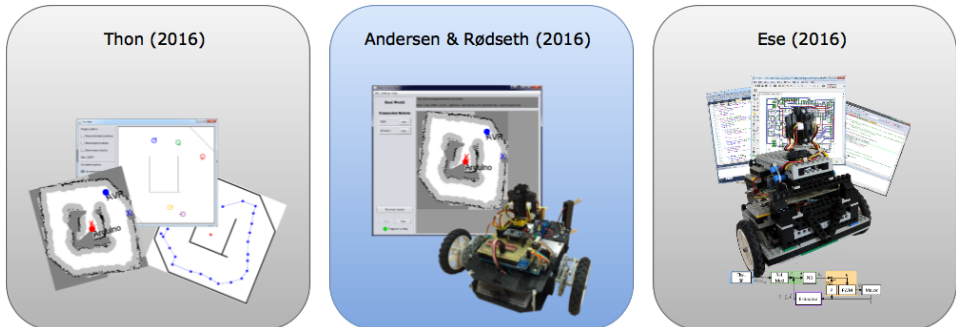


Figure 1.1: The three theses that form the system

This thesis covers the process of re-implementing the server application in the Java programming language, building an Arduino-based robot, designing and developing a Graphical User Interface (GUI), as well as updating the wireless communication using Bluetooth Smart and developing software for the server's communication unit. The thesis is a part of a threefold system, and together with two other theses it forms the "System for Self-Navigating Autonomous Robots (SSNAR)".

Thon (2016) covers the development of the system's simulator, the navigation and path planning, as well as the mapping of discovered area. Ese (2016) provides information about improving the physical robots, developing of the robot's communication unit, as well as developing of the robot's real-time OS, with all of its features.

This thesis consists of three parts. The first part, "Introduction and Theoretical Basis", provides information about background history, previous work, the objectives of the report and relevant theory for the thesis. Part two, "Material, Methods and Results", includes a list over used material (both software and hardware), and is further divided into chapters that cover both methods and results for the different parts of the system. The final part, "Discussion, Conclusion and Future Work", discusses and concludes over the results, and provides some suggestions for future work based on the findings in this thesis.

1.4 Previous work

When this thesis was started, a server control system developed in MATLAB existed. The server could connect to one robot with the use of a Bluetooth-dongle. If the user wanted to connect to several robots, several instances of the MATLAB server application had to be started. This meant that one instance of MATLAB per robot was needed. The different MATLAB instances, running on the same computer, communicated with each other via TCP/IP. The approach required a lot of computer resources and scaled badly when more than one robot was connected.

The MATLAB software relied on a third-party toolbox “Centre for Autonomous System Toolbox” first introduced in a project done by Syvertsen in 2005, and in the following years the server software and the robots were extended and modified several times. Please refer to Homestad (2013) for an extended elaboration about the content of the earlier reports.

During the project and master thesis of Halvorsen (2013 & 2014) cooperation between robots were introduced. The work with the NXT-robot, server application and the implemented simulator was continued.

The lack of the real-time aspect in the system laid the foundation for Ese (2015). During his project, the work with a real-time OS was started. In addition, the NXT-robot was re-built such that the sensors mounted on the robot were the same as on the AVR-robot, but the NXT-robot was not able to map autonomously. It could only produce simple maps, and the reason behind this was not investigated. The developed real-time OS was implemented on the AVR-robot during the project. Ese described the current state of the server application as full of bugs. The simulator worked but crashed sporadic, the application printed error messages during start-up leading to forced restart of the software, the collaboration did not work, and the GUI presented buttons that did not have any function.

2 | Theoretical Basis

This chapter provides necessary theory to understand principles and methods used in the thesis. It provides background information about the physical components in Section 2.1, the different types of communication used in Section 2.2 and the various programming languages used in Section 2.3. This chapter also contains background information about Human-Computer Interaction (HCI) in Section 2.4 and System Architecture in Section 2.5.

2.1 Physical

This section contains background information about some of the physical components used in this thesis.

2.1.1 Electronics

Capacitor

Capacitors can be found in different sizes and materials, the most popular being electrolytic capacitors for values greater than $1\mu\text{F}$, and ceramic capacitors for lower values. In this thesis there were mainly two applications areas for capacitors, these are further explained below.

If a component has a sudden high demand for power, the demand can be enough to drop the voltage in the circuit and other components can start to misbehave. By adding a high-value capacitor between ground and VCC, the capacitor will act as a reservoir of electricity, so that the component will draw the charge from both the supply and the capacitor. A typical component where this is needed is a servo motor that draws a lot of power as the motor is starting. A good default value for a servo motor is $430\mu\text{F}$, depending on the application [36].

The capacitor also has another great feature; in a DC circuit, it can filter out high-frequency AC noise. Not all components need pure DC signal, but some components, such as logic chips, may start to operate incorrectly if the voltages swing too much. By placing a capacitor between ground and VCC, the capacitor acts as a bypass capacitor and shorts the AC signal which removes any AC noise on the DC voltage. Typically this

kind of capacitor is smaller in size than the first described one; a good default value is 100nF ceramic capacitor [46].

For both type of capacitors, it is important to put it as close to the component as possible.

Infrared Sensor

Infrared (IR) sensors work by emitting an infrared light and then detecting if the light gets reflected back to the sensor. There are two types of IR sensors, one who only detects if an object is in close proximity and one that can determine how far away an object is. Since the sensor work by emitting and analysing infrared light, it can be affected by other infrared sources like sunlight [45].

The sensors are an affordable alternative compared to other range sensors. They measure at a narrow beam width which can result in a more detailed representation of the environment [45].

The distance to an object is measured by triangulation. When light returns, it comes at an angle dependent on the distance of the reflecting object, which Figure 2.1 illustrates. The distance can then be calculated from the angle. Light does not reflect the same way off every surface and the sensor reading will be different for different surfaces and/or colours even if the range is the same [45].

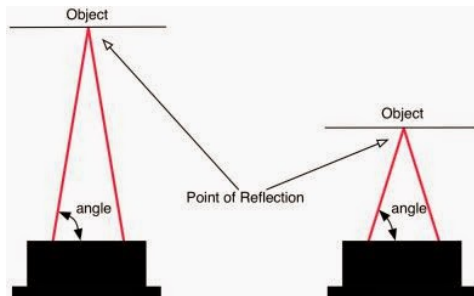


Figure 2.1: Infrared triangulation [40]

2.1.2 Microcomputers

Arduino Mega 2560

There are many different versions of Arduino boards, providing different physical sizes, processors and number of Input/Output (I/O) [3].

The Mega 2560 is based on the ATmega2560 and it is an improved update to the Arduino Mega. It provides 54 digital I/O pins and 16 analogue input pins. Figure 2.2 illustrates the mapping between the Arduino pins and ATmega2560 ports [2].

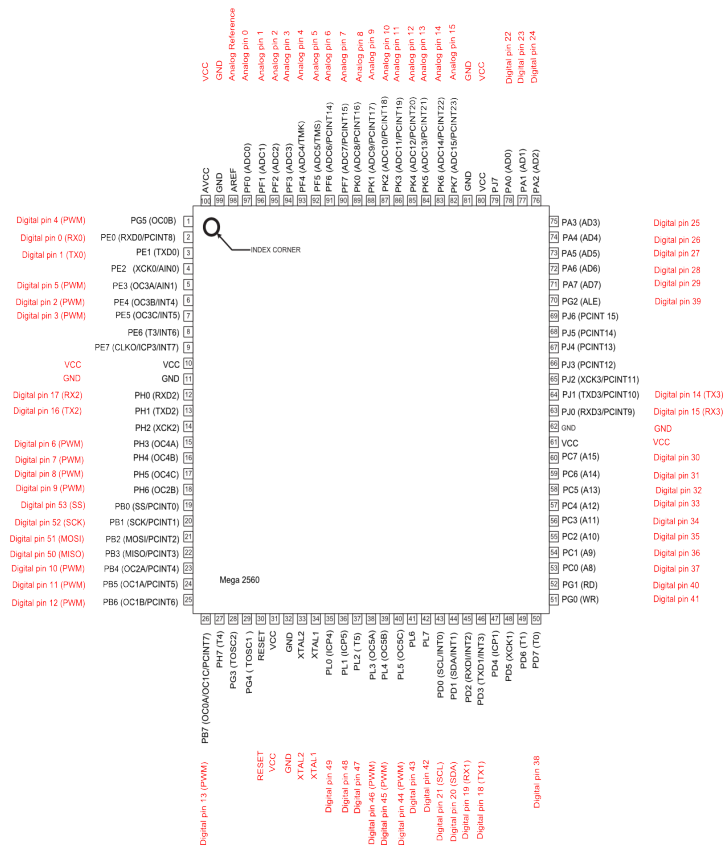


Figure 2.2: Arduino Mega 2560 pin diagram [4]

nRF51

The nRF51-series supports several different protocol stacks, including Bluetooth Smart (Bluetooth low energy), ANT and proprietary 2.4GHz protocols.

The nRF51-dongle is a low-cost USB development dongle; it has six General-purpose input/output (GPIO) pins and can be used as a peripheral between computers and robots [49].

2.1.3 Inertial Measuring Unit

Inertial Measurement Unit (IMU) is a single unit which collects angular velocity, linear acceleration and sometimes magnetic field data. An IMU consists of accelerometer, gyroscope and sometimes a magnetometer [58].

Accelerometer

There are several different types of accelerometers, and the most commonly used in inertial navigation systems are the *closed loop pendulous accelerometers*, which can be implemented in Microelectromechanical systems (MEMS). A mass is maintained in its null position by an electromagnetic device, the magnitude of the electric current in the coils change when there is a change of movement on the accelerometer [58].

Gyroscope

In the same manner as accelerometers, there are several types of gyroscopes. The most common for low and medium cost applications are the vibratory gyroscopes which can be implemented in MEMS. These gyros use the law of Coriolis to measure angular velocity, and are typically designed as an electronically driven resonator fabricated out of a single piece of quartz or silicon. The Coriolis force can then be measured capacitively in a silicon instrument or piezoelectrically in a quartz instrument [58].

2.2 Communication

Both serial and wireless communication are used in this thesis. This section provides background information about the different communication protocols.

2.2.1 Serial Communication

Universal Asynchronous Receiver/Transmitter

The Universal Asynchronous Receiver/Transmitter (UART) is a bus that acts as a converter between parallel and serial interfaces. It enables a computer that uses an RS-232 interface to communicate with peripherals that use two-wired Serial RX-TX (see Figure 2.3) [31]. UARTs are responsible for sending and receiving serial data. On the transmit side, a data packet is created and sent to the TX line, while on the receiving end, the UART has to sample the RX line at the expected baud rate [18].

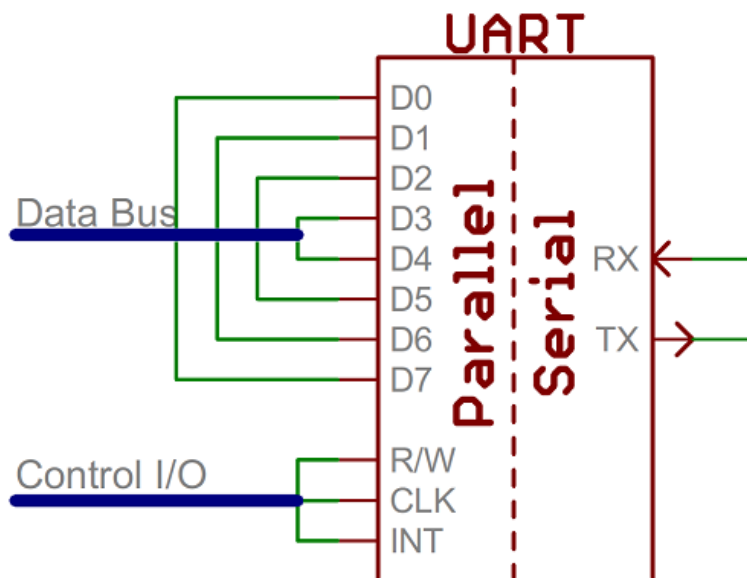


Figure 2.3: Simplified UART Interface [18]

Serial Peripheral Interface

Serial Peripheral Interface (SPI) was developed by Motorola to provide short-distance communication, primarily for use in embedded systems [11]. The standard provides full-duplex synchronous serial communication over a four-wired interface with the following signals [21]:

- *MOSI* - Master-out, Slave-in
- *MISO* - Master-in, Slave-out
- *SCLK* - Serial clock
- *SS* - Slave Select

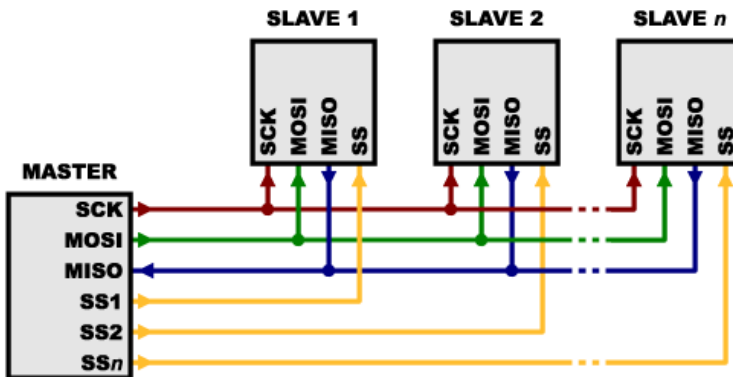


Figure 2.4: SPI Interface [19]

SPI is a single master-multiple slave protocol (see Figure 2.4), and the master sends/requests information to/from the slaves by pulling the SS low at the specific slave, activating the clock signal and generating information onto MOSI while it samples the MISO line [52]. For further reading and as an introduction to the practical use of SPI, see the “Serial Peripheral Interface (SPI)”-tutorial at SparkFun [19].

2.2.2 Wireless Communication

Bluetooth

Bluetooth is a low energy, short-range technology created in 1994 [28]. It was developed by Bluetooth Special Interest Group (SIG) and named after the viking king Harald Blåtand.

The intention of Bluetooth was to allow wireless connectivity and collaboration between products [27].

Bluetooth technology ensures that devices are capable of communicating with each other regardless of the manufacturer of the device [14].

Specifications

The different Bluetooth versions have different specifications, but as the newer versions are backwards compatible [41] they cover the older versions. As *Bluetooth Smart* is used in this thesis, specifications for that standard are listed below:

- Theoretical maximum data rate is 260 kbps.
- Power consumption is $\sim 100 \mu\text{Ah}$ per day.
- In 3ms, a device can connect, send and acknowledge data.
- Bluetooth Smart advertises on 2402, 2426 and 2480 MHz, and thus avoids interference with Wi-Fi traffic [13].

2.3 Programming Platforms

There are four different programming languages used in this thesis. The server application is programmed in Java, the communication dongle is programmed in C, the old program was programmed in MATLAB and testing were done on the robot with Arduino IDE. This section provides basic theory about these four programming languages.

2.3.1 Java

Java is an object-oriented programming language mainly developed by James Gosling and other developers at Sun Microsystems. The language is based on the “WORA” principle [59], “Write Once, Run Anywhere”. This means that Java does not compile to machine code, but instead is compiled and run by Java Virtual Machine (JVM). In this way one can run Java applications on all systems where there is a JVM [39].

All source code is written in plain text using the “.java” file format. The files are compiled to “.class” format that contains bytecode which JVM uses to run the application. The process is illustrated in Figure 2.5.

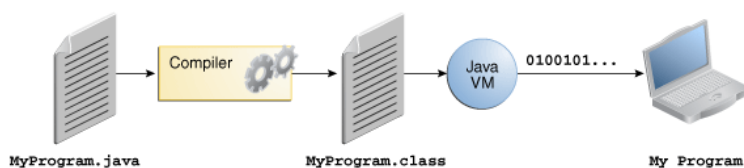


Figure 2.5: Java compiling process [38]

The Java Application Programming Interface (API) is a big collection of ready-made software components that are grouped in libraries that consist of classes and interfaces that are related to each other.

Java requires a software platform to run compiled applications, and one of the official platforms are “Java SE” [39].

2.3.2 Arduino

Arduino is an open-source platform based on easy-to-use hardware and software, developed at Ivrea Interaction Design Institute [5]. Its Integrated Development Environment (IDE) is cross-platform, and the programming environment is easy to use both for inexperienced and experienced programmers.

Arduino programmes may be written in any programming language with a compiler that produces binary machine code, such as C and C++ [22]. Its core libraries are written in C and C++ and is compiled using `avr-gcc` and `AVR Libc` [1].

2.3.3 C

The C programming language was developed by Dennis M. Ritchie and Bell Labs in the early 1970s [37]. The language provides low-level access to memory, making it suitable for programming embedded systems consisting of microcomputers. C is not tied to any particular hardware or system, it is cross-platform. Thus, the programmes will run on any machine that supports C.

C provides a variety of data types and high-level statements such as `if-else`, `switch`, `while` and `for`. Even though it enables access to memory, it does not provide `read/write` statements for input and output [8].

2.3.4 MATLAB

MATLAB is a high-level language and interactive environment for numerical computation, visualisation and programming. MATLAB makes it easy to analyse data and develop algorithms for a range of applications including signal processing. MATLAB is developed by Mathworks [32].

2.4 Human-Computer Interaction

“Design is conceiving and giving form to artefacts that solve problems”

- Karl T.Ulrich

2.4.1 Design Process

In the process of making an artefact, four main steps are defined (see Figure 2.6) :

1. “Sense gap”: Design begins with a gap in the user experience. Without a gap in the user experience, there is no need for design.
2. “Define problem”: The gap has to be explained. At this stage of the design process the user-needs have to be identified, and in the identification phase, there is a possibility that the basis for the design process becomes wrong. The user is often not certain in what he wants, and what he explains can be perceived wrongly by the designer. It is important that the “perfect system” is described as thoroughly as possible.
3. “Explore alternatives”: In this step, the designer tries to explore several alternative ways to find different solutions to the problem. Involving the user in user-testing, including contacting people that have an understanding of the problem, can give an idea of the right approach, thus the designer can provide multiple solutions.
4. “Select plan”: The exploration usually provides several solutions to the problem, and therefore the selection of the plan is an important stage of the design process. Through conversations with the user, the designer can narrow down the results of the previous phase to the best solution.

It is common to iterate through steps 1-3 several times before the best option is found [56].

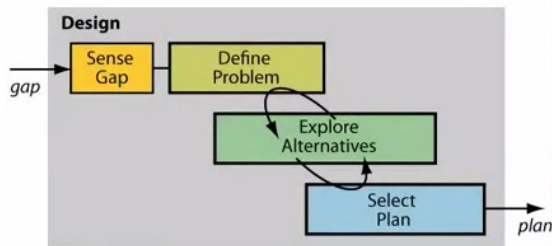


Figure 2.6: The four main steps in a design process [56]

2.4.2 The Gestalt Principles

The Gestalt Principles is a framework describing how users experience and perceive visual interaction with a user interface. Humans are “programmed” in a way that they should perceive their surroundings as whole objects, and they search for structure and contexts instead of accepting discontinuous lines and figures. There are several Gestalt Principles, the most relevant for this thesis being [30]:

1. Proximity
2. Similarity
3. Continuity
4. Figure/Ground

Proximity

The distance between objects influences the user perception of whether the objects are organised in groups or not. Objects close to each other are perceived as connected and thus are influenced by each other. The Proximity Principle affects a Graphical User Interface (GUI) in a way that if the GUI is separated in groups, the user will easier understand the system [30].

Similarity

The similarity between objects causes the objects to feel grouped, and objects that are not in close proximity to each other can still be perceived as having relevance to each other [30].

Continuity

The human mind is “programmed” to see continuity and hence it fills in information to perceive discontinuous shapes as continuous [30].

Figure/Ground

The visual system structures information perceiving that objects in the foreground have a greater importance than objects in the background. Smaller objects placed on bigger objects are perceived as a figure while the larger objects are perceived as the ground beneath the figure. Using the principle of figure/ground can enable the GUI to present valuable information in the foreground [30].

2.4.3 Schneidermans 8 Golden Rules

The eight golden rules of Schneiderman are guidelines for GUI development. The principles were developed based on experience gained by perceptual and cognitive psychology in an attempt to improve the design of interactive systems. According to Schneiderman [7], the guidelines needs to be validated and adapted to specific design domains. A list like this will never be complete, but it has been received as a useful guide for students and designers.

1. Strive for consistency

Actions should be consistent in similar scenarios. This means that the colour, layout and font selection should be consistent, menus and help screens should use the same terminology, and so on [7].

2. Cater to universal usability

Frequent users want to minimise the number of interactions to increase the pace of interaction. Abbreviations, function keys and hidden commands suits the expert user, while explanations and features for novices suits the inexperienced user [7].

3. Offer informative feedback

Every user action should provide a system feedback. Frequent and minor actions should provide a modest response while infrequent and major actions should provide a more substantial response [7].

4. Design dialogue to yield closure

Action sequences should have a beginning, a middle, and an end. At the end of the sequence, the system should provide informative feedback to give the operator the satisfaction of accomplishment [7].

5. Prevent errors

The system should be designed in a way that the user is unable to make a serious error. If the user has made an error, the system should offer an easy instruction for recovery [7].

6. Permit easy reversal of actions

Actions done by the user should be reversible. Providing this feature relieves anxiety since the user knows errors can be undone [7].

7. Support internal locus of control

Experienced users should experience that they are in charge of the interface. They want the interface to respond to their actions without surprises. If the user is forced to fill in large amounts of data, has difficulties to get the required information, and has difficulties to perform actions, his satisfaction will decrease and stress increase [7]

2.5 System Architecture

2.5.1 Concurrent Programming

A concurrent program consists of several streams of operations that may execute concurrently. The program is built in a way that the streams execute at the same time and they can communicate with one another [43].

Threading

Each stream of instruction in a program is called a *thread*. If a multi-threaded application is running, one of the threads may override the others in a way that blocks them from running. This is called *starving*. To prevent starving (the Java language does not guarantee this), most JVM provides fairness allowing all of the threads to execute their tasks [43].

Shared Objects

According to “The little book of semaphores”[16], objects and variables that are accessed by several threads are called “shared objects/variables”. The use of shared objects is a way to make threads interact with one another, but it is also a way to introduce concurrency problems to the application. Some of the problems and solutions are further explained below.

Race Conditions

Race conditions happens when several threads tries writing to/reading from the same variable or object at the same time. The resulting outcome depends on the order of execution, and in worst case, the result may be erroneous [53].

Synchronization

The main preventing action in Java to avoid race conditions is object locking. By synchronizing a block of code or a whole method, the object lock-key will be acquired by the first thread that enters the code block, while the other threads will have to wait. This way one can prevent that several threads access variables at the same time [43].

Deadlock

Deadlock is the ultimate form for starvation (see Section 2.5.1). Deadlock happens when two or more threads are waiting on a condition that cannot be satisfied [35]. The most classic problem to illustrate how deadlock happens is “The Dining Philosophers Problem”[42]. In the philosophers problem, no philosopher can eat unless he has two forks, and since there are not enough forks at the table for them to acquire, all of the philosophers starves to death.

2.5.2 Code Complete

Software Construction

Developing software is a complicated process, and there are several activities that form the software construction. Topics like “Problem definition”, “Software architecture”, “Integration” and “System testing” are central in the process of constructing a well functioning application (see Figure 2.7).

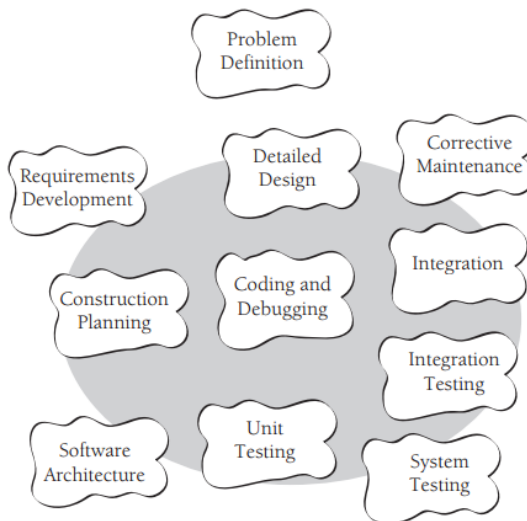


Figure 2.7: Activities that form software construction [33]

When designing software, it is important to minimise the complexity. Make the system easy to understand, without making designs that are “clever” inside your head. The system should be self-explanatory in the means that it should be easy to understand if someone in the future would want to do some maintenance to it. It is important that

the connection between different parts of the program follow the principles of “High cohesion” and “Loose coupling”:

- Cohesion - Cohesion refers to how the classes in the system are defined to take care of one responsibility, and how the different classes use methods and such from one another [9].
- Coupling - Good abstractions, encapsulation and information hiding makes the number of interconnections as low as possible, and thus leads to loose coupling [9].

Extensibility and re-usability results in easier maintenance and implementation as well, and should therefore be taken into account.

Levels of Design

When starting the design of a software application, Steven McConnell mentions the five levels of design. The five levels of design, shown in Figure 2.8, describes the levels of detail in a software system and how the system developer should approach the problem [33]:

- Software system - Describe the entire system and how it should work.
- Division into subsystems/packages - Describe how the system should be built up by subsystems that take care of different areas of responsibility, and how the subsystems should use each other.
- Division into classes within packages - Identify the classes in the system and describe how the classes interact with the rest of the system.
- Division into data and routines within classes - Describe the class routines in detail. This will lead to a better understanding of the class' interface.
- Internal routine design - Describe the detailed functionality of the routines (e.g. using pseudocode).

Naming Conventions

It is important to choose good, describing names when programming a complex software system. The routine-name should describe the behaviour and the output of the routine, and in that way, it is clear what the routine should be used to do. The routines should not be differentiated by number, but by functionality. The variables should also have describing names, and be differentiated from the types using lower case on the first letter: `TypeName variabelName`

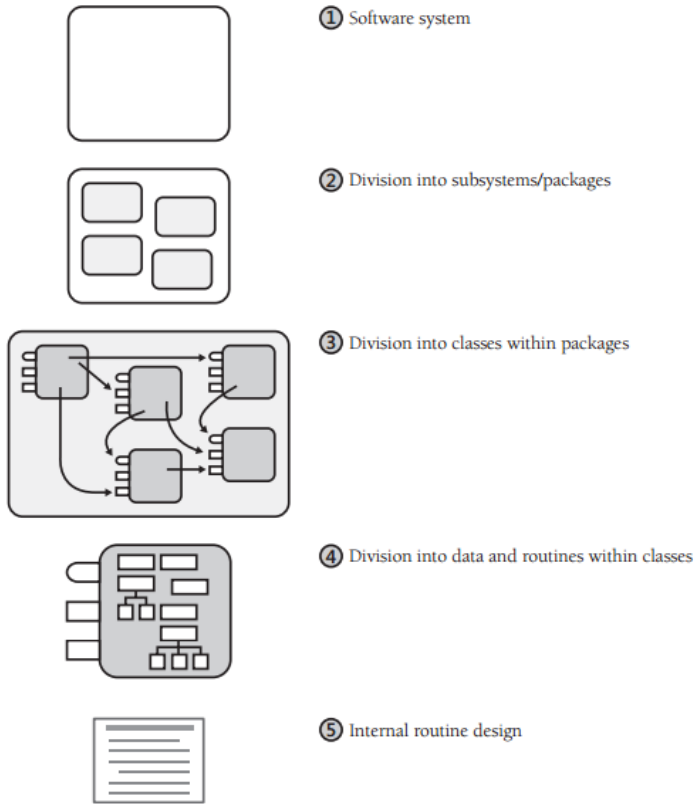


Figure 2.8: The five levels of design in a program [33]

The different programming languages have different conventions. Some of the Java-specific conventions are [33]:

- Class names capitalise the first letter of each word.
- *i* and *j* are integer indexes.
- *get* and *set* are used for accessor methods.

PART II:
MATERIAL, METHODS AND RESULTS

3 | Material

The following sections provides a list and describes the software and hardware components used in this project.

3.1 Software

- **Arduino IDE**, described in Section 2.3.2.
- **Atmel Studio** (version 7) is an integrated development platform for developing Atmel-based microcontroller applications, by Atmel Corporation. The environment provides write, build and debug functionality for an application written in C/C++ or assembly code. Windows is the supported operating system [12].
- **Eagle** (version 7.5), Easy Applicable Graphical Layout Editor, is a Printed Circuit Board (PCB) design software. It provides three modules, where the user can design the schematics, edit the board layout and route the wires manually/automatically. The software is cross-platform and is available both for Windows, Linux and Mac [57].
- **Keil μ Vision 5** combines source code editing, program debugging, project management and build facilities in one environment. The software provides functionality for building and programming device peripherals [24]. μ Vision can only be installed on Windows.
- **MATLAB** (version R2015a), described in Section 2.3.4.
- **Microsoft Project** is a project management tool for Windows that let the users plan their project progress in detail. It provides information about scheduled workloads for all the participants and helps to keep progress in the project. Through the software, the team can generate useful illustrations like Gantt diagrams to get a clear presentation of the project development [34].
- **MicroStation** is a Computer-Aided Design (CAD) software for producing drawings and documentation, developed by Bentley. The software can e.g. be used to make wiring diagrams for embedded systems [26]. Version 8.11 is used in this thesis. MicroStation can only be installed on a Windows computer.

- **NetBeans** is a tool for software development. Installing different plugins let the user write programs in Java, JavaScript, HTML5, C/C++ and more. The software is cross-platform and is available for Windows, Linux and Mac and the latest version available is NetBeans IDE 8.1 [10].
- **nrfGoStudio** is a Windows application, developed by Nordic Semiconductor, which eases development when using the nRFgo Development kit. The software provides a visual editor and functionality for configuring and programming nRF51-devices [51].
- **SolidWorks** is a modelling CAD software, developed by Dassault Systèmes. It provides functionality for design, simulation and publishing of 3D-modelled figures. Windows is the supported operating system [55].
- **Source Tree** is a free client for Git and Mercurial projects. It provides a straightforward graphical user interface and makes it easy to manage projects [6]. It can be installed on both Mac and Windows.

3.2 Hardware

Arduino Mega 2560 R3

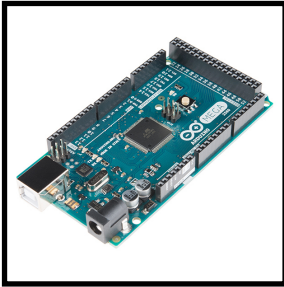


Figure 3.1: Arduino Mega 2560 [17]

- ATmega2560 microcontroller
- Input voltage - 7-12V
- 54 Digital I/O pins (14 Pulse-Width Modulation (PWM) outputs)
- 16 Analog inputs
- 256k Flash memory
- 16Mhz Clock speed
- Bought from SparkFun
- 45.95 USD

Dual Motor Driver

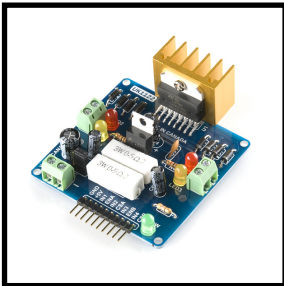


Figure 3.2: Motorcontroller [17]

- Dual bidirectional motor driver
- On-board user-accessible 5V regulator
- Enable and direction control pins
- Bought from SparkFun
- 34.95 USD

IMU, 6 Degrees of Freedom

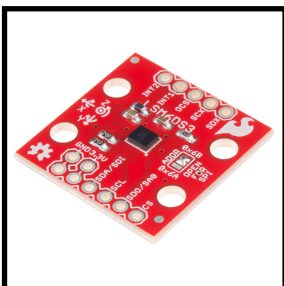


Figure 3.3: IMU [17]

- Low power consumption
- Smart FIFO up to 8 kB buffer
- Featuring both I^2C and SPI
- 3.3V device
- Bought from SparkFun
- 19.95 USD

Infrared Proximity Sensor - Sharp



- 4x Infrared (IR) sensors
- Range: 10-80 cm
- Bought from SparkFun
- 13.25 USD each

Figure 3.4: Infrared Sensor
[17]

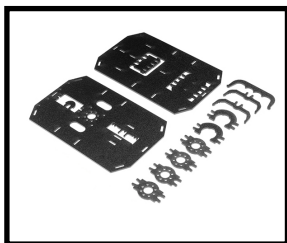
Nordic Semiconductor nRF51-dongle



- Supports Bluetooth Smart, ANT and 2.4GHz
- Virtual COM port interface through UART
- 6 solder pads for GPIO/interface connections
- Bought from Omega Verksted, NTNU
- 150 NOK

Figure 3.5: nRF51-dongle
[49]

Shadow Chassis



- Light weight ABS plastic.
- 196mm Length
- 126mm Width
- 44mm Height
- Bought from SparkFun
- 12.95 USD

Figure 3.6: Robot Chassis
[17]

Wheels, motors & encoders



- 2x 65mm Wheels
- 2x DG02S Mini DC gear motor
- 2x Neodymium 8-pole magnets
- 2x Hall effect sensor
- Bought from SparkFun
- 19.95 USD

Figure 3.7: Motor and encoder kit [17]

In addition, the following parts (see Table 3.1) have been used during this project.

Please refer to Appendix 1 for datasheets regarding the hardware.

Table 3.1: Extra Material

Part	Amount	Description	Acquired From	Cost
3D printed materials	4	Motor holders and axle holders	MW ^a	-
Ball caster	1	3/8" metal ball caster	SparkFun	2.95 USD
Battery	1	11,1V 4600mAh lithium battery	Elfa Distelec	938 NOK
Breadboard	1	For prototyping	EW ^b	-
Capacitors	2	220µF before the micro servo, 100nF before nRF51	EW	-
Charging towers	2	Plastic tower that fits the charging station	MW	-
Compass	1	For better position estimates	Erlend Ese	-
Glass fuse	1	To protect the robot's circuit	EW	-
LEDs	3	Status LEDs, red, green and orange colours	EW	-
Lego bricks	6	Various size and form, used on IR tower and gears	Tor Onshus	-
Lego gears	8	Gear down the wheels	Tor Onshus	-
Logic level converters	3	Bi-Directional, 5V to 3.3V and 3.3V to 5V	SparkFun	2.95 USD each
Metallic parts	5	For IR tower, motors and ball caster	MW	-
Nuts	20	Different sizes	MW	-
PCB shield	3	Printed at ElproLab, soldered at EW	ElproLab	-
Resistors	3	To limit the current in the LEDs	EW	-
Screws	20	Different sizes	MW	-
Springs	4 x 4cm	Conductive springs for the charging towers	MW	-
Strips	15		MW	-
Switches	2	On/Off switches for power and charging	EW	-
Wheels	2	New softer wheels, 81,25cm diameter	EW	-
Wires	-	Various wires used to connect components	Erlend Ese	-
			EW	-

^aMechanical Workshop^bElectronic Workshop

4 | Project Management

Using Microsoft Project (described in Section 3.1) as project management tool, has enabled the group to get useful insight in the project progress. Tools like MS Project is letting the user plan every part of a project in detail, which is necessary and valuable both for small and large projects (e.g. a Master Thesis).

MS Project has provided information about how the project has progressed and made it possible to adjust the workload according to the planned progress. The adjustments provided a chance to stick with the schedule and made the group able to complete the project within the deadline. Appendix 9 contains Gantt diagrams from each month.

Throughout the project, the team has logged all that has been done, and this has proven to be an useful resource during the writing of the thesis. The log contains how many hours each group member has worked, and which tasks the members has been working with at the given time. The log has contributed to a general insight in what the other member has been working with. Each month the group documented their progress with a progress report, these are written in Norwegian and can be viewed in Appendix 8.

As well as logging the work, the group held meetings with both the employer and the other teams that have been working on other parts of the system. Doing so, has enabled the teams to cooperate the best way possible, and made them able to finish the project in time. The group has logged all meetings with the employer as “Minutes of Meeting” and they can be viewed in Appendix 5.

During the development of the software application, Git was used to ensure proper version control, and thus enabled the groups to cooperate writing code on the same application. Future projects can access a shared repository which will be updated after project end, and to prevent misunderstandings, the repository that was used during this project will not be available. This is because this project has more than 100 commits alone, and if every project were on the same repository, it would get messy fast. The thesis was also written using version control with Git.

In addition to version control, the teams used a shared document in Microsoft OneNote to share thoughts and ideas throughout the project period. Dropbox was also used to share relevant documents.

5 | Arduino-Robot

This chapter describes the whole process making the Arduino-robot, from designing and building, to programming and testing. Section 5.1 provides information about how the robot parts were acquired and connected, Section 5.2 describes how to program the Arduino, Section 5.3 explains how the robot was improved, while Section 5.4 presents the final robot and its functionalities.

5.1 Designing the Robot

The robot design was draughted during the first week of the project. During that week, the group discussed which parts were necessary to build a robot that met our goal. The entire robot was planned out before ordering any parts to make sure that there were no unnecessary purchases. It was important to order all parts as soon as possible as the shipping time can be significant when ordering from the US.

5.1.1 Acquiring Materials

The chassis and most of the electronic parts of the Arduino-robot were ordered from SparkFun [17] early in the second week. As the project progressed, some additional parts were needed and the group ordered metallic parts and the two charging towers at the Cybernetics Mechanical Workshop at NTNU.

The robot should be able to charge its battery, and as there is only one charger that all of the robots shares, the robot should be able to use that one. Because of this, the Arduino-robot has an 11.1V lithium battery which suits the named charger. The battery was bought from a Norwegian vendor called Elfa Distrelec[15], since the price of lithium batteries was the same as ordering from outside Norway.

Ese (2016) bought five nRF5-dongles from Omega Verksted NTNU for communication use, and one of these was mounted on the Arduino-robot.

The PCB was designed in Eagle, see Section 3.1, and printed by Elprolab[20] at NTNU. The plastic housings around the motors were designed in SolidWorks and 3D printed by the Cybernetics Mechanical Workshop.

The switches, LEDs, resistors, capacitors and wires were all acquired from the Cybernetics Electronic Workshop at NTNU.

5.1.2 Assembly

The robot chassis consists of two parts that can be separated into top and bottom. Additionally, small plastic pillars were installed all around the robot to connect and support the two parts, as well as the charging towers were designed to add an extra connection between the two.

Bottom

The driving wheels were initially mounted in the center on the bottom plate; this makes the robot turn radius as small as possible. The IMU was placed between the two wheels at the exact center of the robot, which makes the IMU output as precise as possible. The wheels and IMU were moved at a later stage of the project, this is further explained in Section 5.3. The metal casting ball was mounted in the center back of the car, and a thin metal plate was cut to make sure the casting ball was at the correct height so that the bottom plate were parallel to the ground. The battery, which is the heaviest component of the car, was placed right above the casting ball, making sure the robot was tail heavy. As there was only a small space left in the front of the bottom plate, the breadboard was placed there. Figure 5.1 shows the bottom part of the robot.

Top

The servo and the IR tower was placed in such order that the tower rotates at the center of the robot. The tower itself was made out of Lego bricks, the same way as the IR tower on the other robots. It is easy to take off and switch the sensors if needed. The motor controller is placed right behind the IR tower, and in front of the tower is the Arduino mounted. Next to the motor controller there are two switches, one for powering the robot and one for enabling the charging towers. A switch on the charging tower is something the other robots do not have, but the group decided that it was necessary since driving with the charging towers enabled is a hazard. E.g. if the robot moves around and makes the charging tower hit something that can conduct electricity, the switch will eliminate the danger of short-circuiting the battery. The switches are marked with clear symbols to reduce user failure. Additionally, there is installed a glass fuse to protect the Arduino. The motor controller, IR tower and the Arduino is elevated so that wires can run beneath them. Connected to the top plate but facing towards the bottom plate is the wheel encoders. The encoders are placed in such order that they match with the motors

mounted on the bottom plate, with about 2mm distance from the magnet rings on the motors. Figure 5.2 shows the top part of the robot.

There are several pictures and videos of the robot in Appendix 3.

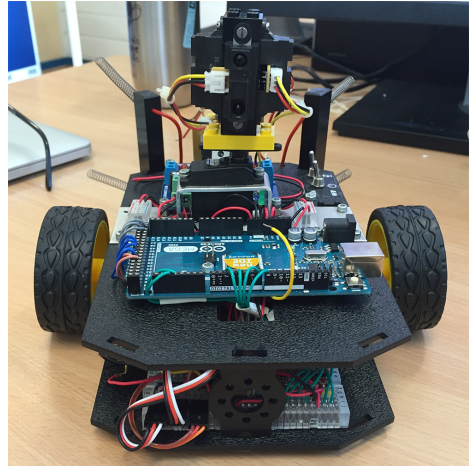
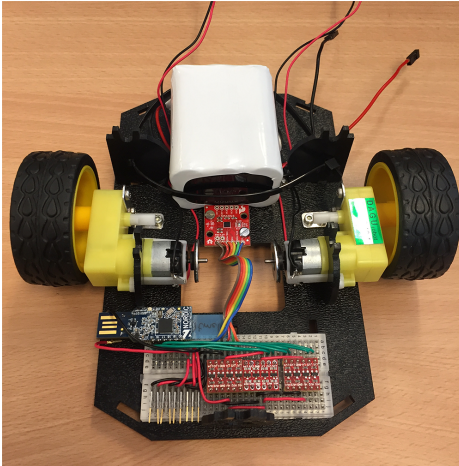


Figure 5.1: Bottom part of the robot (prototype) **Figure 5.2:** Top part of the robot (prototype)

5.1.3 Wiring

Before the components were connected, a wiring diagram was created. The diagram was updated whenever an improvement on the design was made, and Figure 5.3 shows the newest version. The diagram ensured that all connections were done correctly and made it easier for maintenance/troubleshooting. Standard colouring conventions were used, with red as positive power, black as ground, while signal cables used other colours depending on the component. When the prototype first was created, all wires were connected to a breadboard. This made it easy to test if all components were correctly connected and if they worked as intended. At a later stage in the project, the breadboard was replaced by a PCB that acts as an Arduino shield, further explained in Section 5.3.

Connections between components were made with self-made jumper wires, which makes it more straightforward to troubleshoot than many single wires on the same board. Soldering was used to connect wires to the two switches and the charging tower springs. All the soldering work was done in the Cybernetics Electronic Workshop since the workshop provides a safe environment with sufficient ventilation.

As seen in Figure 5.3 the 5V was taken from the voltage regulator on the motor controller

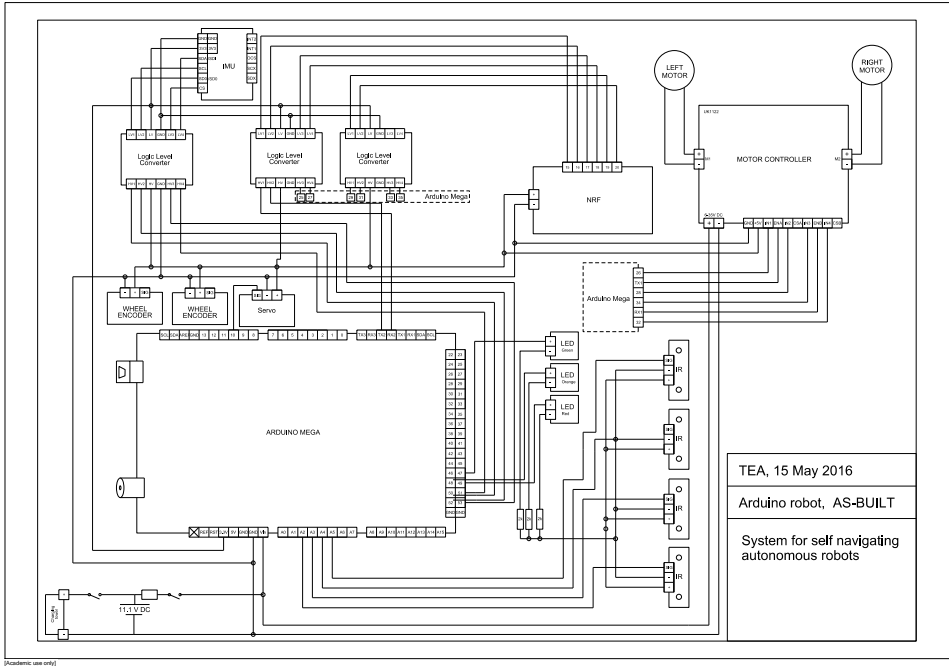


Figure 5.3: Wiring diagram for the Arduino-robot, as built

and not from the Arduino, this because the voltage regulator on the Arduino could not supply enough power. The IMU and the six GPIO pins on the nRF51-dongle require 3.3V power, so these wires go via the Logic Level Converters. For communication, nRF51 uses UART (Section 2.2.1) and the IMU uses SPI (Section 2.2.1).

As explained in Section 2.1.1, a capacitor is often used to prevent voltage level drop when components draw too much power. When the servo motor starts it demands a lot of power, so a capacitor is placed right next to it. Also, a ceramic capacitor is placed next to the nRF51-dongle to remove AC noise on the DC voltage.

5.2 Programming the Arduino-Robot

It was important to make the Arduino-robot able to use the same software as the other robots; this would save work and help the robots always having the newest software. The existing robots are programmed in C and since the Arduino language is a set of C/C++ functions it is possible to copy the code directly and use it on the Arduino Mega. It is also

possible to set-up a computer to directly compile and upload C code to the ATMEGA2560 chip on the Arduino. There are several steps to this, but once everything is installed and properly configured it is quick and easy to make changes and upload new programs. By doing so, the same program can run on all the robots from the different projects, and if there is an improvement in the code in the future, it can easily be installed on all of the robots.

The software that runs on the Arduino-robot is the same as the one on the AVR-robot, for more information about the software see Ese (2016). The next subsections will focus on how to program C code on the Arduino.

5.2.1 Requirements

AtmelStudio 6.2

It is possible to configure external tools such as AVRdude, to run straight from AtmelStudio 6.2 (described in Section 3.1). This makes it easy to edit and upload new programs to the Arduino.

AVRdude

As mentioned above, AVRdude is used to program the Arduino board. AVRdude is a command-line program and by passing the correct arguments, it is possible to program an Arduino board. The easiest way to find the right arguments is by showing verbose output during an upload from Arduino IDE.

AVRdude comes with a normal Arduino IDE installation and can be found in “Arduino/hardware/tools/avr/bin/avrdude.exe”. It is also possible to build AVRdude from source files, but this is a process much more complex, so it is recommended to download and install Arduino IDE.

Arduino IDE

Described in Section 2.3.2.

5.2.2 Configuring Software

There are several steps to configure AtmelStudio to work with AVRdude and Arduino. First, we need to generate the correct command-line parameters for AVRdude, then implement them into AtmelStudio. The process is explained below.

Create command-line parameters for AVRdude

1. Start Arduino IDE and plug in your Arduino board.
2. Press Tools -> Board, and find the correct board. In our case:

Arduino/Genuino Mega or Mega 2560

3. Press Tools -> Processor and find the correct processor. In our case:

ATmega2560 (Mega 2560)

4. Press Tools -> Port and set the right COM-port. If everything is configured properly, the board will be listed behind the COM-port name.
5. Press File -> Preferences and enable verbose on upload.
6. Upload a minimal example project. If everything is configured properly AVRdude will print out some red text.
7. Copy and paste the last white line of text into a text editor e.g. Notepad. In our case the line is as follows:

```
C:/Arduino/hardware/tools/avr/bin/avrdude -CC:/Arduino/hardware/tools/avr/
etc/avrdude.conf -v -patmega2560 -cwiring -PCOM6 -b115200 -D -Uflash:w
:C:/Users/teanders/AppData/Local/Temp/
build219bee5ee35bdb906c90832b67c0fe23.tmp/teste.ino.hex:i
```

This is the arguments AtmelStudio needs to use to upload a project.

Configure AtmelStudio

1. Start Atmel Studio 6.2.
2. Press File -> New -> Project -> GCC C Executable Project -> choose the correct processor, in our case:

ATmega2560

3. Create a minimal example code, or copy paste the blinking led example in Figure 5.4.

4. Press Tools -> External Tools...

5. Change the Title to:

```
&Deploy code
```

6. Change Command to the first part of the text that is copied from AVRdude and then append “.exe”. In our case:

```
C:/Arduino/hardware/tools/avr/bin/avrdude.exe
```

7. Change arguments to “-F ” + the parameters from the text copied from AVRdude. In our case:

```
-F -v -patmega2560 -cwiring -PCOM6 -b115200 -D
```

The path in Uflash needs to be a variable in order to get the correct path from AtmelStudio. To manage this, change “-Uflash...hex” to:

```
-Uflash:w:"$(ProjectDir)Debug/$(ItemFileName).hex":i
```

Last add the path for AVRdude config file, in our case:

```
-CC:/Arduino/hardware/tools/avr/etc/avrdude.conf
```

The total argument field is in our case:

```
-F -v -patmega2560 -cwiring -PCOM6 -b115200 -D -Uflash:w:"$(ProjectDir)
  Debug/$(ItemFileName).hex":i -CC:/Arduino/hardware/tools/avr/etc/
  avrdude.conf
```

8. Toggle “Use Output window” on.

9. If everything is configured properly it is now possible to upload C code directly to the Arduino board by pressing Tools -> Deploy code.

NB! Quotes are needed before and after the file structure if the file structure contains spaces, e.g.

```
-C"C:/Program Files (x86)/Arduino/hardware/tools/avr/etc/avrdude.conf"
```

```
/*
 * Blinking led
 *
 * Created: 19.02.2016 10:15:22
 * Author: teanders
 */

#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    DDRB = (1<<PB7);
    while(1)
    {
        PORTB = (1<<PB7);
        _delay_ms(100);
        PORTB = 0;
        _delay_ms(100);
    }
}
```

Figure 5.4: Blinking led example in C

5.3 Improving the Robot

After the robot had been built according to Section 5.1, the robot had a few flaws. Since the group was a few days ahead of the schedule, some improvements to the initial design were made.

5.3.1 Designing the Motor and Axle Plastic Housings

The assembled robot was too heavy in the center, and this led the wheels leaning towards the chassis. This can be seen in Figure 5.5. As this would have influenced the performance of the robot while driving, the group decided to improve the assembled robot to neutralise the maldistribution of weight. It was decided to design and 3D-print a motor holder that prevents the wheel angulation.

There was some problematics with PWM velocity regulation using the original wheel and motor; please refer to Ese(2016) for elaboration. The groups decided to gear down the rotation speed from the motor instead, but when doing so, the original axle got broken. As it now was needed to fix the motor by making a new shaft, it was possible to build a new gearing system using the same parts as used on the AVR-robot. Using different Lego parts to gear down the rotation presented the need for an axle housing to hold the new

wheel axle in place.

As SolidWorks, see Section 3.1, is a modelling software, it was suited to use for this task. Measuring the physical robot gave an accurate basis for designing the motor and axle housing, taking screws and nuts mounted to the chassis into account. The measurements and shapes obtained from the physical robot were modelled in SolidWorks and was 3D-printed at the Cybernetic Mechanical Workshop at NTNU. Detailed schematics for the parts can be found in Appendix 6 “Modelled Parts”.

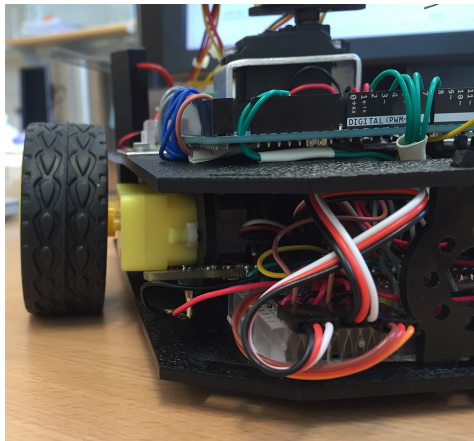


Figure 5.5: Wheels leaning towards the chassis before improvements

5.3.2 Designing the Printed Circuit Board

Connecting wires using a breadboard suits a prototype, but a finished product should have a PCB that connects the different components together. When all components were tested and the wiring was confirmed to be correct a PCB was designed in Eagle. The following five steps explains the process:

1. The wiring diagram created in Microstation, see Figure 5.3, was converted to a schematic in Eagle. The schematic determines how the different components are connected to each other, as can be seen in Figure 5.6. The Eagle-file and a full-scale picture can be found in Appendix 16 “Wiring Diagrams”.

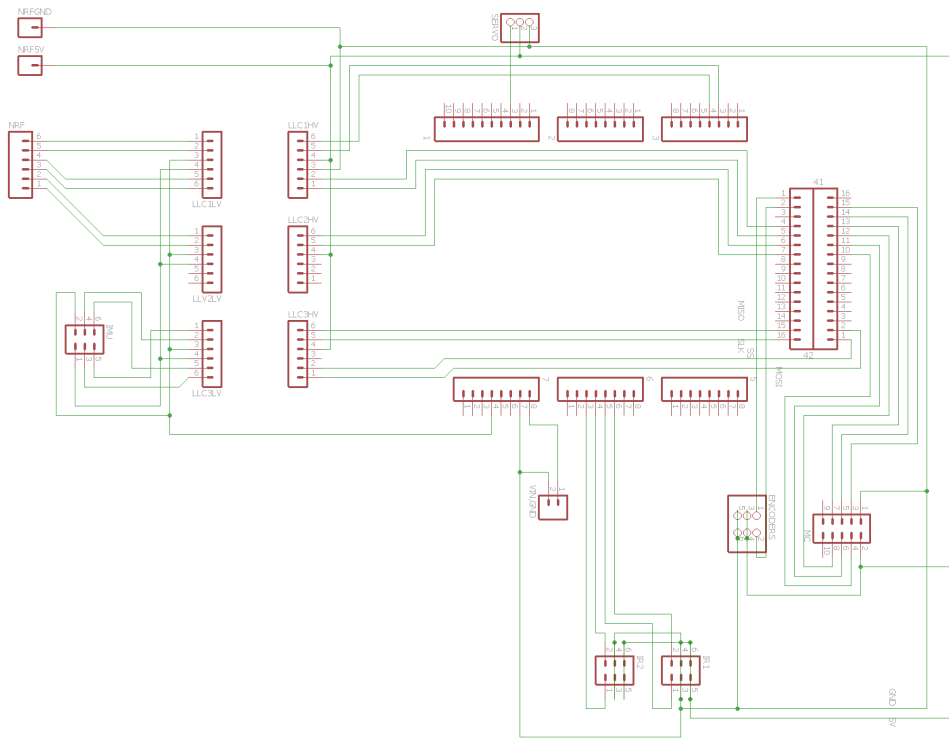


Figure 5.6: Eagle Schematic drawing

2. From the schematic, the board is created in Eagle. The board file determines the layout and how the different components and wires should be printed on the finished PCB. The board was designed in such order that it could be placed right on top of the Arduino. Figure 5.7 illustrates the board design. The Eagle file and a full-scale picture can be found in Appendix 16.
3. When the PCB design was finished in Eagle, the Eagle-files was converted to Gerber-files. The Gerber-files gives instructions to the PCB printer for how the board should be printed.
4. The Gerber-files were sent to Elprolab and the PCB was printed.
5. Every component were soldered onto the PCB at the Cybernetics Electronic Workshop.

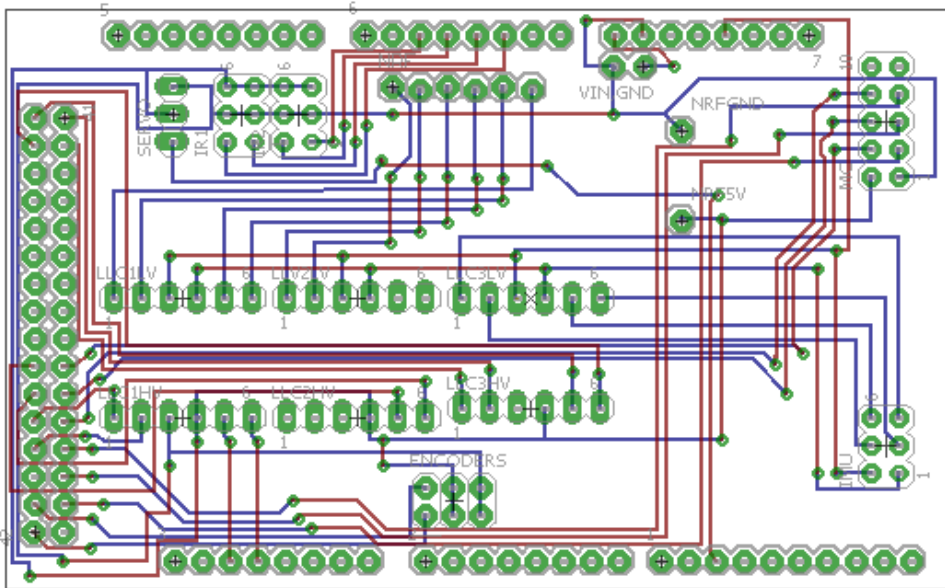


Figure 5.7: Eagle Board drawing

5.4 The Final Product

The finished Arduino-robot is pictured in Figure 5.8. In addition to the improvements described above, some minor improvements were made. Three LEDs (Red, Orange, Green) were added to help with debugging, and the AVR-robot has the same three LEDs. To assist the position estimation, Ese (2016) added a compass.

As seen in Figure 5.8 adding the gears described in Section 5.3.1 resulted in moving the wheels 47mm forward, and the group therefore moved the IMU so that it is still mounted between the wheels. In addition to moving the wheels, the wheels themselves were replaced with larger and softer wheels. Softer wheels help with traction, which again minimise error when estimating the position. The new wheels are 82mm in diameter and the gear ratio is 12.5:1.

After the PCB was printed, there were discovered a few mistakes on the board and these were fixed by adding extra wires correcting them. Later in the project period, some pins were moved to other inputs to optimise the board. The LEDs were also added after the PCB was made, and these are connected via the breadboard. Since there were several changes after the PCB was made, a new and updated PCB should be printed. This is further explained in Future Work, Section 12.

During final testing of the robot, the wheel encoder got damaged, resulting in wrong position estimates when the robot is driving. This is the only part that does not work on the robot, and therefore the code that makes the robot perform its commands is commented out. The robot can still be used in cooperation with other robots as the communication and IR scanning works, but the robot will not move.

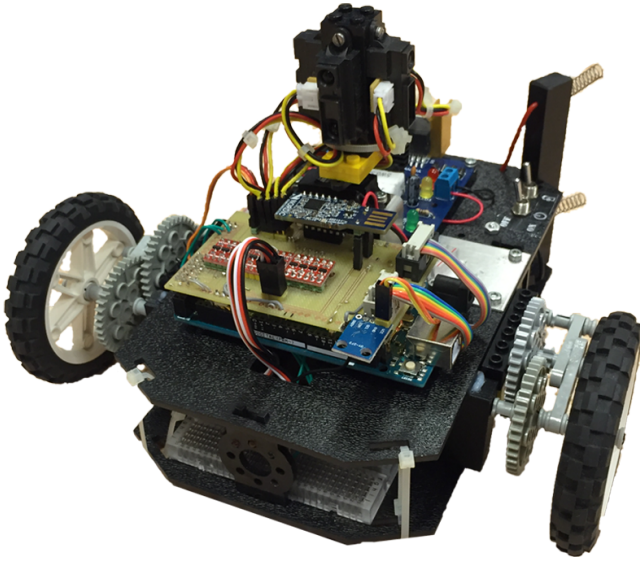


Figure 5.8: The finished Arduino-robot

6 | Wireless Communication

This chapter covers the wireless communication between the robot and the server computer. Section 6.1 and 6.2 provides information about how to program the nRF51-server dongle and how the software was developed, while Section 6.3 describes how the message protocol was created. In the end of the chapter, Section 6.4 presents distance and integrity tests.

6.1 Programming the nRF51-dongle

This section covers the programming of the nRF51-server dongle. For information about the nRF51-peripherals, see Ese (2016).

Dependencies

To program the nRF51-dongles, the following software has to be downloaded and installed:

- nRF51 SDK10[50]
- nRFgo Studio v1.21[51]
- Keil MDK-ARM v5.20[23]

Drivers for the nRF51-dongle comes with nRFgo Studio, it is recommended to install the drivers from nRFgo Studio before inserting the nRF51-dongle in your computer. This because Windows tries to use “Windows Update” to search for the drivers, and as Windows does not find the correct drivers, it will scan indefinitely.

Setting up the nRF51-dongle

Having installed the software listed above, the following procedure has to be followed to flash the dongle and prepare it for programming:

1. Plug in the nRF51-dongle to one of the external USB-ports at the computer.
2. Start nRFgo Studio, and choose the desired device in the “Device Manager”-list.
3. Click “Erase all”.
4. Choose the “Program SoftDevice”-pane in the right panel and click “Browse...”.

5. Navigate to “[yourDestination]/nrf51_sdk10/components/softdevice/s130/hex” and choose the “s130_nrf51_x.x.x_softdevice”.
6. Click “Program”.

6.2 Developing the nRF51-server Software

The nRF51-server should be able to communicate with several nRF51-peripherals. It has to provide functionality for sending messages to the connected peripherals and receiving messages from other connected peripherals at the same time. Since the server and peripherals are tightly coupled, the software on the dongles was developed together with Ese (2016). The nRF51-server source code can be found in Appendix 10.

Nordic Semiconductor, the manufacturer of the dongle, has sample code for communication between two dongles, but this sample code can not be used when several dongles are connected to the server at the same time. There is also another sample code with peripherals and central (server), however in this code, the communication can only go one way. Marco Russi [47] has modified Nordics “central”-program to enable communication both ways. There was a meeting 12th of February together with Andersen, Ese and Nordic Semiconductor. The agenda of the meeting was to get a briefing about how Nordics peripheral- and central-code works, to get more information about the nRF51-dongle and to do some investigation in Marco Russi’s code.

The finished nRF51-server software is a modified version of Russi’s central-software, which again is just a modified version of Nordics central software. Whenever the server-dongle gets a message, it adds an ID to the message before it gets forwarded to the computers COM-port. The message structure becomes: `[robotID]:Message` where “robotID” is the ID the dongle generates for the connected peripheral and “Message” is the message content received from the peripheral through Bluetooth.

The message content from the peripheral has to be embraced by curly brackets and end with a linebreak (`\n`), and the messages cannot contain square brackets. In addition, each message should start with the robot name then a colon before the content.

Example:

Message from peripheral: `Arduino:{U,100,100,90,0,70,65,40,75}\n`
Server adds robotID: `[1]:Arduino:{U,100,100,90,0,70,65,40,75}\n`

Because of a limitation in Bluetooth, there is a cap of 20 bytes per package sent, this

means that if a robot sends a message longer than 20 bytes it gets split into several packages. As of right now, the robot can send messages up to 60 bytes resulting in maximum three packages per message. If more than one robot tries to send messages at the same time, it is possible that the packages get intertwined alternately. The following example illustrates this effect:

Example:

Robot 1 says `Hello beautiful World` and Robot 2 says `Hello`, the result read from the computer's COM-port could be:

```
[1]:Robot1:{Hello beautif [2]:Robot2:{Hello}\n[1]:Robot1:ul World}\n
```

The unravel of the messages happens in the Java applications Communication package, see Section 7.3.

The biggest limitation in the nRF51-server software is that it is not possible to connect more than three robots, since the central can maximum hold three Bluetooth connections at the same time.

6.3 Creating the Message Protocol

When implementing the communication between the physical robots and the server software, the groups realised the need for a general message protocol. The message protocol should be complex enough to consist of different message types and parameters, as well as being simple enough to be easy to understand and implement in software.

The groups sat down and discussed which kind of messages the protocol should implement, and what the messages should include of parameters. The message types and parameters were listed as shown in Table 6.1 and Table 6.2.

At first, the group developed a message protocol using JSON-structure. The benefit of using JSON to structure a message protocol is that it is easy to understand just by reading the message, as well as the order of the parameters does not have to be specified. The update message from server in JSON-structure was defined as:

```
{"Update": [{"Orientation": "60"}, {"Distance": "100"}]}
```

Due to some problems with the nRF51-server software not being able to receive messages with a size greater than 60 bytes, and the nRF51-peripheral not being able to receive messages larger than 20 bytes, the group converted the message protocol from JSON to

Robot	
Message type	Parameters
Handshake	Message type Physical robot width and length Tower offset from center of the robot Axle offset lengthwise Sensor offset from tower center Initial sensor heading
Update	Message type Robot position (x,y) Orientation Tower heading Sensor values
Status	Message type Idle

Table 6.1: Sketched list over messages from the Robot

Server	
Message type	Parameters
Update	Message type Orientation Distance
Status	Hanshake confirmed Pause robot Unpause robot Robot finished

Table 6.2: Sketched list over messages from the Server

a more compact designed structure. The update message (from server) including the same values as the JSON-example, was then defined as:

```
{U, 60, 100}
```

The remaining message types were defined in the same manner. Figure 6.1 illustrates a simple overview of the messages that are sent and received, while Appendix 4 “Message Protocol” presents all message types and parameters implemented in the message protocol. The message handler in the server application discards all messages that does not fit the defined standard (corrupt messages).

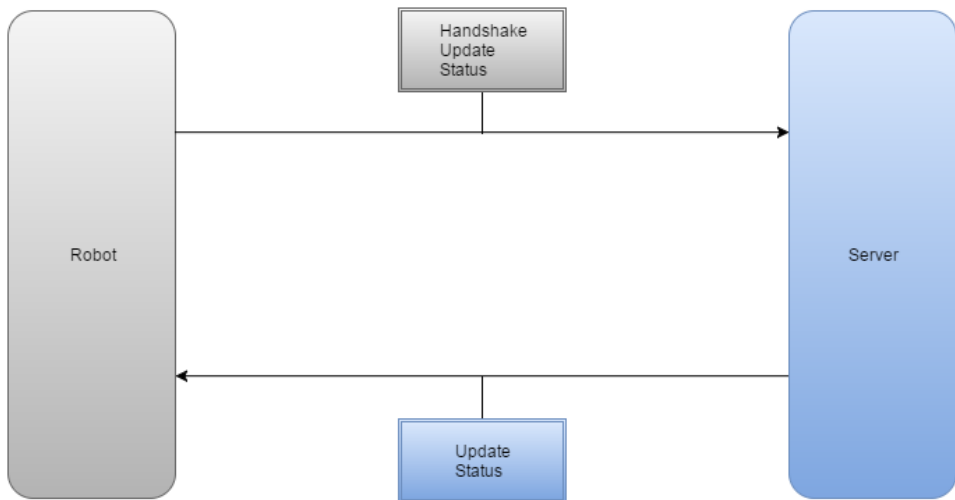


Figure 6.1: Flow of the different message types between robot and server

6.4 Distance and Integrity Tests

The purpose of the test was to determine how far away the robot could move from server while delivering sufficient updates.

Corridor

Environment

The environment was a school corridor with no physical obstacles. The walls, roof and floor was made of concrete, and the size of the corridor was:

- Height: 2.50m
- Width: 2.00m

Data Acquisition

The server computer was placed at the end of the corridor, and the server application was started. When the communication was established between the server and the Arduino-robot, one of the group members lifted the robot up and moved it 5 meters away from the server while the server counted how many messages that contained errors during 1 minute of testing. The procedure was repeated at every 5 meters until the message integrity was too bad to continue.

Test Result

Environment: Corridor		
Distance [meter]	Message error [/min]	Fault percentage [%]
1	0	0
5	3	1,000
10	7	2,333
15	2	0,066
20	10	3,333
25	-	-

Table 6.3: Message Integrity Test in Corridor

Table 6.3 shows the data acquired during the test. The robots sends update messages every 200ms, which means that the fault percentage is:

$$\frac{No.error\ messages}{5messages/second * 60seconds}$$

At 25 meters the robot had problems with sending messages and therefore the test ended. The robot did not loose the connection to the server, but no messages was received. The test results implicates that the robot should never be further apart than 20 meters from the server dongle in a corridor.

Open Landscape

Environment

The environment was the open mingling area “Glassgården” at “Elektrobygget” NTNU. The height and width of the area are so large that it could be counted as open landscape.

Data Acquisition

The same procedure as described in the corridor test was done during this test.

Test Result

Environment: Open landscape		
Distance [meter]	Message error [/min]	Fault percentage [%]
1	0	0
5	5	1,667
10	3	1,000
15	14	4,667
20	-	-

Table 6.4: Message Integrity Test in Open Landscape

Table 6.4 shows the data acquired during the test. The robot had more problems in “Glassgården” than in the corridor. The robot had some problems at 15 meters and at 20 the server did not receive any messages. Likewise as the corridor example the robot did not loose its connection, but could not send messages. The test results implicates that the robot should never be further apart than 15 meters from the server dongle in open landscape.

7 | Server Application

The following chapter describes the process of making the Server Application: How the group defined the scope, designed the system architecture and the real-time system flow. Section 7.3 gives information about the structure of the packages while Section 7.4 describes the system flow. Finally Section 7.5 explains the functionality and Section 7.6 the limitations of the system. Please refer to Appendix 11 for application source code and Appendix 12 for the software program.

7.1 Designing the System Architecture

To start off the system design, the group acquired reports and code from previous projects. By reading the code, testing the functionality and studying the reports, the group got insight and understanding of the existing system. It was important to know the current system before starting to develop the new system, and therefore, the group used two full weeks studying it.

Initially, the group defined the scope of the new system. The system should:

- Inherit the functionality from the existing system.
- Be stable and not introduce bugs.
- Present mapped data in an easily and understandable way.
- Provide a user-friendly GUI.
- Be easy to understand and improve for future projects.

The list of system goals laid a basis for the system named “System for Self-Navigating Autonomous Robots”, henceforth referred to as SSNAR.

One of the primary goals was to develop the system in a way that makes it easy to understand for groups that want to improve the system in future projects. The group had object-oriented programming in mind in every part of the system design process, and the system was divided into subsystems and packages that took care of different areas of responsibility and functionality. Figure 7.1 shows how the group was thinking during the brainstorming, trying to structure the responsibilities into modules.

Subsequently, classes were defined in each module. According to the naming conventions in “Code Complete” [33], see Section 2.5.2, the classes were named according to their responsibilities with self-explanatory names, making them easy to understand without having to read their code. Furthermore, the classes’ routines were defined. All of the routines were designed to do one thing and one thing only, for facilitating high cohesion and low coupling, as described in Section 2.5.2.

Throughout the developing, the group held coordination meetings with both Ese (2016) and Thon (2016) to make sure the system was designed in the best way possible.

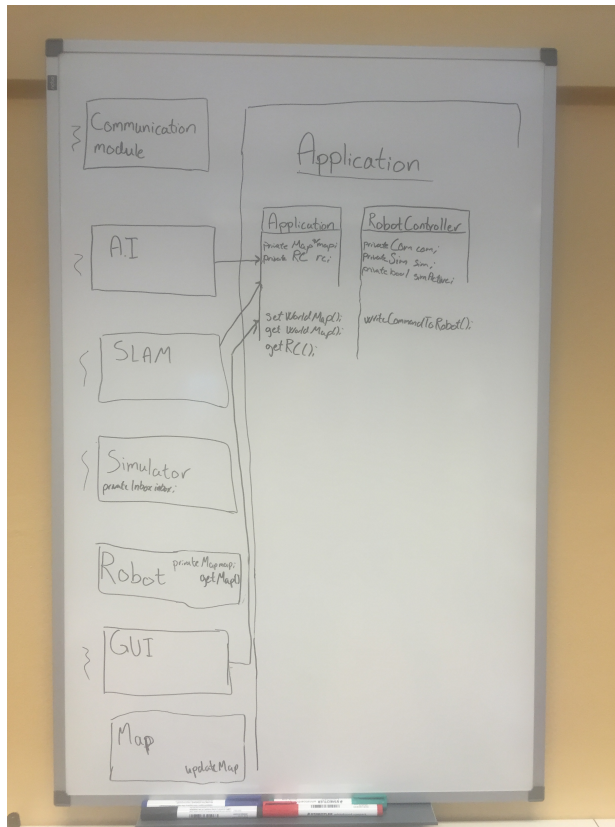


Figure 7.1: Dividing the system into subsystems and packages during project planning phase

7.2 Designing the Real-Time System Flow

The purpose of this year's project was to make the robots able to drive and scan in real-time, not sequentially drive-stop-scan. Therefore, it was important that the server application from the first line of code were developed concerning concurrent programming.

The group sat down with the list of modules and decided which modules that should consist of threads. As seen in Figure 7.1, the modules containing threads are marked with a curly sign at the left of the module. Defining which modules that should include thread-handling in the early phase of the development, was contributing to avoid thread-handling issues like race conditions and deadlocks as described in Section 2.5.1.

In the server application, some of the classes were defined as shared objects. E.g. the class that should contain all incoming messages had to both provide functionality for putting messages into the inbox, as well as being read from by a "mailbox reader". To prevent race conditions in the shared objects the code blocks were synchronised, such that only one thread could access them at the time. `ConcurrentLinkedQueue` is a thread safe list that was used to store measurements, while JavaFX's `observableList` were used to store robot objects.

7.3 Structure

Seven modules form the system, see Figure 7.2, each taking care of one area of responsibility. This thesis covers the four modules "Application", "Communication", "Robot" and "GUI". For information about the three modules "Simulator", "Mapping" and "Navigation", please refer to thesis Thon (2016).

Application

The "Application" module is the main module in the system. The module is the one that ties the system together by being the main artery for communication and program flow internally in the program. "Application" instantiates the other modules and connects the packages in the program.

Robot

The “Robot” module represents the physical robots that the system uses. When connecting to a real-life robot, e.g. the Arduino-robot, an object related to that robot is instantiated in the module. The object holds information about the associated robot and contains measurements received from the robot.

Communication

The “Communication” module represents the link between the physical world and the server application. It takes care of the serial communication with the nRF51-dongle as well as providing message input and output. The module also provides functionality for encoding and decoding messages.

GUI

The “GUI” module provides the applications HCI interface. It presents information about the robots connected to the system and map plotting of the mapped areas, as well as it provides functionality for connecting/disconnecting and controlling the robots.

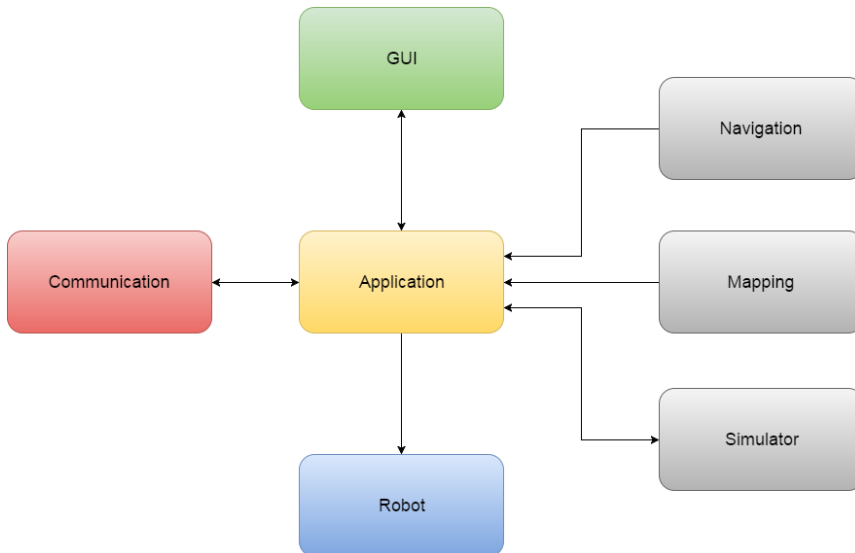


Figure 7.2: Structure of the system modules

7.4 System Flow

To give better insight in the system, the following cases demonstrates how the system flow works during runtime, see Figure 7.3 for reference. For more detailed information about the functionality of the system, please refer to the Javadoc in Appendix 2 and the UML diagrams in Appendix 14.

Case 1: “Update message” received

- When the nRF51-server dongle receives a message from a connected robot, it writes the message to the connected COM-port.
- The class SerialCommunication reads the in-stream, converts the received bytes into Strings and forwards them to the Inbox class.
- The Inbox class simply holds a “Received message” list.
- InboxReader retrieves message batches from Inbox and merges them into complete messages. Furthermore, InboxReader uses MessageHandler to parse the complete message and extracts information about “Sender ID”, “Message type”, and the message-specific information. Finally, the information retrieved gets passed into RobotController.
- RobotController adds the update values as Measurements into the related Robot object.

Case 2: Scanning after robots

- When the user clicks the “Connect robots” button in the GUI, the MainGUI class creates a ConnectRobotsGUI object which handles the connection routines.
- ConnectRobotsGUI starts a scan through a method call for the Application.startScan().
- Application uses Communication to set the nRF51-server dongle to “command mode” and sending a “scan” command to the dongle.
- After 1 second the scan is stopped in the same way it was started, and a “list” command is sent to the dongle.
- When the list of found devices is returned, the in-stream from the nRF51-dongle at the COM-port is handled by SerialCommunication and Inbox the same way as described in “Case 1”.

- InbxDReader interprets the message content (robot IDs and robot names) and passes the found robots to RobotController.
- RobotController adds the discovered devices in an observableArrayList.
- The GUI has a listner on the same list, and when a robot is added to the list it gets listet in “Found Robots” in the GUI.

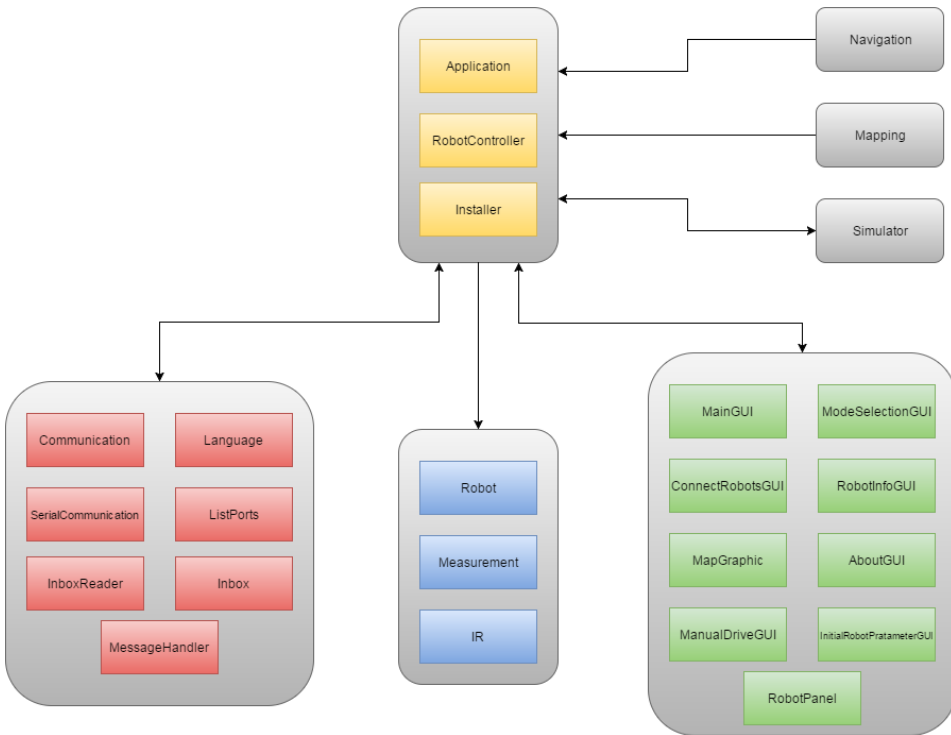


Figure 7.3: Simple class diagram of the Java application

7.5 Functionality

Cross-platform

The server application in its entirety is written in the Java programming language, which makes the software able to run on any device that has Java Virtual Machine. Because Java applications are compiled before runtime, the server application will not be as demanding as applications that are compiled runtime.

Connecting to N-robots

The software is developed in a way that each robot that is connected to the system has one object of the class Robot connected to it. This means that as long as the computer can add another instance of the class Robot, it can connect to more robots.

Real-time mapping and plotting

The robots attached to the system sends update messages every 200ms, and the information extracted from the messages is processed and plotted in the map presented in the GUI, which means that the user will be able to see the mapping progress in real-time. The system also continuously updates the robots in real-time, which means that the system is fully exploiting the functionality of ESE(2016) real-time OS running at the connected robots.

Modular system architecture

The system architecture, as described in Section 7.3, is formed by modules that take care of one area of responsibility each. If future projects adds more modules, or implements more functionality to one of the existing modules, it will be easy to do this simply by using the modules interfaces. The application is already designed to be extended to include Simultaneous Localization and Mapping (SLAM) and not just mapping, as implemented in the present version.

7.6 Limitations

Disconnecting before connecting

Due to an indexing problem in the nRF51-dongle, it was needed to constrain the progress of connecting to additional robots. Before initializing a scan for new robots, all of the robots that are attached to the system has to be disconnected. The reason is that when the nRF51 scans and discovers new unattached robots, it indexes them from 0 and up, even though there already were robots connected from before the scan. This will lead to several robots with the same ID (e.g. two robots with ID = 1), which is undesirable.

Connecting to more than three robots in real world, and ten in the simulated world

Even though the server application is designed to be capable of connecting to n-robots, a constraint regarding the maximum connected robots was implemented. As explained in Section 6.2, the nRF51-dongle has a limitation of maximum three peripherals connected at the same time. This resulted in no need for n-robots connected, and therefore the software is constrained to a maximum of three robots during real world mode, and a maximum of ten robots during simulated world mode. It is easy to change the limit in the application, but as of right now the application is limited to checking one digit in the ID parameter of the messages, as the nRF51 only sends one digit IDs.

8 | Graphical User Interface

The following chapter provides information about how the Graphical User Interface was designed and developed. Section 8.1 covers the design process, Section 8.2 the programming of the GUI, while Section 8.3 presents the final product.

8.1 Design Process

The design process was divided into five phases, and each phase dealt with the various key elements in a design process, as described in Section 2.4.1.

8.1.1 Sense the Gap

To sense the gap, the group studied earlier projects to get an insight in the previous solutions to the problem. The group discovered several shortcomings in the existing solution and got an understanding of the gap in the user experience. The group draughted a list of criteria (see list below) that the final product should meet, and it made the foundation for further process. The new design should:

- Be reliable.
- Be easy to use.
- Minimise the likelihood of user error.

8.1.2 Understand and Refine the Problem

Further, the group defined the problems of the existing solutions and its shortcomings. “System for Self-Navigating Autonomous Robots” is a project that has been performed several times, but the system is no longer working. This year’s thesis’ main objective is to produce a better solution that will last significantly longer than the previous solutions. The group explored the existing solution to understand why it did not work as expected and tried to connect the findings to the three criteria defined in the “Sense the gap” phase.

Reliability

The existing solution was not perfect regarding reliability. During testing of the system the user often experienced that the actions not always led to the expected behaviour. Many of the button's functionality had been removed, but the buttons were still present in the GUI. With this in mind, the group designed an interface that consisted of a minimal set of functionality that met the requirements for functionality, as well as removing redundant components.

Usability

The existing solution had a GUI that contained lots of information and functionality. Many of the buttons were connected without any information about how they were related, some of the buttons were representing the same actions, and some buttons did not work at all. The buttons, together with many boxes presenting information, made the user interface complex and bad in terms of usability. The group used the existing solution to define guidelines for how the new system should be, considering usability.

Minimise the likelihood of user error

Putting all of the functionality in the same window makes it more likely for the user to make mistakes. Dividing functionality into sections and making the program as a sequence of actions prevents user errors, and makes the system appear more usable and reliable for the user.

8.1.3 Explore

Based on the information in the specification, the exploration phase started. The group members developed various suggestions for the GUI with the specification as basis, but also took the Gestalt principles (Section 2.4.2) and Schneiderman's 8 golden rules (Section 2.4.3) into account.

Before developing design solutions the group made a list consisting of all the functionality that the GUI should provide (see Table 8.1).

Maps	World
Robot information	Robot name Position Orientation Tower angle
Indicators	Simulator on/off Number of robots connected
Buttons	Simulator on/off Manual drive/Autonomous Connect to /disconnect robots Start/Stop

Table 8.1: Criteria for functionality in the GUI

Graphical User Interface, suggestion 1

The suggestion in Figure 8.1, has a start-up screen where the user selects either to run the simulator or to run the system in the real world. Choosing the mode brings the user to the main screen. In the main screen, the information about the robot is presented as well as a local/world map toggled by a radio button group. To add more robots the user has to click on the plus sign (new pane), and a pop-up window presents available robots.

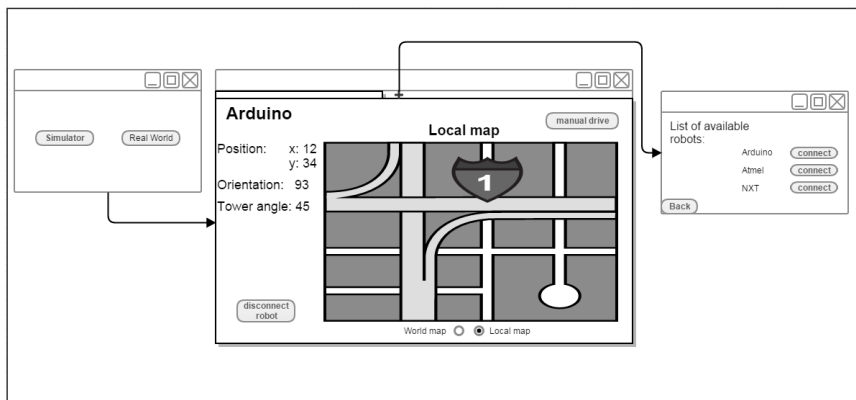


Figure 8.1: Graphical User Interface, suggestion 1

Graphical User Interface, suggestion 2

The suggestion illustrated in Figure 8.2 has, like the first one, a start-up screen that presents two different modes (simulator and real world). Choosing the mode brings the user to a new dialogue box where the user is presented with another two options: manual drive or autonomous. Clicking on the desired option brings the user to the main screen. The main screen consists of a horizontal list of the connected robots and their information, as well as a map that can switch between different modes. To add, a robot the user has to click “Add robot”, and he/she is presented available robots in a pop-up.

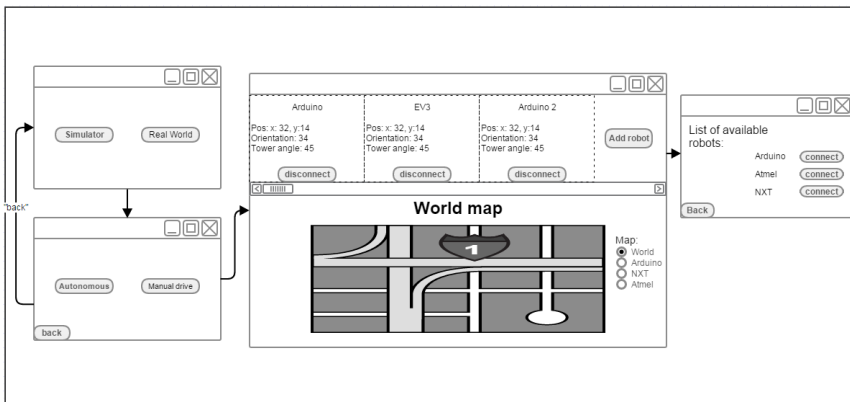


Figure 8.2: Graphical User Interface, suggestion 2

Graphical User Interface, suggestion 3

The suggestion in Figure 8.3 does not have a start-up screen. The main screen presents the two different modes in the top left corner, and the rest of the functionality will be available after selecting a mode and pressing start. Further, the main screen contains a list of connected robots and a world map. Clicking “connect to more robots” presents a pop-up similar to the other solutions while clicking “more” (buttons next to connected robots) opens the robot screen. The robot screen presents information about the specific robot as well as an opportunity to enter manual drive mode for the robot.

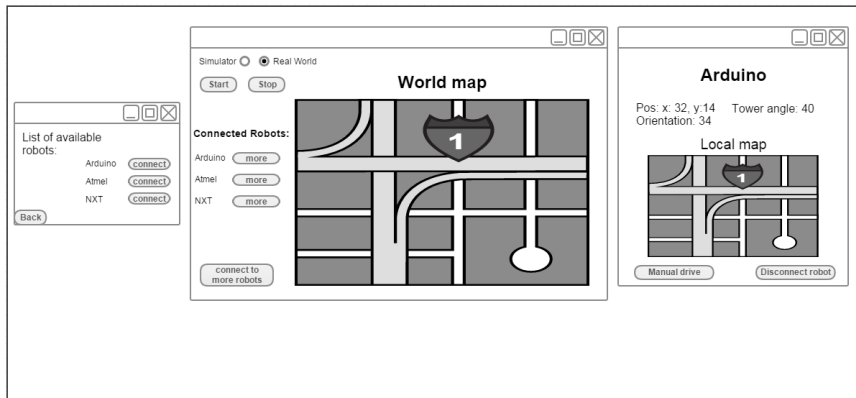


Figure 8.3: Graphical User Interface, suggestion 3

8.1.4 Evaluation and Feedback

To evaluate the different suggestions, the group printed the GUI suggestions on sheets and arranged usability testing to get user-experienced feedback. The different screens of the interface were covered using sticky notes, but these were removed as the test person “navigated” through the GUI. The test person was given some pre-defined tasks to perform, and during the interaction he/she was not receiving any help. After performing a set of tasks, the person was handed an evaluation sheet, where he/she evaluated the different parts of the GUI.

The results from the user testing are shown in Table 8.2. In addition to rate the different suggestions, the test persons got to comment them as well.

Suggestion	Comments	Rating
1	<ul style="list-style-type: none"> - A solution using panes to represent each robots gets complex if many robots are connected. - It is tiring having to switch between panes to get info about different robots. 	2nd
2	<ul style="list-style-type: none"> - The main screen presents almost too much information. - Choosing modes before entering the main screen can help prevent user errors. 	3rd
3	<ul style="list-style-type: none"> - It is straightforward to present the information about the respective robots in a separate screen. - Less is more. 	1st

Table 8.2: User feedback regarding GUI Design

8.1.5 Select Plan

This phase focuses on choosing a plan for the design, based on the results from the “Evaluation and feedback” phase. The group discussed the results and evaluated the suggestions based on their experience, before selecting the plan.

The design that was chosen consists of functionality and design parts from all of the three suggestions. The mode selection from suggestion 1 and 2, see Figure 8.2, was included to prevent user errors that may have occurred if the selection was present in the main window. The three window solution in suggestion 3 laid the basis for the design of the main windows in the application. Some minor changes, such as moving the “Start” and “Stop” buttons and removing the “Local map”, as well as adding a menu bar at the top of the main window were made.

8.2 Programming the Graphical User Interface

The GUI was programmed and designed in NetBeans (Section 3.1), since NetBeans offers a very intuitive GUI builder for Java applications. NetBeans generates a “.form”-file in addition to the normal “.java”file, and the “.form”-file is encoded in XML which contains all parameters the GUI-builder needs. When designing a GUI in NetBeans the user do not need to worry about the “.form”-file, as NetBeans does all the work. NetBeans will automatically recognise a NetBeans project and then load “.java”- and “.form”-files together, because of this different users can easily modify an existing project.

The GUI is initialised by calling *new MainGUI(this)* from the Application class, and MainGUI has the responsibility of initialising the rest of the GUI. There is a total of nine Java classes in the GUI package, where eight of them are extending JFrame and one extending JPanel. Detailed information about the classes can be found in the Javadoc, see Appendix 2.

8.3 The Final Product

The final product can be viewed in Figure 8.4, which shows the main window of the application. In this figure, there are two simulated robots (Arduino and Atmel) driving around in a simulated world. The GUI that controls the simulation can be opened by

pressing the menu “Window” and then “Simulator”. For more information about the Simulator GUI, please refer to Thon (2016).

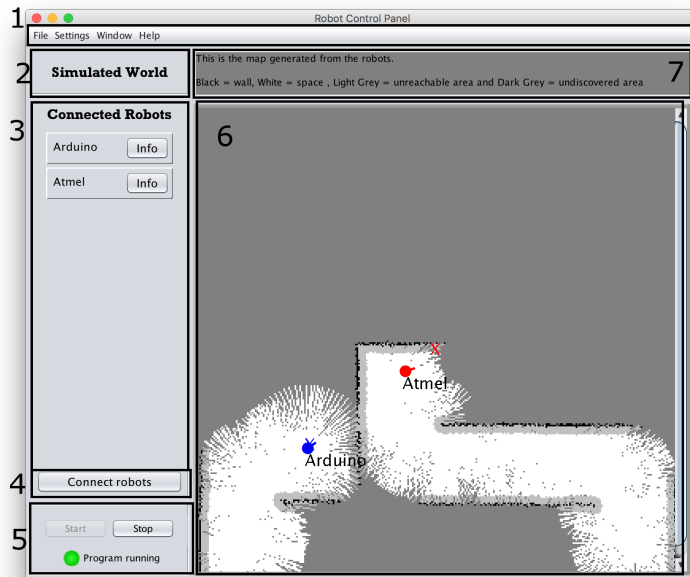


Figure 8.4: GUI, Main window

The main GUI is divided into seven parts as illustrated by Figure 8.4. Objects that have similar functions are grouped together, and the different groups are separated by borders. The GUI is created with the Gestalt Principles (Section 2.4.2) and Schneiderman’s 8 golden rules (Section 2.4.3) in mind, and if anything is unclear the user can press the menu “Help” and then “User manual”, which will open the user manual in the default PDF viewer. The user manual can be found in Appendix 15, and it provides information about every function in the user interface.

In the following list, the parts of the main window are described:

1. Menu bar - The menu provides mode-specific functionality, as well as a help menu where the user manual can be found.
2. Mode title - This panel tells you which mode the program is currently running in.
3. Connected Robots - This panel will be updated with the connected robots. Clicking the “Info” button will open up the Robot info window.

4. Connect robots - This button will open up the “Connect Robots” window.
5. Start/Stop - The start and stop buttons will start and stop the application. The indicator will turn green if the program is running.
6. Map - The map will be painted in real-time as the robots discover new areas.
7. Info - The information panel will display informative text as the mouse cursor hovers over the different parts of the GUI.

Figure 8.5 illustrates the navigation through the different parts of the final GUI. After choosing a mode, the mode-specific window opens. Clicking “Connect robots” brings up the related window, and as the scan finishes, the available robots gets listed. After selecting the desired robot(s) and clicking “Connect”, the communication with the robot(s) is established the server receives a handshake message from the robot(s) confirming the connection. Furthermore, the user sets the initial values for the specific robots, and after the values are set, clicking “Start” will initialise the mapping.

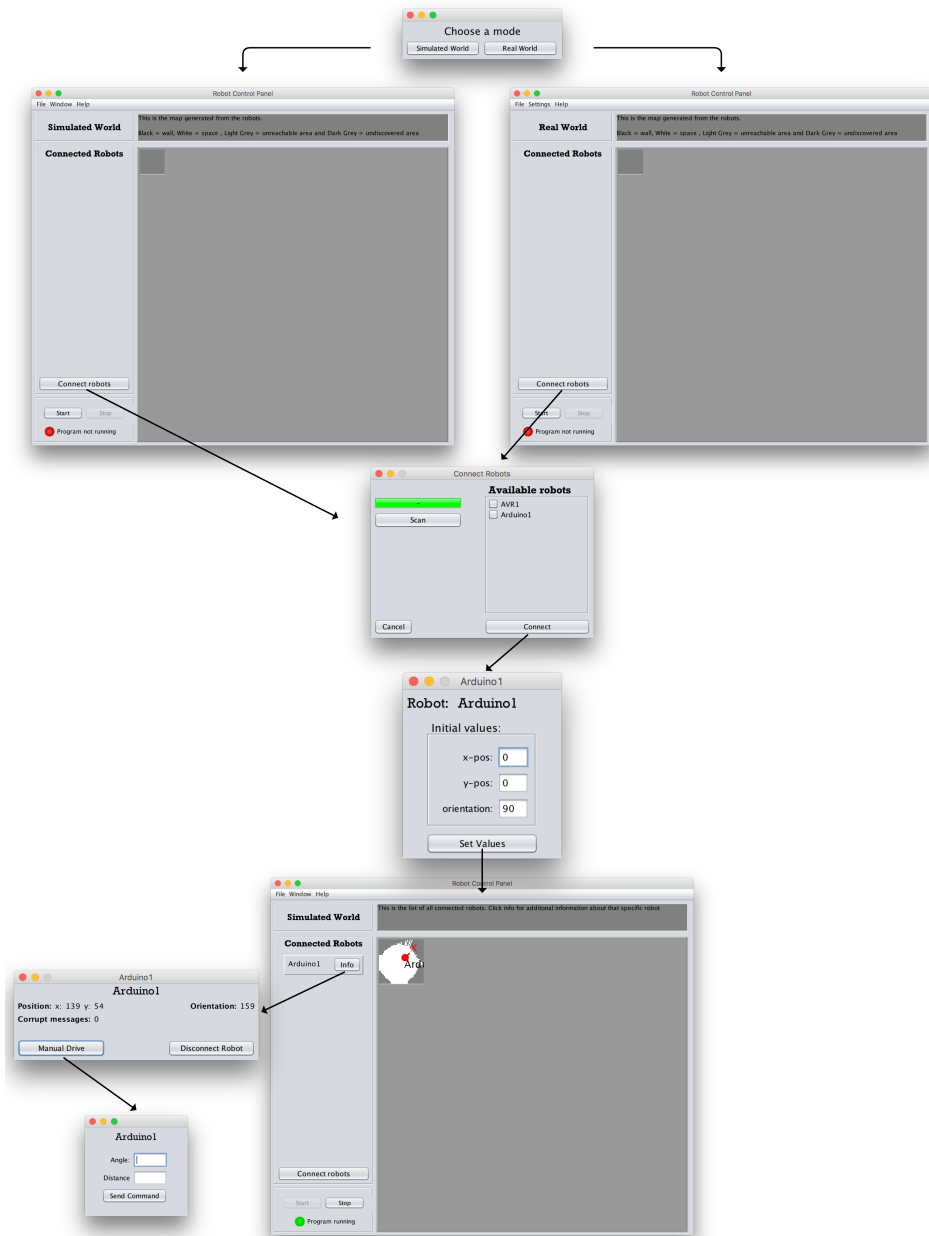


Figure 8.5: GUI Flow

9 | Testing the System

In addition to the communication tests in Section 6.4, the groups arranged a full-system test when all parts of the system were finished and implemented. A maze was built, see Figure 9.1, and both the Arduino and the AVR-robot was placed in it. As it is important for the application to know how the robots are positioned relative to each other, the distance between them was measured.

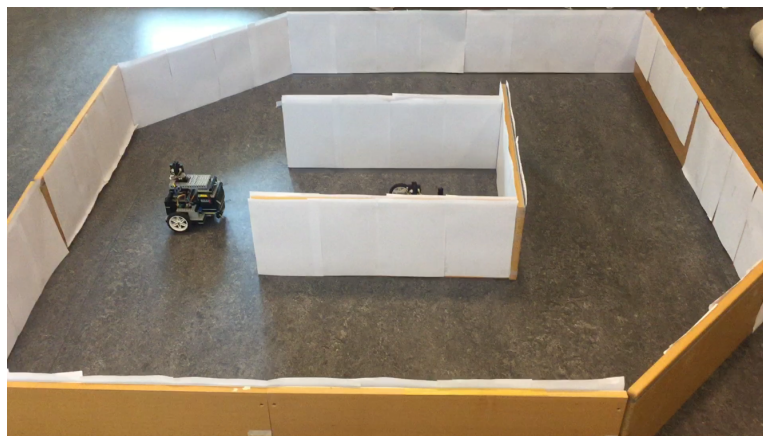


Figure 9.1: Maze used during the testing of the system

As the Arduino-robot's wheel encoder was broken, as elaborated in 5.4, the robot was placed in the center of the maze. Even though it could not perform the orders given to it by the server application, it performed scans and sent update messages during the test. The groups performed three tests to validate that the system could provide consistent results.

Figure 9.2 presents the maps created by the three tests, as well as including a fourth mapping done during the demonstration of the project. As can be seen, the four tests gave results that are similar, but none of them are exact replicas of the real world maze. The reason for this is that the position estimation diverges over time. Let's say that a robot starts in the top left corner and moves one lap around the maze. The wall that is first drawn and the wall that is last drawn during the mapping may not be perfectly aligned, even though they represents the exact same wall in the real world.

Appendix 3 "Media" contains videos and images from the tests.

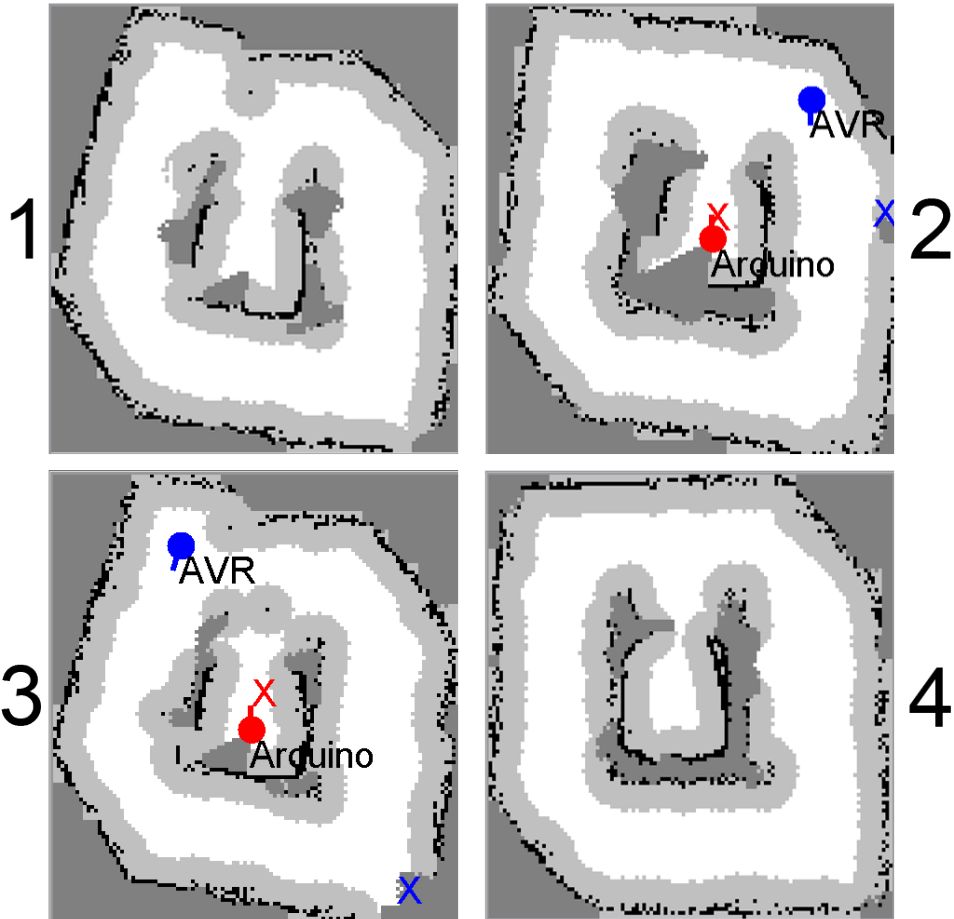


Figure 9.2: Four map plots of the same maze

PART III:
DISCUSSION, CONCLUSION AND FUTURE
WORK

10 | Discussion

10.1 Arduino-Robot

The Arduino-robot is built with the same specifications as the AVR-robot; it has the same wheels, gears, IR sensors, compass, gyroscope and the same charging mechanism. What differs from the AVR-robot is the wheel encoders, the motor controller, the motors, the ability to turn off the charging towers and the Arduino microcontroller with the self-made PCB shield.

Using an Arduino as microcomputer has several benefits, it is easy to connect other components to its circuit, and by creating a shield it gets even better. This makes the robot easy to modify if the robot is to be extended, for example, with new sensors in the years to come. It is also straightforward to upload new C code by following the steps in Section 5.2.2.

Arduino has a 5V output, but as the output could not provide power to the whole circuit, the group had to use the 5V from the motor controller instead. The motor controller can deliver up to 2A at 5V in addition to powering the motors, and therefore it has no problem powering the entire robot.

At this point, everything but one wheel encoder works on the Arduino-robot. The wheel encoder got damaged during the last week of the project period. It is very important that the wheel encoder is placed 2-3 mm away from the magnet, and that it does not touch the magnet wheel. Due to the wheel encoder not working, the robot can not estimate its position. The robot has a compass, gyroscope and accelerometer all working, but the software is not optimised to use input from them. For more information about the software, see Ese (2016).

10.2 Using Bluetooth as Communication Protocol

In the early phase of the project, the groups discussed which communication protocol to use. The two protocols WIFI and Bluetooth early excelled as alternatives. The groups had programming experience with WIFI from previous school projects, but no experience with Bluetooth. One of the main advantages using WIFI as communication protocol

is that it is possible to communicate wirelessly without being close to the robot (e.g. controlling a robot, placed at the school, over Internet while sitting on a computer back home). The main disadvantage using WIFI was that components such as WIFI-shields and dongles often have a significant power consumption.

When it comes to Bluetooth, the disadvantage was the lack of experience. Earlier projects on this topic used Bluetooth as communication, so it was in keeping with tradition to choose this protocol. As well as the lack of experience, the communication protocol itself does not provide long-distance communication, as the range is limited. The advantages using Bluetooth is that it is available on most devices (tablets, computers, phones), it is suited for short-range on-site communication and it offers the Bluetooth Smart version that has a very low power consumption.

According to Bluetooth SIG [29] it is planned that Bluetooth is increasing its internet of things functionality by supporting mesh-networking during 2016. Mesh-network means that not all of the connected devices has to communicate directly to the server, but all of the devices talks to each other to share the network connection across a large area [25]. Mesh-networking means that the range can significantly be extended to be greater than 15m (as measured in Section 6.4).

10.3 Using nRF51 as Communication Unit

The nRF51-dongle was chosen as communication unit after choosing Bluetooth as the communication protocol. The dongle is developed by Nordic Semiconductor; a company that originated from NTNU.

The nRF51 supports Bluetooth Smart as well as 2.4GHz proprietary applications [49]. Due to the Bluetooth Smart technology, it has a low power consumption which suits the use in this thesis. As it will support Mesh-networking when Bluetooth supports it, more functionality can be added to the system in the future. The dongle was relatively cheap (150 NOK), and was available for the groups at “Omega Verksted” at NTNU.

One of the main advantages with the nRF51-dongle is that the SDK, available at Nordic Semiconductors website [50], provides several example codes for different use of the dongle. This made it easier to understand the functionality the device provides. As well as providing example code, Nordic Semiconductor answered on emails and arranged a meeting with the groups, which gave the group a better insight in the device.

Since the software on the nRF51 has a limitation on the capacity of connected devices, it is only possible to connect to three robots at the same time. At this point, there are only two robots available, so the limitation does not have any effect. However, if more robots are to be added in the future, this could introduce a problem. In addition to the maximum device limitation, the software limits the message size of the message protocol.

10.4 The Message Protocol

During the project, the teams developed a message protocol that laid the basis for all communication between the robots and the server. Initially, the message protocol used JSON-structure, which is an easy-to-read structure. The disadvantage using JSON to structure the messages in the protocol is that it adds a lot of extra characters to the message (curly brackets and colons) that works as parameter dividers. Due to the message length constraint in the nRF51-software, the groups had to move away from using JSON.

Developing a more compact message protocol made it possible to add more parameters to a single message, while making sure the message size was as small as possible. The message protocol provides all the message types required for the system to work, as well as it lays a basis for how to structure additional messages if future projects want to add more functionality to the system. One of the drawbacks using a compact message protocol is that if the user intends to study the messages but does not have the definition of the message content, it will be difficult to understand what the parameters in the message means.

10.5 Graphical User Interface

The design process is a vital part of developing a system that involves Human-Computer Interaction (HCI), and therefore, the group has let the design process form the basis for developing the Graphical User Interface. Through the process of designing the GUI, the group has defined problems, explored alternatives and found a solution that the group believes is the best solution to the initial design problem, see Section 8.1.1 and 8.1.2.

Especially has the Gestalt Principles, Section 2.4.2, and the eight golden rules of Schneiderman, Section 2.4.3, been in focus during the design of the layout. With the Gestalt

Principle “Proximity” in mind, the GUI is designed in a way that objects placed next to each other are connected, as well as linked to the same functions. Figure 10.1 shows the “Start - Stop” panel from the final product. The panel demonstrates how proximity is used as a tool to connect functionality of two buttons as well as an indicator lamp and a descriptive text. According to the principle “Common Fate”, all of the objects that are connected are affected when interacting with one of them. Let’s say the stop button is clicked; The system stops, the clicked button gets disabled, the “Start” button gets enabled, the indicator lamp turns red as well as the descriptive text changes to “Program not running”.

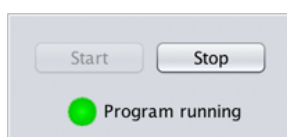


Figure 10.1: Start-Stop panel from GUI, Main window

The layout of the main window is similar whether the user chooses to run the “Simulator” or the “Real World” version of the application. This is a choice made taking Schneiderman’s rule about “Consistency” into account. The user feels that the user interface is complete and recognisable, which leads to better usability. According to Schneiderman’s rule number three, all actions leads to something in the GUI. As tasks are being done, the application responds by showing progress bars and information in the “Information Panel”, see panel 7 in Figure 8.4.

As the examples above illustrates, the GUI follows different rules and principles making the interface more usable for both experienced and new users.

10.6 Selecting the Programming Platform

The group had to choose programming language early in the project period. The members had experience with several languages, such as Java, C, Google Go and Python from earlier school projects. Choosing a suitable programming platform to develop the system was vital, and the group had to take several factors into account when selecting one to go forth with.

The existing system was written using MATLAB and Simulink. The program consisted of several “.m”-files and included different toolboxes, such as CAS¹ for implementation of

¹CAS Robot Navigation Toolbox

SLAM functionality. In addition to requiring a bit of processing power and memory from the computer, MATLAB is not suited for developing software that should provide GUI and real-time functionality.

Java, as an object-oriented language, had many features that suited the system development. Using packages and classes made it easy to divide the system into modules with different responsibilities, and this allowed for a clear and neat structure of the application. As software developed in Java is built and compiled before running, as well as being able to be run on every computer that has the Java Virtual Machine (JVM), the software program is cross-platform and lightweight.

11 | Conclusion

During the Master Thesis, the group has developed a server application and a Graphical User Interface in Java, software for an nRF51-dongle in C, as well as built an Arduino-robot. The different parts of the project seems to be good solutions to the problem description the group made at project start.

Choosing Java as programming platform facilitated a clear structure of the code and provided functionality for dividing the system into modules. The server application is developed in terms of the guidelines in “Code Complete”, and is structured in a way that makes it easy for future projects to understand and further develop the software by adding new functionality. The code is generalised, so that it is possible to add robots with different specifications than the ones already existing, e.g. a robot with more than four infrared sensors. The server application inherits most of the functionality implemented in previous solutions, however, in our implementation the system is processing data and controlling the connected robots in real-time.

The user interface is developed during a design process where the human perception has been in focus. The process has helped the group focus on the aspects that are important when designing a graphical user interface, and the final product inherits most of the functionality provided in the previous solutions, but presents them in a more user-friendly way.

Choosing Bluetooth Smart as communication protocol enabled fast, short-range and low power consuming communication. The protocol was not known to the groups at beforehand, but by studying the standard, the groups managed to implement it into the system. The server-dongle receives and processes five messages per second from each robot connected, and delivers update and status messages to the robot dependent on the server application state. The communication range suited for the system is less than 15 meters between the server and the connected robots, this to ensure that all messages are received.

The group has built a new robot, increasing the existing collection of robots. The robot uses an Arduino as the microcomputer and it is built using the same wheels, gears, infrared sensors, tower and inertial measurement unit as the AVR-robot. Using AVR-dude enabled the Arduino-robot to use the same real-time operating system as running on the AVR-robot, without having to rewrite too much of the code. One of the wheel

encoders got damaged during the last week of the project, and therefore it cannot estimate its position at this point in time. Other than the broken encoder, the robot works as intended and is therefore successfully built and implemented.

During the project period, the group has gained useful experience regarding project management and implementation. As “System for Self-Navigating Autonomous Robots” is a part of a system consisting of three dependent projects, the key to success has been communication and cooperation. Regular conversations and meetings with both the employer and the other groups working with the system have enabled effective solving of any changes to the project tasks. Using Git as version control tool for the developing of the software made it possible for several groups to work on the same application.

Compared to the scope of the project, the group is pleased with the technical result. The solution meets all the requirements of technical functionality and the solution seems to be a modernised solution compared to the prior implementations of the system. Updating the communication standard from Bluetooth to Bluetooth Smart, rebuilding the server application from MATLAB to Java, and building a new robot based on an Arduino microcomputer can all be said to extend the system in a provident way.

12 | Future Work

Create an Updated PCB Shield

As described in Section 5.4, there were made some changes to the robot after the PCB shield was printed. This resulted in that the group had to change some of the wiring on the robot by manually soldering extra wires onto the board.

Updating the Eagle files in Appendix 16 with the changes done on the shield, printing a new PCB, and then solder the board is a work that should be done. In Appendix 16 there is a wiring diagram of the robot as-built, the new Eagle schematic should be updated with the same connections.

Improving the position estimates

As described in Section 5.4, the Arduino-robot's wheel encoder got damaged during the final testing; this is the only part that does not work on the robot. It may be possible to repair the encoder, but if it is not possible, a new one can be bought from SparkFun[17]. The software code on the robot that reads the encoder should also be looked at, as a better position estimate can be found if the robot uses both the compass, the accelerometer and the gyroscope in addition to the wheel encoders.

Communication

As described in Section 10.3, the nRF51-dongle is limited to maximum three connected robots. If it is desired to extend the robot project with more robots, it is necessary to find a better solution than the current server and peripheral code.

The message protocol described in Section 6.3 may also be extended to include additional messages, e.g. "Wall hit" and "Battery low" statuses. If this is desired, the code in the Communication package in the Java application should be modified to include these messages.

Java Application

The Java server application is designed to be a solid foundation for future projects. If the communication gets changed in the future, only the communication package will be affected. Hence, only classes inside the communication package should be changed.

The application is ready for future improvements like an improved Simulator, Artificial Intelligence or SLAM.

Acronyms

- AI** Artificial Intelligence. 82
- API** Application Programming Interface. 14, 82
- CAD** Computer-Aided Design. 25, 26, 82
- COM** Communication port. 38, 46, 47, 57, 82
- GPIO** General-purpose input/output. 10, 36, 82
- GUI** Graphical User Interface. i, v, xiv, xv, 4–6, 16, 17, 53, 57, 58, 62–67, 75–77, 79, 82
- HCI** Human-Computer Interaction. 7, 56, 75, 82
- I/O** Input/Output. 9, 82
- IDE** Integrated Development Environment. 14, 82
- IMU** Inertial Measurement Unit. v, 10, 34, 36, 43, 79, 82
- IR** Infrared. v, 8, 28, 34, 44, 73, 79, 82
- JVM** Java Virtual Machine. 13, 19, 59, 77, 82
- LED** Light-Emitting Diode. 34, 43, 82
- MEMS** Microelectromechanical systems. 10, 82
- OS** Operating System. v, 4–6, 59, 79, 82
- PCB** Printed Circuit Board. 25, 33, 41–43, 73, 81, 82
- PWM** Pulse-Width Modulation. 27, 40, 82
- SIG** Special Interest Group. 12, 74, 82
- SLAM** Simultaneous Localization and Mapping. 59, 77, 81, 82
- SPI** Serial Peripheral Interface. 12, 36, 82
- SSNAR** System for Self-Navigating Autonomous Robots. vi, 5, 53, 61, 80, 82
- UART** Universal Asynchronous Receiver/Transmitter. 11, 36, 82
- UML** Unified Modelling Language. 82

Bibliography

- [1] Arduino. *Arduino Build Process*.
<https://www.arduino.cc/en/Hacking/BuildProcess>. Apr. 2016.
- [2] Arduino. *Arduino MEGA 2560*.
<https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. Apr. 2016.
- [3] Arduino. *Arduino Products*. <https://www.arduino.cc/en/Main/Products>.
Apr. 2016.
- [4] Arduino. *ATmega2560-Arduino Pin Mapping*.
<https://www.arduino.cc/en/Hacking/PinMapping2560>. Apr. 2016.
- [5] Arduino. *What is Arduino?* <http://www.arduino.cc/en/Guide/Introduction>.
Apr. 2016.
- [6] Atlassian. *Source tree app*. <https://www.sourcetreeapp.com>. May 2016.
- [7] Catherine Plaisant Ben Schneiderman.
Designing the User Interface: Strategies for Effective Human-Computer Interaction.
first. Addison-Wesley Publishing Company, 2010.
- [8] Dennis M. Ritchie Brian W. Kernighan. *The C Programming Language*. second.
ISBN: 0-13-110370. Prentice Hall, 1988.
- [9] Adam Carlson. *COUPLING and COHESION*. <http://courses.cs.washington.edu/courses/cse403/96sp/coupling-cohesion.html>.
Computer Science & Engineering University of Washington. Apr. 1996.
- [10] NetBeans Community. *NetBeans IDE*. <https://netbeans.org/>. Apr. 2016.
- [11] Corelis. *SPI Interface*. http://www.corelis.com/education/SPI_Tutorial.htm.
Apr. 2016.
- [12] Atmel Corporation. *Atmel Studio*.
<http://www.atmel.com/tools/ATMELSTUDIO.aspx>. Apr. 2016.
- [13] CSR. *Bluetooth Low Energy*. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=227336.
Mar. 2010.
- [14] A. Prathap Reddi C.S.R. Prabhu.
Bluetooth Technology and Its Applications with Java and J2ME. first.
ISBN: 81-203-2443-9. Prentice Hall, 2004.

BIBLIOGRAPHY

- [15] Elfa Distrelec. *Litiumbatteri*. <https://www.elfadistrelec.no/no/litiumbatteri-11-4600-mah-hy-line-h2b181/p/16901653>. Jan. 2016.
- [16] Allen B. Downey. *The Little Book of Semaphores*. second. The Free Software Foundation, 2008.
- [17] SparkFun Electronics. *SparkFun*. <https://www.sparkfun.com/>. Apr. 2016.
- [18] Sparkfun Electronics. *Serial Communication*. <https://learn.sparkfun.com/tutorials/serial-communication/uart>. 2016.
- [19] Sparkfun Electronics. *Serial Peripheral Interface (SPI)*. <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi>. 2016.
- [20] Elprolab. *Elektronikk og prototypelaboratoriet*. <http://intern.iet.ntnu.no/elprolab/index>. Apr. 2016.
- [21] ePanorama.net. *Serial Buses*. <http://www.epanorama.net/links/serialbus.html>. 2011.
- [22] Google Code Archive. *Arduino*. <https://code.google.com/archive/p/arduino/>. 2016.
- [23] ARM Group. *ARMKeil Microcontroller Tools*. <https://www.keil.com/download/product/>. Apr. 2016.
- [24] ARM Group. *μVision IDE*. <http://www2.keil.com/mdk5/uvision/>. Apr. 2016.
- [25] HowStuffWorks. *How Wireless Mesh Networks Work*. <http://computer.howstuffworks.com/how-wireless-mesh-networks-work.htm>. June 2007.
- [26] Bentley Systems Inc. *MicroStation*. <https://www.bentley.com/en/products/brands/microstation>. Apr. 2016.
- [27] Bluetooth SIG Inc. *Bluetooth*. <https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth>. 2016.
- [28] Bluetooth SIG Inc. *Bluetooth Smart(Low Energy) Technology*. <https://developer.bluetooth.org/TechnologyOverview/Pages/BLE.aspx>.
- [29] Bluetooth SIG Inc. *Bluetooth technology to gain longer range faster speed mesh networking in 2016*. <https://www.bluetooth.com/news/pressreleases/2015/11/11/bluetooth-technology-to-gain-longer-range-faster-speed-mesh-networking-in-2016>. Nov. 2015.

- [30] Jeff Johnson. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. first. Morgan Kaufmann, 2010.
- [31] Margaret Rouse Ken Conway Michael DeHaan.
UART (Universal Asynchronous Receiver/Transmitter).
<http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter>. Feb. 2011.
- [32] Mathworks. *MATLAB*. <http://se.mathworks.com/help/matlab/>. Apr. 2016.
- [33] Steve McConnell. *Code Complete*. second. ISBN: 0-7356-1967-0. Microsoft Press, 2004.
- [34] Microsoft. *Microsoft Project*.
<https://products.office.com/nb-no/Project/project-for-office-365>. Apr. 2016.
- [35] Sun Microsystems. *Starvation and Deadlock*. <http://www.math.uni-hamburg.de/doc/java/tutorial/essential/threads/deadlock.html>. 2005.
- [36] Simon Monk. *If the Servo Misbehaves*. <https://learn.adafruit.com/adafruit-arduino-lesson-14-servo-motors/if-the-servo-misbehaves>. May 2015.
- [37] Dennis M.Ritchie. *The Development of the C Language**.
<http://www.bell-labs.com/usr/dmr/www/chist.html>. 2003.
- [38] Oracle. *Compiling Java*. "http://docs.oracle.com/javase/tutorial/figures/getStarted/getStarted-compiler.gif. Apr. 2016.
- [39] Oracle. *The Java Language Environment*.
<http://www.oracle.com/technetwork/java/intro-141325.html>. 2016.
- [40] Circuits Planet. *Infrared Sensors*.
<http://circuitsplanet.blogspot.no/2014/12/interfacing-arduino-with-ir-sensor-and.html>. Dec. 2014.
- [41] Bluetooth Report.
Bluetooth versions comparison. What's the difference between the versions?
<http://bluetoothreport.com/bluetooth-versions-comparison-whats-the-difference-between-the-versions/>. 2013.
- [42] Jr Robert S. Cartwright. *Deadlock*.
<https://www.cs.rice.edu/~cork/book/node98.html>. Jan. 2000.
- [43] Jr Robert S. Cartwright. *What is Concurrent Programming?*
<https://www.cs.rice.edu/~cork/book/node96.html>. Jan. 2000.

BIBLIOGRAPHY

- [44] Center for Robot-Assisted Search and Rescue. *The Taiwan Rescue: Robots*. <http://crasar.org/2016/02/05/the-taiwan-earthquake-robots/>. Feb. 2016.
- [45] Society of robots. *Infrared vs. Ultrasonic*. http://www.societyofrobots.com/member_tutorials/book/export/html/71. Apr. 2016.
- [46] Kevin Ross. *Bypass Capacitors*. <http://www.seattlerobotics.org/encoder/jun97/basics.html>. June 1997.
- [47] Marco Russi. *nrf51 multi nus central*. https://github.com/marcorussi/nrf51_multi_nus_central. Mar. 2016.
- [48] Popular Science. *MEET JAPAN'S EARTHQUAKE SEARCH-AND-RESCUE ROBOTS*. <http://www.popsci.com/technology/article/2011-03/six-robots-could-shape-future-earthquake-search-and-rescue>. Mar. 2011.
- [49] Nordic Semiconductor. *nRF51 Dongle*. <https://www.nordicsemi.com/eng/Products/nRF51-Dongle>. Apr. 2016.
- [50] Nordic Semiconductor. *nRF51 Software Development Kit 10.0.0*. https://developer.nordicsemi.com/nRF5_SDK/. Apr. 2016.
- [51] Nordic Semiconductor. *nRFgo Studio*. https://developer.nordicsemi.com/nRF5_SDK/. Apr. 2016.
- [52] Byte Paradigm sprl. *Introduction to I2C and SPI protocols*. <http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/?/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>. 2016.
- [53] Ching-Kuang Shene Steve Carr Jean Mayo. *Race Conditions: A Case Study*. Tech. rep. <http://www.cs.mtu.edu/~shene/PUBLICATIONS/2001/race.pdf>. Department of Computer Science, Michigan Technological University, 2003.
- [54] Intergovernmental Committee on Surveying and Mapping. *History of Mapping*. <http://www.icsm.gov.au/mapping/history.html>. Jan. 2016.
- [55] Dassault Systèmes. *Solidworks*. <http://www.solidworks.com/sw/3d-cad-design-software.htm>. Apr. 2016.
- [56] Karl T. Ulrich. *Design, Creation of Artifacts in Society*. University of Pennsylvania, 2011.
- [57] Cadsoft USA. *What is Eagle*. <http://www.cadsoftusa.com/eagle-pcb-design-software/about-eagle/>. Apr. 2016.

- [58] Bjørnar Vik. *Integrated Satellite and Inertial Navigation Systems*. Norwegian University of Science and Technology, 2014.
- [59] Computer Weekly. *Write once, run anywhere?*
<http://www.computerweekly.com/feature/Write-once-run-anywhere>. 2002.

BIBLIOGRAPHY

Appendices

CD

Appendix 1	Datasheets
Appendix 2	Javadoc
Appendix 3	Media
Appendix 4	Message Protocol
Appendix 5	Minutes of Meetings
Appendix 6	Modelled Parts
Appendix 7	Previous Reports
Appendix 8	Progress Reports
Appendix 9	Project Management (Gantt Diagrams)
Appendix 10	Source Code nRF51-server
Appendix 11	Source Code SSNAR
Appendix 12	SSNAR Application
Appendix 13	Student Contracts
Appendix 14	UML Diagrams
Appendix 15	User Manual
Appendix 16	Wiring Diagrams