Torgeir Myrvang

Position control of robot used for localization and mapping

TTK4551 Specialization project, 7.5 credits
Supervisors: Tor Onshus

Trondheim, June 2020

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU
Innovation and Creativity

Problem Description

The main objective of this project is to improve upon the general behavior and position control of the Arduino IR. This includes finding better way of interfacing hardware, making the robot more reliable and investigate a new position controller. The goal of the project can be divided into three mayor parts:

1. The first part is to asses the Arduino IR robot by conducting a performance test using motion tracking equipment in a dedicated motion tracking lab. Investigate hardware and software issues related to the Arduino IR robot will also be a focus.
2. The next part is to develop solution motivated by fixing any issues, add missing functionalities and improve the robot.
3. The third part is dedicated to investigate a new form of position control for the robot. This involve develop and implementation of a feedback control system, using a set of developed PID controllers. The new controller shall also be put trough a series of performances tests in the motion tracking lab.

One can summarize the main approach to the problem as

- *Improve the general behavior of the robot and develop a new position control system*

Summary and conclusion

This report, associated with the course TTK4551 - Engineering Cybernetics, Specialization project, describe the implementation and development done on the Arduino IR robot. At the start of the project a performance test on the robot was conducted in the motion tracking lab. The test reviled sporadic and random movement and the robot was deemed nonoperational at the start of the project.

During the project three significant hardware issues were identified. One of these issues was short circuiting between the Arduino board and the PCB extension card. In addition, several issues in the provided software for the Arduino IR robot were identified. Among other things, errors in hardware drivers and missing features. The main goal for the project was shaped by these observations. As a result, the focus was to fix the hardware issues, improve the general behavior of the robot. This was motivated by building a solid foundation for the new position controller.

Most of the drivers on the Arduino 2018 saw mayor to minor changes. This was motivated by making the position controller as accurate as as possible. The structure of the running tasks was streamlined to increase responsiveness of the robot. The new position controller used the notion of artificial attractive potential field. To support this, a method for discrete PID control was developed, where the new position controller uses two of these. One for minimizing the distance between the robot current position and a target position. The second minimize the deviation between the angle to target and the orientation of the robot. A motor controller was developed to manipulate the motors in such a way that one could controller the speed and orientation of the robot using two inputs. The outputs of the two PID controllers was used as inputs for the motor controller.

Moreover, the new implementation was tested in the motion tracking lab. Three

position tests were conducted; "Square test", "Staircase test" and "Zik Zak test". In addition, two simple fault tolerance test were performed by introducing hardware fault to the system. From the different position test one could see a response that resulted in fast and smooth movement of the robot. There was observed an consistent offset proportional to the traveled distance. This was in the range of 50 - 150 millimeters. Although, the robot was able to reach a target position within acceptable deviations, as well as, consistently being successful in returning to the starting position.

The fault tolerance test reviled a weakness with the controller. As the controller depends on the position and orientation of the robot, the accuracy is determined by how good the estimation is. Despite this, the fault tolerance test also reviled the strength of the new implementation, as the robot was able to maneuver despite introducing simulated faults.

Preface

The content presented in this report is affiliated with the course TTK4551 - Engineering Cybernetics, Specialization project. This is a 7.5 credit course with expected work load of 11.25 hours per week. The project is given by supervisor Tor Onshus. This report act as standalone document describing the work carried out in the project.

The content that was provided at the start of the project, in which forms the foundation of said work includes: Source code for the selected robot, source code for the server and reports, documents and other related text based content carried out by students from earlier years. For equipment, a workstation was provided, access to the motion tracking lab in B333, Campus NTNU, *Gløshaugen*, table with test track and oscilloscope for measurements. As this project was carried out during the spring of 2020, the Covid-19 pandemic and the following national measure imposed on the public, impacted the planned road map. As the Campus was closed during this period, critical equipment was not available for half of the project period. This meant that some planned work was not feasible.

Special thanks to Tor Onshus for providing source material, vital guidance and consulting during the project. A thanks also goes to the follow students in 313b for providing discussion and useful information. In addition, tanks to fellow student Arild Stenset for lending private soldering bit for carry out vital repairs on the robot.

Contents

Problem Description **i**

Summary and Conclusion **ii**

Preface **iv**

List of Tables **vii**

List of Figures **ix**

List of Symbols **x**

1 Introduction **1**
1.1 Problem description 2
1.2 Project scope 3

2 Background **4**
2.1 Theory 4
Feedback control 4
Path planning 5
2.2 Digital Systems 5
Protocols and Interfaces 5
Real-time System 6
FreeRTOS 6

3 System description **7**
3.1 Hardware system description 8
3.2 Software system description 12

4 Performance of the Arduino 2018 system and issues **14**
Testing of the system 15
Issuses 16

5 Implementation **20**
5.1 Hardware fixes 20
5.2 Software implementation and improvements 21
Utility library 22
Drivers 23
Interfaces 25
New task implementation 28
5.3 Position controller 33

6 Result **36**
6.1 Position tests 36
Square test 37
Staircase test 42
Zik zak test 43
Summary position test 45
6.2 Fault tolerance test 46

7 Further work **50**
7.1 Getting the robot fully operational 50
7.2 Further improve the position controller 50
7.3 Finding dynamic model of the robot 51
7.4 Sensors 51
7.5 Collision detection and path planning 51
7.6 Distributed system 52
References 53
Appendix 55
.1 Rigid monition 55

List of Tables

4.1 Result from Square test running Arduino 2018. 16

6.1 SP coordinates for Square test 1, 1.5 and 2. 37
6.2 SP coordinates for the Staircase test 37
6.3 SP coordinates for the Zik zak test 37
6.4 Result from Square test 1 38
6.5 Result from Square test 1.5 39
6.6 Result from Square test 2 40
6.7 Result from the Staircase test 42
6.8 Result from the Zik zak test 44

List of Figures

3.1 Picture of the Arduino IR robot. Photo: Myrvang 2020 8
3.2 Components layout for the Arduino IR lego robot.[20] Illustration: Myrvang 2020 9
3.3 The Arduino Mega 2560 card. Source: [17] 10
3.4 The nRF51422 dongle. Source: [19] 10
3.5 The IMU LSM6DS3 sensor. Source: [18] 11

4.1 Square test running the Arduino 2018 software 15
4.2 Picture showing of the shorting issues 17
4.3 Picture showing the 10-pin connector 17
4.4 Picture shown the connection for the fuse 18

5.1 Illustration of the activation of each motor in the motor controller. Illustration: Myrvang 2020 26
5.2 New structure for the source code. Illustration: Myrvang 2020 . . 28
5.3 New structure for the source code. Illustration: Myrvang 2020 . . 31
5.4 Illustration of the main states and events for the position controller task. Illustration: Myrvang 2020 32
5.5 Block diagram of the position controller. Illustration: Myrvang 2020 33
5.6 Relationship between the SP position and the position of the robot. Illustration: Myrvang 2020 34

6.1 Square test 1 running the new software 38
6.2 Square test 1.5 running the new software 39
6.3 Square test 2 running the new software 40
6.4 Staircase test. Length for each SP is 200 mm 42

6.5 Plot of the result from the Zik zak test. 43
6.6 Fault tolerance test 1 46
6.7 Fault tolerance test 1 48

List of Symbols

List of Symbols

ADC	Analog to Digital Converter
ARQ	Automatic repeat request
BLE	Bluetooth Low Energy
DC	Direct Current
FreeRTOS	real-time kernel
IR	Infrared
IMU	Inertial Measurement Unit
PID	Proportional, Integral and Derivative control
PWM	Pulse width modulation
RISC	Reduced Instruction Set Computer
SLAM	Simultaneous localization and mapping
SPI	Serial Peripheral Interface
USART	Universal Synchronous and Asynchronous serial Receiver and Transmitter

θ_r	Angle of robot
θ_{at}	Angle to target
∇_θ	Delta angle
ϵ	Angle bound threshold
d_r	Distance robot
d_{sp}	Set-point distance

Chapter 1

Introduction

The selected topic for this project revolve around small robots for mapping and localization given by supervisor Tor Onshus. The project is part of a collective collaboration under the nickname *The Lego project* that dates back to 2004. As of writing this report, the collective consist of four small robots. The robots can be categorized as:

- The Arduino IR robot
- The Arduino LiDAR robot
- The NTX robot
- The nRF52 robot

The robots communicate with a main server over wireless BLE (*Bluetooth Low Energy*) communication. The server is implemented by previous students using the programming language Java. The robots have various devices used for mapping, including IR-sensors and LiDAR. This data is sent to the server, which carry out the localisation and mapping using an implementation of SLAM (Simultaneous localization and mapping). The server sends commands to the robots in the form of position coordinates, in which, the robot respond to by maneuvering to the target location. The software running on the robots are manly developed using the programming language C.

From the start in 2004, each year students are tasked with finding ways of improving and adding new useful functionalities to the robot. The collective is motivated by using off the self, easily available parts and scientific methods for making robust and functional robots.

1.1 Problem description

The selected focus for this project and report is the Arduino IR robot. The robot have suffered lack of maintenance over the years and is in bad structural shape. The software running on the robot dates back to 2016 and was original made by Ese [5]. Over the years various small changes have been made to it, without any unified direction. A consequence of this is the accumulation of small bugs and errors that have resulted in overall reduced performance. Also, as stated by Nilssen [13], *the code appears unclear and not intuitive.*

For the reason stated above, the main objective of this project is to improve upon the general behavior of the robot and how the robot is controlled when maneuvering. This includes finding better way of interfacing hardware, making the robot more reliable and investigate a new scheme for position control. The goal of the project can be divided into three parts:

1. The first part is to asses the Arduino IR robot by conducting a performance test using motion tracking equipment in a dedicated motion tracking lab. The focus will also be in investigate hardware and software issues related to the Arduino IR robot
2. The next part is to develop solution motivated by fixing any issues, add missing features and improve the robot.
3. The third part is dedicated to investigate a new form of position control for the robot. This involve develop and implementation of a position feedback control system, using a set of developed PID controllers. One will also conduct performance tests for validating the system.

One can summarize the main approach to the problem as

- *Improve the general behavior of the robot and investigate new position control system*

1.2 Project scope

To make a clear deviation between own implementation and previous work related to the robot, all software and hardware present before the project will be referred to as *Arduino 2018* in this report. When referring to Master thesis or project from other students, their name, type of assignment and date, as well as, citation will be given. Including the introduction, this report can be structured in the following way

- Chapter 1 - Introduction
- Chapter 2 - Background: Here the reader will be presented with the necessary theory and definition related to the project.
- Chapter 3 - System description: This chapter focus on providing an overview of the Ardiuno 2018 system, including both hardware and software.
- Chapter 4 - Performance test and Issues: The chapter is dedicated to describe and detail how the Ardiuno 2018 system performs and discovered issues.
- Chapter 5 - Implementation: Description of the implementation made for making the robot operational, improving the behavior of the robot and development of new position controller.
- Chapter 6 - Result: In here, the reader will be presented with the result from the performance tests.
- Appendix - Provide theory related to rigid motion.

Chapter 2

Background

2.1 Theory

Feedback control

For single input, single output feedback control a PID compensator can be used. The control gain u for the PID compensator can be defined as

$$u(t) = K_p e(t) + \int e(\tau) d\tau + \frac{d}{dt} e(t) \tag{2.1}$$

The error e is defined as

$$e = y_r - y_m \tag{2.2}$$

where y_r is the reference signal and y_m is the measured or sampled signal.[10] The continuous form of eq. (2.1) can be also be expressed as the transfer function

$$C(s) = \frac{u(s)}{e(s)} = K_P + \frac{K_I}{s} + sK_D \tag{2.3}$$

A discrete version of the continuous PID compensator can be obtained by transforming eq. (2.3) using bilinear z-transform

$$s = \frac{2}{T_s} \frac{z-1}{z+1} \tag{2.4}$$

where T_s is the sampling time. [6] This result in the following transfer function

$$C(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.5}$$

which can be transformed to the following difference equation [6]

$$\begin{aligned} u[k] = &-\frac{a_1}{a_0} u[k-1] - \frac{a_2}{a_0} u[k-2] \\ &+ \frac{b_0}{a_0} e[k] + \frac{b_1}{a_0} e[k-1] + \frac{b_2}{a_0} e[k-2] \end{aligned} \tag{2.6}$$

Path planning

Path planning is method of position control that uses an artificial attractive potential field U_{att}. In this scheme, the potential field is used to guide the search for an object that moves from its current location to an specified final location. The function describing the field U_{att} should be monotonically increasing with the distance.[10] One such field is the Euclidian distance between a current location q and the final location q_{final}

$$p_f(q) = ||q - q_{final}||$$

2.2 Digital Systems

This section concerns with defining and giving an overview of background related to the system of the robot. The reader will be present with description of terms and methods used in Embedded and digital systems. The section provide an overview of topics related to real-time systems and FreeRTOS - an real time kernel for embedded systems.

Protocols and Interfaces

PWM

Pulse width modulation or PWM is a method for modulation a digital output to generate analogue signals. It generates a pulse train, where the magnitude of the signal is constant and the width of the pulse is modulated. The width of the pulse is described as duty cycle and specify the percentage of the signals high state and the amount of low state for a given frequency. [23]

ADC

Analog to Digital Converter or ADC is a device that converts an analog signal to digital signal. It can be used to measuring voltage and other analog signals. An ADC device uses an appropriate circuits to generate a voltage proportional to the quantity to be measured. [23] The resolution of the ADC dictates the number of discrete values it can generate for a given voltage rate. [3]

USART

The Universal Synchronous and Asynchronous serial Receiver and Transmitter is a serial communication protocol. It is used for sending and receiving serial data to and from devices by converting parallel data to a serial bit stream using a shift register. [3]

SPI

The Serial Peripheral Interface is a type of synchronous master-slave protocol, that provides a simple interface between microcontrollers and other small peripheral devices. Devices uses a common clock signal which is generated by the designated

master device. Slave devices synchronize the acquisition of received data by using the clock signal. [3]

Real-time System

A definition for Real-time System can be taken from (Burns & Wellings 2005) which refers to (Randell 1995): [1] *A real-time system is a system that is required*

to react to stimuli from the environment within time intervals dictated by the environment

In real-time system on can define two form of deadlines:

- Soft real-time: A deadline is stated, but not meeting the deadline would not render the system useless.
- Hard real-time: A deadline is defined and breaching the deadline would render the system useless, or result in failure of the system. [11]

For resources shared between different task or threads ensuring mutual exclusion is of importance. Mutual exclusion means that a task or thread have exclusive access to a resource until it is returned to a consistent state. [11] Mutual exclusion and synchronization between task can be achieved with message passing. This involves using some form of First-In First-Out queue system where each task has its own queue. [1] Semaphores are another method that can be used. A Semaphores is an integer that can be initialized but can only be altered by atomic methods. These methods increments or decrements the value and ensures mutual exclusion [4]

FreeRTOS

FreeRTOS is a real-time kernel for use in small embedded systems. It is owned, developed and maintained by Real Time Engineers Ltd and is provided for free to develop real-time operation software. [11] The FreeRTOS API provides an interface for concurrent operation of multiple task on a single processor. It allows applications to be organized as a collection of independent threads of execution. [11]

The FreeRTOS provides among other thing the following standard features:

- Preemptive or cooperative operation
- Priority assignment
- Task notification
- Queues
- Semaphores

Chapter 3

System description

This section is dedicated to give an overview of both hardware and software system for the Arduion IR robot as it was before the project start. The chapter can be divided into two parts.

1. Hardware description
2. Software description

The first section focus on how the robot is built, what components the robot uses and how these interact with each other. Next, the second part deals with describing how the robots software is structured, what kinds of drivers and interfaces that are present, as well as overview of the task that are originally defined. One can summarise the main contribution for this specific system to Ese (Master 2016) [5], Andersen & Rødseth (Master 2016) [20], Lien (Master 2017) [8] and Nilssen (Master 2018) [12].

3.1 Hardware system description

Main components

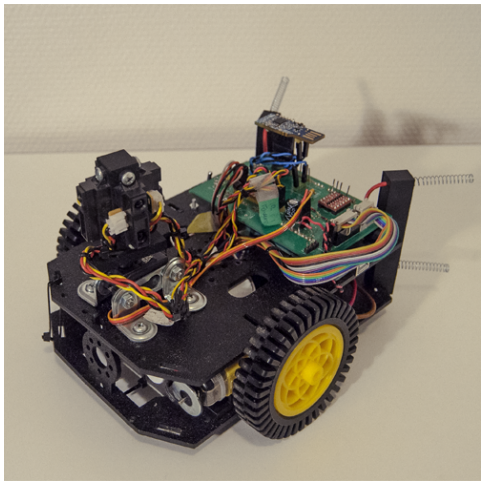

Figure 3.1: Picture of the Arduino IR robot. Photo: Myrvang 2020

.

Figure 3.1 shows a picture of the robot. Overall the Arduino IR robot consist of the following main parts:

Atmega 2560 Rev 3	8-bit AVR RISC-based microcontroller.
nRF51422	BlueTooth Low Energy dongle
Dual Motor Driver	Bidirectional motor controller card
Motor Encoders	Two neodymium 8-pole magnets and two hall-effect sensors
IMU LSM6DS3	Six Degrees of Freedom inertia sensor
Sharp G2D12	Infrared Proximity Sensors
Servo	PWM controlled servo
"Li-ion"	Rechargeable 11.1 volt Litium-ion battery
PCB shield	Extension card for the Atmega 2560 [12]

The Atmega 2560 microcontroller is the main logic controller for the robot. It runs all the necessary software on the robot. The nRF51 dongle is responsible for transmission and reception of data from the server. The motor controller card controls the direction and speed for the two DC-motors. [8] The motor encoders, mounted on each of the two motors, generates pulses that can be counted for measuring the speed and travel of the wheels. [12] The IMU is used to find the orientation of the robot. The infrared proximity sensor is used for measuring distance and mapping

the surroundings. [12] The robot uses four of them and are mounted on the servo. The litium-ion battery is the main power supply for the system on the robot. Figure 3.2 shows the layout of the main components used in the system and how they communicate with each other.

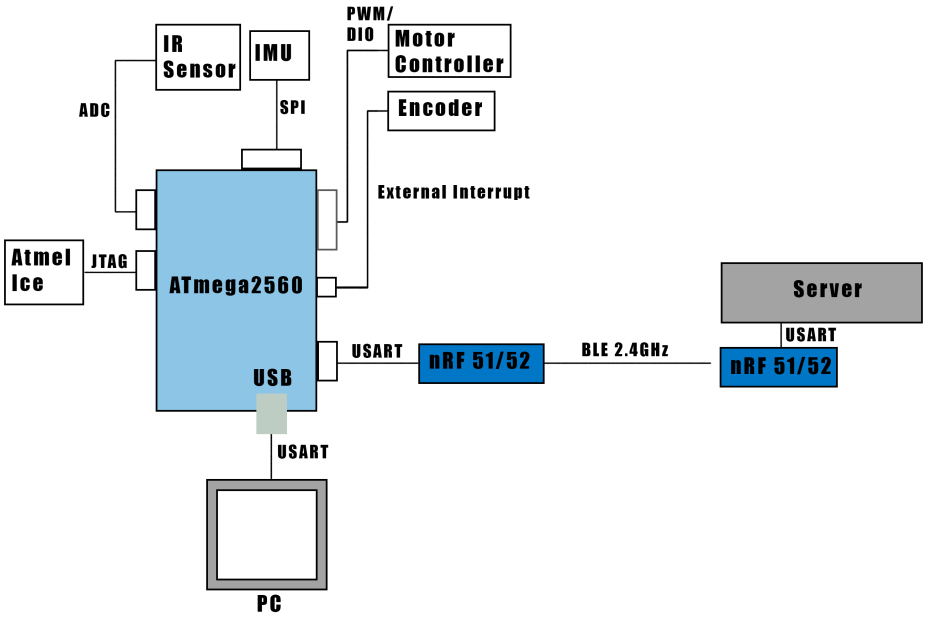

Figure 3.2: Components layout for the Arduino IR lego robot.[20] Illustration: Myrvang 2020

.

Arduino Mega 2560

The Arduino Mega 2560 board uses the Atmega 2560 chip and have 54 general purpose I/O lines. Figure 3.3 shows the specific board. The Atmega 2560 chip is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture and have 54 general purpose I/O lines. The controller have 256 Kb in-system programmable flash and 8 Kb of SRAM. [2]

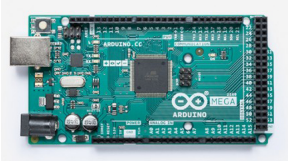

Figure 3.3: The Arduino Mega 2560 card. Source: [17]

.

The feature of the microcontroller includes, but not limited to; [2]

- Real time counter
- Six Timer/counter, four 16-bits and two 8-bits
- Interrupt support
- Four USART modules
- 10-bit ADC
- Watchdog Timer
- SPI serial port

nRF51422

The nRF51 dongle is responsible for transmission and reception of data from the server. The nRF51422 dongle is an ultra-low power 2.4 GHz wireless system on chip. The dongle can be seen in figure 3.4. It support Bluetooth Low Energy and can be used for, among other things, remote control and Personal area networks. [14]

Figure 3.4: The nRF51422 dongle. Source: [19]

.

IMU LSM6DS3

The IMU LSM6DS3 is an integrated circuit featuring 3D digital accelerometer and a 3D digital gyroscope, is used to measure motion and orientation. [9] The device supports SPI which can be used to access the registers in the device. The register can be used to control the device behavior and retrieve raw angular rate $[dps]$ and linear acceleration $[g]$.

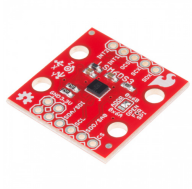

Figure 3.5: The IMU LSM6DS3 sensor. Source: [18]

.

Sharp G2D12

The sensor Sharp G2D12 is a distance measuring sensor unit, with a range of 10 to 80 cm. [21] The device outputs a voltage corresponding to the detected distance and four ADC channels on the Atmega2560 controller is used to measure this voltage.

3.2 Software system description

As a continuation from the hardware description of the robot, this section focus on the software that runs on the robot and how it is implemented. As an overview, the robot runs several concurrent task using the real-time kernel FreeRTOS. The robot communicate with a server and exchange update and order messages. The server was developed by Andersen & Rødseth [20] and is implemented using the Java programming language. Arduino 2018 also includes implemented drivers for controlling the two DC motors, reading sensor data from the IMU, the IR sensor and the encoders, as well as handling communication with the server.

Drivers

The Arduino 2018 have the implemented drivers for the two motors, the servo, IR sensor, IMU and server communication. The motor driver implements method for controlling the speed and direction of the two motors. It uses a 8-bit timer generating a PWM signal for controlling the speed of the motors. The direction is set by activating and deactivating specific output pins. The state of each motors is stored in a pointer variable. [5] The servo driver is used for rotating the Sensor tower based on a angle input. The driver uses a PWM signal for setting the position of the servo and static predefined array for mapping between the angle input and the PWM signal.[5] The IR sensor driver implements methods for reading the for four ADC channels corresponding to the four IR sensors. It maps the raw sensor data to distance using predefined array. [24] [5]

A network stack forms the basis for the communication with the server and consist of four layers. The top layer of the stack is responsible for converting Order, Update, handshake and other messages to a arrays of bytes. The next layer handles multiple connections and connection id, as well as, sending acknowledge messages to the server. Continuing, the third layer is responsible for encoding and decoding transmitted and received packages. The usart layer is responsible for transmitting the bytes holding the messages. Reception and storing is handled by a interrupt service routine. [8]

Tasks

There are total of five task that runs on the Arduino IR robot, whereas four of them can be categorised as the main tasks of the system. [5] The four main tasks are:

1. Communication task
2. Position estimation task
3. Position control task
4. Sensor tower task

Communication task

The Communication task handles the communication to and from the Java server. It uses a form of acknowledge based messaging, where the robot send a response to received messages. On power-up or reset of the robot, the task initiate the connection to the server through sending a connection and handshake message. The handshake message contains the name of the robot and physical parameters. The task also activates a additional sender task, that sends data to the server every 20 ms. The communication task uses a global data structure for reception of data. This variable is handled by temporally disables all system interrupts. Incoming messages contains commands and includes among other things, set or resetting a global handshake flag for activating the other tasks. [8]

Position Estimation task

The Estimation task is responsible of computing the position and orientation of the robot based on sensor reading from the IMU and wheel encoders. The task runs periodically and is activated using the global handshake flag set in the communication task. Ticks from the wheel encoders are sampled for each motor. From these reading the distance traveled and angle for each wheel is computed based on the arc length of a circle. The task also samples the angular rate from the IMU. This is processed by multiplying the value by a given period. The position estimate is stored in a global variable and semaphore is used for synchronize the variable. [5]

Position controller task

Position controller task handles the motion-control of the robot and uses two simple speed controller for each wheels. The task is activated and deactivated based on the global handshake flag. It takes in a global variable for the estimated position and orientation, as well as the set-point coordinates. The set-point is distributed from the communication task using a queue. The two controllers are simple implementation of the PID controller in eq. (2.1). It uses predefined threshold for handling various angels between the robot and the set-point and in some cases constant speed gain. The task computes delta values δ_x, δ_y between the estimated position and set point. These are used to compute the distance using eq. (2) and angle using eq. (3) for the target value and current position. An reference for the left and right motor is computed by using ticks from the wheel encoder. [12]

Sensor Tower task

This task runs periodically and is responsible to read the four IR sensor on the sensor tower and rotate the tower using the servo. The amount of rotation for the servo is based on the direction of the two motors. The Sensor Tower task is also responsible for sending the update message to the server, as well as sending idle messages after a counter have expired. [12]

Chapter 4

Performance of the Arduino 2018 system and issues

The problem description in section 1.1 described that the first task was to asses the performance of the robot. During the span of the project several hardware and software related issues were discovered. The robot does not seem to be very well behaved running the Arduino 2018 software, and during testing sporadic and random movement of the robot was observed. The following chapter will detail this and can be divided into two mayor parts:

- Testing of the system
- Issuses

Part one will describe how the system was tested, as well as, discuss the obtained result. The next part details the discovered issues. Again, one can divide the second part into two sub parts:

- Hardware issues
- Software issues

Testing of the system

For testing the Arduino 2018 system, a so called Square test was performed. The test was carried out in a motion tracking lab for recording the true position of the robot. The square test consist of issuing a series set-point coordinates to the robot using the Java server. For the test, the following coordinates was used

SP:	SP_1	SP_2	SP_3	SP_4
Coordinate [mm]	(1000 , 0)	(1000 , 1000)	(0 , 1000)	(0 , 0)

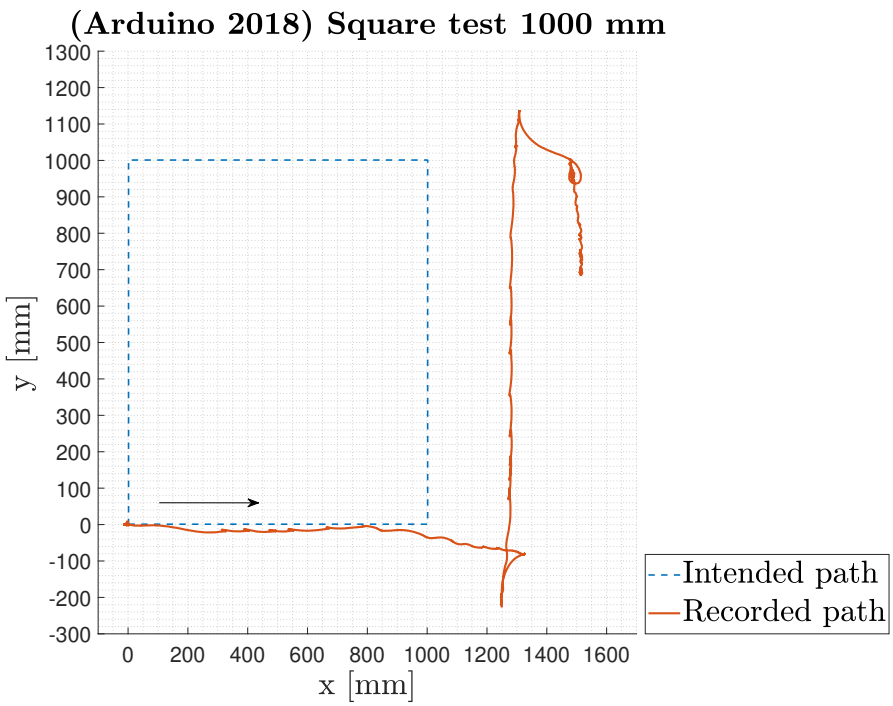

Figure 4.1: Square test running the Arduino 2018 software

.

Figure 4.1 shows the result from the square test. One can observe that the robot manages to complete SP_1 and SP_2, but fails to complete the third. From the plot one can see that the robot starts to move in the opposite direction, before spiraling around its own axis. At that point one attempted to abort the test by disconnecting the robot, which resulted in the robot speeding up and became completely uncontrollable. It was then necessary to manually reset the robot.

SP	**Value** [mm]	**End** [mm]	textbfStart-Finish time [s]
SP_1	$(1000\ ,\ 0)$	$(1324.0\ ,\ -81.0)$	$\sim 24.7s$
SP_2	$(1000\ ,\ 1000)$	$(1308.1\ ,\ 1135.2)$	$\sim 31.7s$
-			
-			

Table 4.1: Result from Square test running Arduino 2018.

One thing that is not shown in figure 4.1, is the complete behaviour of the robot. From table 4.1 one can see that the robot used upward to half a minute to cover a distance of 1 meter. It also went back and fourth when maneuvering and had jagged motion for each motor. There was also a problem of getting the robot to start the test in the first place. This is part to the fact that the robot sends messages every 20 ms. [8] This causes the server to sporadically crash and reconnecting was necessary by resetting the robot. This behavior was consistently observed over several trials. From the observed result one concluded that the robot was not operational at the time of project start.

Issuses

Hardware

For hardware related issuses, there were discovered especially two issues resulting in failure of the system:

1. Shorting between the extension card and a shield covering the USB port on the Arduino board.
2. Swapped wires for a 10-pin cable running from the extension card and the motor controller card.

Figure 4.2 shows the USB port (bottom) and the extension card. Soldering points for the the black 10-pin comes in contact with the USB shield.

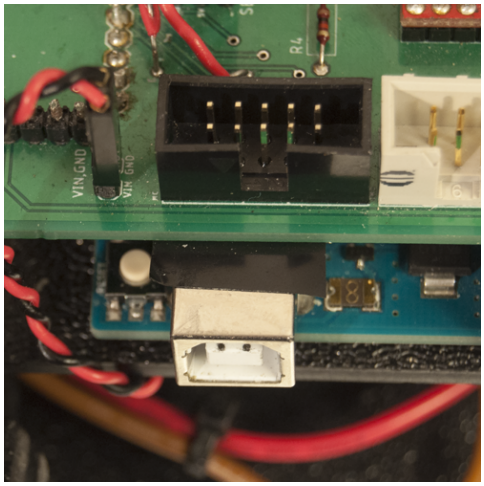

Figure 4.2: Picture showing of the shorting issues

.

Figure 4.3 shows the 10-pin connector running from the the extension card to the motor controller card. (*Note: the header in the picture have the right connection*)

Figure 4.3: Picture showing the 10-pin connector

.

Another issues was a bad soldering point connecting the battery to a fuse on the robot. This eventually broken off, resulting in somewhat hazardous handling of

the robot. Figure 4.4 shows a picture of the fuse.

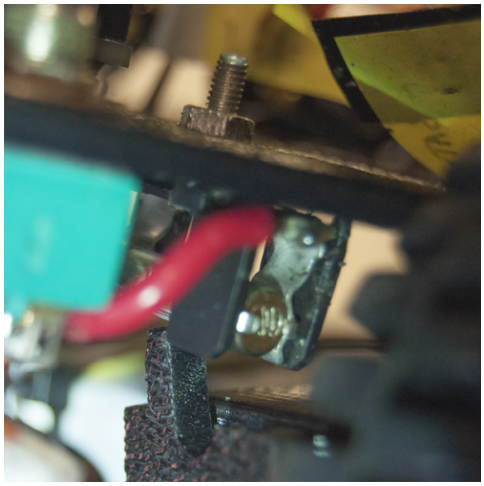

Figure 4.4: Picture shown the connection for the fuse

.

Software

Including the behavior observed in the test, other issues and lack of functionalities were also discovered

- There is no functionalities for reconnecting to the server in the event of connection losses
- System crash when the robot is disconnected before it have reached the target coordinate.
- The robot send to many messages and to fast causing the server to crash.
- System running out of RAM from bad dynamic memory management.
- Inefficient way of converting sensor data and actuator outputs.
- The servo does not move when the robot is moving.
- Uneven and insufficient control of the motors.
- Inadequate control for maneuvering to a given target coordinate.
- Drifting in the IMU sensor readings.

Some of the issues related to the software can be categorised as:

- Lack of readability and clear interfaces
- Convoluted code
- Extensive memory use

As this is a real-time system with multiple concurrent tasks, the behavior of the robot is in some notion non-deterministic. This makes it hard to pin down the exact points of failure. There is also the accumulation of small, seemingly insignificant bugs, that combined result in reduce performance and unexpected behavior. This in combination with lack of readability and convoluted code makes it difficult to debug the system. The whole code lacks readability and have in many cases weak interfaces. The problem have also be identified in earlier reports. [13]

An example of bad interfacing is the motor drivers that activates all the necessary register for controlling the motors, but also activates external interrupt for the nRF51 dongle. Another is the sensor tower task that is responsible for sending update to the server, which is an odd choice when one have defined a communication task.

An example of odd design choices is the use of a static defined array for mapping angle to PWM signal for controlling the servo. This is also apparent for mapping ADC voltage readings from the four IR sensors to distance in cm. This is a 4×256 sized matrix, where each element is one byte, resulting in at least 1Kb for storing the variable.

The current implementation also uses up to 80% of the available RAM. The network task uses dynamic memory allocation. There is guards and events for freeing this memory, but it have been observed that the robot runs out of RAM, meaning there is no guarantee that it always frees the memory. An check in the main-file for this error is also defined, indicating that this have been a problem in the past.

Chapter 5

Implementation

From the content presented in the previous chapter, it was concluded that the robot was not operational at the start of the project. The following chapter will present the changes that were made to make the robot operational and improvements made to the system. The chapter is also dedicated to document and describe how the new position controller for the robot was implemented. The chapter can be divided into three parts:

1. Hardware fixes
2. Software implementation and improvements
3. Position controller for the robot

5.1 Hardware fixes

As described in chapter 4 there were discovered three mayor hardware faults. This was the shorting between USB port and the extension card, swapped wires in header connector and broken connection for the fuse. A considerable amount of time was used to find the first two issues, due to the nature of both. In both cases, the majority of the hardware and connection on the robot was checked. For the shorting issues it was found that one or more solder terminals for the a 10-pin male header connection on the extension card came in contact with a metal shield covering the USB port on the Arduino card. Among other things, resulted in the left motor failing to rotate backwards.

In this case, an extensive debugging were carried out on the robots electronics. This involved checking every connection on the extension card for short circuiting using a multi-meter. The problem was hard to pin down, as it depended on the amount the extension card was pressed down on the Arduino card. Due to this

"Heisenbug" effect, the fault was discovered late in the project and hampered other developments. Due to limited resources, it was partial fixed by isolating the USB with electrical tape.

For the swapped wires, the robot was completely taken apart and every wire running to the extension card were checked. This fault was odd, as some tools were required to open the plastic cover off the header pin and would normally not come off naturally or without using excessive force. Due to the events described in the Preface, soldering equipment dedicated to integrated circuit was off limited. Because of this, the broken connector running to the fuse was partial fixed by soldering equipment designed for electrical equipment in cars.

5.2 Software implementation and improvements

The software improvements are limited to parts that directly and indirectly affects the performance of the position control for the robot. The methods and implementations described in this section all forms the backbone for the new position controller, which will be detailed later on. It is important to note that the new implementation uses the Arduino 2018 as a basis described in 3.2. Therefore, the most significant changes to the software will be presented. The section can be structured into the following parts:

1. Utility library
2. Drivers
3. Interfaces
4. Tasks

In the following sections will also briefly describe how the following issues where addressed:

- Reduced memory use and removal of dynamic memory allocation
- Improvement to the motor, IR sensor, IMU and communication drivers
- Abstraction for controlling the movement of the robot
- Watchdog timer for handling system crashes
- Improvement to the task implementations
- Improved the responsiveness of the system

Utility library

This library was added such that different methods and functionalities not directly related to the robot could easily be reused. It is also indented reduced the repetition of code and memory impact. This is an extension and improvements of the original implementation that had small collection of these in a file called *functions*. The library includes functionalities, such as operations on arrays, various math related functions, control flow functions and type operations. Some of the most important methods are

Discrete PID	A discrete PID implementation of eq. (2.3) with the use of bilinear transform
Linear mapping	Linear mapping of inputs values in the range $[\mathrm{in}_{\mathrm{min}}, \mathrm{in}_{\mathrm{max}}]$ to a output in the range $[\mathrm{Out}_{\mathrm{min}}, \mathrm{Out}_{\mathrm{max}}]$
Set/Rest Latch	Software Set/Reset latch used for flag operations
Stack	Interface for pushing and popping elements onto a Stack with a predefined sized array

Discrete PID

To enable reusing a PID controller for various regularization task and providing a robust controller for handling fast dynamics, an interface was made to provide this functionality. The interface was developed by analytically transforming the continuous time PID controller in eq. (2.3) to the z-domain using bilinear transform. From the resulting transfer function which has the from defined in eq. (2.5) a difference equation on the form eq. (2.6) was obtained. The parameters $b_0, b_1, b_2, a_0, a_1, a_2$ where analytically found to be

$$\begin{aligned}
b_0 &= (4 + 2T_s N)K_P + (2T_s + NT_s^2)K_I + 4NK_D \\
b_1 &= -8K_P + 2NT_s^2 K_I - 8NK_D \\
b_2 &= (4 - 2T_s N)K_P + (NT_s^2 - 2T_s)K_I + 4NK_D \\
a_0 &= (4 + 2NT_s) \\
a_1 &= -8 \\
a_2 &= (4 - 2NT_s)
\end{aligned} \tag{5.1}$$

The discrete PID implementation uses a structure for storing the variables u_k, u_{k-1}, u_{k-2}, e_k, e_{k-1}, e_{k-2}, computed weights and Ts, N. Three methods for the PID controller where implemented:

1. INIT PID CONTROLLER
2. UPDATE PID WEIGHTS

3. PID CONTROLLER

The method INIT PID CONTROLLER computes the weights u and e with 5.1 based on on the parameters K_P, K_I, K_D, T_s, N. The method UPDATE PID WEIGHTS recomputes b_0, b_1, b_2 and allow changes of the weights on run time, as well as, parameter based PID control. The method PID CONTROLLER computes the control input u and shift the previous value of u_k and e_k into u_{k-1}, u_{k-2} and e_{k-1}, e_{k-2}.

Linear mapping

As the robot interface various hardware devices and have to scale different variable from one range to another, a compact and small method for doing this was sought after. A simple yet highly useful function called LINMAP was implemented and can be used to convert various input -to -output values. For a given input, it calculates the output using the following relationship

$$\mathrm{In} = \frac{\mathrm{Out}_{\mathrm{max}} - \mathrm{Out}_{\mathrm{min}}}{\mathrm{In}_{\mathrm{max}} - \mathrm{In}_{\mathrm{min}}} (\mathrm{Out} - \mathrm{Out}_{\mathrm{min}}) + \mathrm{In}_{\mathrm{min}}$$

Drivers

Atmega2560 drivers

Three new drivers for interfacing functionalities for the Atmega2560 were added to the source code. These are external interrupt, watchdog timer and GPIO. The GPIO driver is meant to simplify manipulation of the register for controlling the IO-ports on the Atmega2560 micro-controller. For this, three function have been made:

Configure Port pin	Method for configure a IO-port as input or output
Set pin state	Method for changing the state of a IO-port
Get pin state	Method for extracting the bit value for a IO-port

Continuing, the ADC driver used for the IR sensors, as well as, the driver interfacing USART was changed. For the ADC driver an interface for selecting the individual ADC channels have been implemented. A filtering scheme for ADC readings is also employed, where a fixed sample mean have been implemented.

The original USART implementation had FreeRTOS elements in it, such as semaphores and accommodated only the specific communication with the server. This was changed in favour of a more general USART driver. The driver implements basic transmission of bytes or character, as well as, an array of character over the

two USART port USART0 and USART2. It is implemented such that the two other USART ports mention in section 3.1 can easily be added for latter use.

Motor driver

One of the task in the problem description was to improve the drivers such that it accompanied the new position controller. For that reason, three significant changes was made to the motor driver.

- The first is the resolution of the PWM for controlling the motors. This is changed from 8-bit to 16-bit such that the range can be extended from $[0, 255]$ to $[0, 2000]$.
- The second was to abstract away the need to input a specific PWM value by making use of the LINMAP function described in section 5.2. The input value is now in the range [0, 1000] and a corresponding PWM signal is calculated for the motor speed.
- The last change is the addition of a method called GET_MOTOR_STATE for getting the state of each motors. In the Arduino 2018, the direction of each motor is stored as pointer values. Problem with this method is that it will not give the exact state of each motor as it depends on the running time of the tasks using the values. This is changed to a method that directly reads the register that holds the value for the output pins controlling the motors.

Servo driver

The section 4 briefly stated that the problem with the servo driver was the use of a static declared array for mapping between angle in degrees to a corresponding PWM signal [22] As this is a problem of converting from one range to another, the driver was changed to use the LINMAP function instead.

IR sensor driver

Following the convention established in the introduction of this section and observation made of the four IR sensors, it was found that this driver also needed improvements. For making the IR sensor more memory efficient and easier to modify, the static 1000 element array for mapping between ADc values and distance in cm, as mention in section 3.2, is now removed. This is replaced with the LINMAP function.

Interfaces

To further bridge the gap between the drivers and the task, as well as, bringing more abstraction to the implementation, two new interfaces was developed. The two interfaces are for controlling the speed and direction of the robot and sampling data from the IMU sensor.

Motor controller

The main idea for this implementation was simplify further the controlling of the speed and direction of the robot using the two mounted DC motors. A similar design could be found in the Arduino 2018. The main difference the Arduino 2018 controller and the new controller is the number of inputs and the way the direction and speed of each motor is handled. The motor controller is meant to abstract away the intricate details of controlling a robot using two bidirectional motors. The method uses a state based design and a form of *joystick* control where the sign and magnitude dictates the direction and speed of the robot. The method takes in two inputs $\mathrm{x_{axis}}$ and $\mathrm{y_{axis}}$ that controls the speed and direction of the robot. The method can be divided into three parts:

1. Calculate direction based on the sign of input
2. Calculate turning speed
3. Set speed and direction for the left and right motor

First the direction is decided based on five axis region with the following rules:

Input $(\mathrm{x_{axis}}), \mathrm{y_{axis}}$	**Axis region**
$(0, 0)$	0
$(\mathrm{angle} > 0, \mathrm{speed} > 0)$	1
$(\mathrm{angle} < 0, \mathrm{speed} > 0)$	2
$(\mathrm{angle} < 0, \mathrm{speed} < 0)$	3
$(\mathrm{angle} > 0, \mathrm{speed} < 0)$	4

Likewise for the first part, the second parts divide the calculation of the turning speed in three cases. The amount of turning for the robot is not only dictated by the angle input, but also the speed of the robot. For this reason, three turning mode are defined, and two speed values $\mathrm{Speed_increased}$ and $\mathrm{Speed_reduced}$.

Turn mode	Speed
Turn sharp	$\text{Speed_increased} = \text{y}_{\text{axis}}$
	$\text{Speed_reduced} = \text{y}_{\text{axis}}$
Turn normal	$\text{Speed_increased} = \text{y}_{\text{axis}} + \text{x}_{\text{axis}}$
	$\text{Speed_reduced} = \text{y}_{\text{axis}}$
Turn increased	$\text{Speed_increased} = \text{y}_{\text{axis}} + \text{y}_{\text{axis}}/\text{x}_{\text{axis}}/100$
	$\text{Speed_reduced} = 0$

The first case in the second part is based on a threshold value S, where exceeding this causes opposite direction for each motor. The last part, sets the appropriate speed and direction based on the axis region. Figure 5.5 shows a illustration of how the motors are set based on the axis region and turning mode. Only axis region 0, and 1 is shown, whereas region 2, 3 ,4 have similar structure and the last two are used for backwards motion.

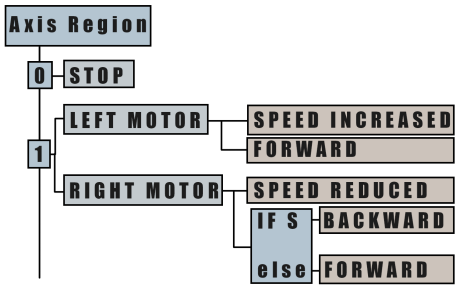

Figure 5.1: Illustration of the activation of each motor in the motor controller. Illustration: Myrvang 2020

.

IMU controller

During the project it was observed that the IMU suffered drifting in the calculated angle and noisy measurement. Because of this, an effort was put into investigate the behavior of the sensor. As mention in chapter 4, the Arduino 2018 suffered from being convoluted. The previous IMU driver contributed to this. The code was re-implemented to be more memory efficient. Another problem spotted in Arduino 2018, was that the driver used only the lower part of the register holding the sensor data. As the IMU sensor stores the senor data in two registers [9], the code was changed to extract the two values and concatenate them together.

In dealing with the noisy measurement a method CRUDE_RUNNING_MEAN was developed and gives a sufficient filtering of the data. The was discovered a positive bias in the raw data sampled for the IMU. This was handled by subtracting the data with an appropriate offset value. For a more precise approximation of the angle from the measured raw angular velocity data, two methods were developed; CONVERT_GYRO_DATA and PROCESS_GYRO_DATA.

The method CONVERT_GYRO_DATA converts the raw data read from the IMU sensor by using a scaling factor. It uses a separate scaling factor for when the data is positive and negative, where both values was found empirically. The method PROCESS_GYRO_DATA was designed utilising the following relationship:

$$\begin{aligned} \omega &= \dot{\theta} \\ &= s\theta \end{aligned} \tag{5.2}$$

eq. (5.2) was converted using bilinear transform and a difference equation for θ was analytically found as

$$\theta_k = \theta_{k-1} + \frac{K}{T_s}(\omega_k + \omega_{k-1}) \tag{5.3}$$

New task implementation

The changes from the previous sections, in which all individual components where tested to ensure correct behavior, made the task implementation from Arduino 2018 somewhat less compatible. This and the issues mention in chapter 4 it was decided to re-implement all of the current task. As mention in section section 3.2 there was an communication task, pose estimation task and pose controller task, as well as, sensor tower task in the original implementation. In this project mayor changes were made to the communication task, and pose controller task. The sensor tower task and estimator task saw only minor changes. The new system uses now the following task:

System monitor task
Server communication task
Position controller task
Position estimation task
Sensor tower task

A new task have also been added for system monitoring. As of now, this task is responsible for resetting the watchdog timer and have, relative to all the other task, the highest priority.

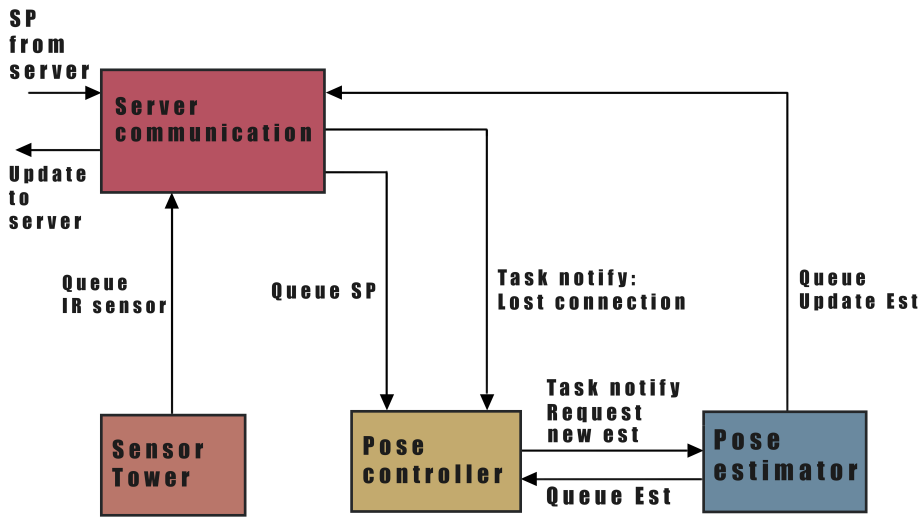

Figure 5.2: New structure for the source code. Illustration: Myrvang 2020

.

The decision of re-implementing the task system open up the opportunity to stream-

line the communication and synchronization between the tasks. The new implementation relies exclusively on queues and notification signaling interfaces provided by the FreeRTOS kernel. Figure 5.2 shows a illustration on how the different tasks interact with each other. Each task is designed to be independent of each other, in the sens that they do not share resources with each other and only exchange read-only data. A advantage of this is the limited need of using semaphore and mutex mechanics. This is a mayor department compared to the original implementation, which used a combination of queues, global variables and semaphores for sharing information and synchronization. One of the problem with semaphores is that they do not scale well, where changes to one task result in the need to change all dependent task. This can be avoided by using queues and message passing, where one only need to define input and output channel for each task.

Figure 5.2 also illustrate the communication flow, where one can see the communication flow starts when the robot receive a new set point. The data is then forwarded to the Position controller task, in which takes appropriate actions. Among other thing, signaling the estimator task to send the latest position and orientation estimate.

All the task are now periodically, with the respective period as follows

Name	**Period** (ms)	**Priority**
System monitor task	3940	10
Server communication task	485	3
Position control task	25	4
Position estimation task	50	5
Sensor tower task	80	2

The periods are selected such that the interference between the task is reduced. The priority assignment is based on the critically of the task. As such, the position controller relies on the latest estimate to function properly. Therefore, the estimator task have a higher priority than the position control task.

Senor tower task

The Sensor tower task uses the the improved driver for the IR sensor described in section 5.2. One of the issues described in chapter 4 is that the servo does not move when the robot moves. This is now been addressed by using a implemented ramp function for continuously increasing and decreasing the angle of the servo.

As mentioned in section 5.2 the ADC driver employs filtering and the four IR sensors are further filter in the Sensor tower task. It uses a low pass filter, which was originally used in the Arduino 2018 for filtering the Gyro measurement. Continuing, the task also uses the LINMAP function described in section 5.2 for mapping the raw sensor data in the range [0, 1023] to distance in the range [80, 5]. The output range for this mapping is inverted such that low values of the raw data correspond to large values in the output distance.

Estimator task

As with the senor tower task, only minor changes to the Estimator task have been made. One difference for the estimator task is the use of queue instead of a global variable for distributing the estimated position and orientation. The task also uses the function GET MOTOR STATE described in section 5.2 for getting the direction of the motor instead of a variable. For calculating the orientation of the robot it was found that using the encoder resulting in large deviation and drifting. As a consequence of this the estimator task use only the gyro for finding the angle of the robot using the methods described in section 5.2.

Communication task

As mention in chapter 4 the robot did not reconnect in the event of losing the connection and the issue of exception on the server side, as well as insufficient handling of semaphore made system crashes on several occasions. During the research of the Arduino 2018 it was noted that the robot only need to send and receive messages with a maximum size of 50 bytes. Because of this and the mention issues it was decided to streamline and reduce the impact of the communication on the system.

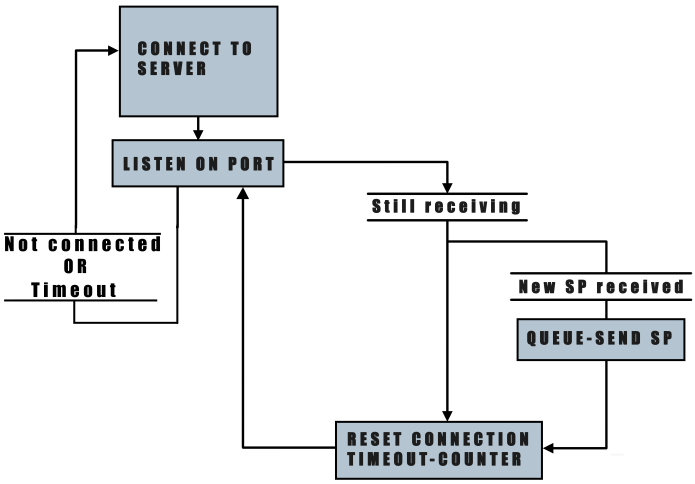

Figure 5.3: New structure for the source code. Illustration: Myrvang 2020

.

The communication task uses a stated based design. Figure 5.3 gives a overview of the execution flow of this task. At the start of the program it goes into to the CONNECT TO SERVER state. After a connection message is sent the task will enter a LISTEN ON PORT state where it will check for any received messages. If the message contain a new set-point coordinate it will then transferred this to a queue and send it to the Position controller task.

The communication stack is now reduced to three layers. The network layer have now been transformed to a state based design for reception. In similar fashion as the Arduino 2018, the new implementation uses an interrupt service routine ISR for capturing bytes received over USART. What differs is the amount of operation carried out in this routine. The old implementation used a large buffer with a loop, in side the interrupt routine. These types of ISR should not use operation exceeding 10 ms [2]. Therefore, changes have been made such that the bytes are pushed to a stack described in section 5.2. The new implementation uses a SR-latch for signaling when a complete massages is received and lock the reception of further bytes until the message is processed. The processing is done in a separate method. The reception in the server communication layer is also state based. As the communication between the server is acknowledge based, the reception method send an acknowledged message for each received massage.

Position controller task

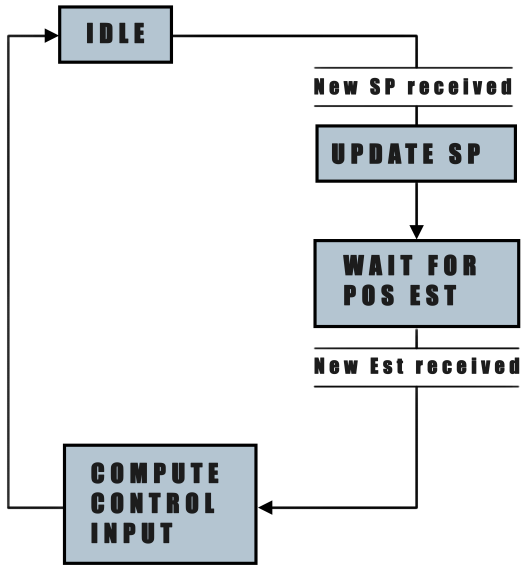

Figure 5.4: Illustration of the main states and events for the position controller task. Illustration: Myrvang 2020

.

A state based design was also found fitting for the position controller task. It a state machine, where the structure can be divided into three states; IDLE, UPDATE SP and COMPUTE CONTROL INPUT. New set-point coordinate and estimate position from the communication task and estimator task is used as the main event in the state machine.

Figure 5.4 gives an simple overview of how the task is structured. The controller starts in the IDLE state. Here it waits for new set-point coordinates to be sent from the communication task. If a new SP is added to the queue, then the task will lock a latch variable and go to the state UPDATE SP. In this state, the content from the queue is read and copied to a local variable. Distance for the set-point value is also computed here, as well as, sending a notification to the estimator task. After finishing computation, the task will the go back to the idle state and wait for new estimated position and orientation from the Estimation task. When a new estimate is available necessary parameters are computed and the task goes to the COMPUTE CONTROL INPUT.

5.3 Position controller

In chapter 4 it was stated that the robot suffered from jagged motion and inefficient control when maneuvering. This motivate further the stated objective of develop a new position controller for the robot. The previous section described implementation and changes to improve the general behavior for the robot and forms the foundation for the new position controller. The Position controller task is responsible for running the new position controller.

The new position controller utilize the open-loop motor controller described in section 5.2 that makes it possible to control the robot using only to inputs. Correspondingly, a natural choice for regulating the speed and heading of the robot was to use two discrete PID controller described in section 5.2 for feedback control of the position and orientation.

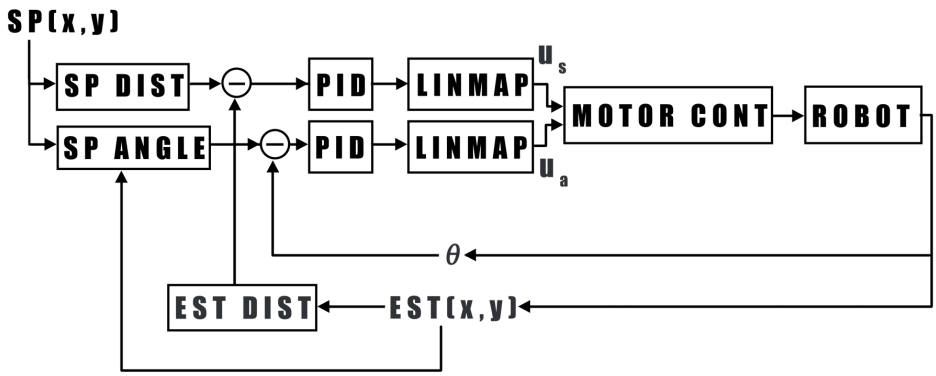

Figure 5.5: Block diagram of the position controller. Illustration: Myrvang 2020

.

Figure 5.5 shows a block diagram of how the controller is structured. The controller scheme uses notion of attractive filed described in section 2.1, with the two PID controller referred to as $\mathrm{PID}_{\mathrm{dist}}$ and $\mathrm{PID}_{\mathrm{angle}}$. The output of $\mathrm{PID}_{\mathrm{dist}}$ is used as the speed input and the output of $\mathrm{PID}_{\mathrm{angle}}$ is used as the angle input in the motor controller. Both controllers outputs values in the range of [-10000, 10000]. This is scaled down two [-1000, 1000] for the the motor speed and [-2000, 2000] for the motor angle. using the LINMAP function described in 5.2.

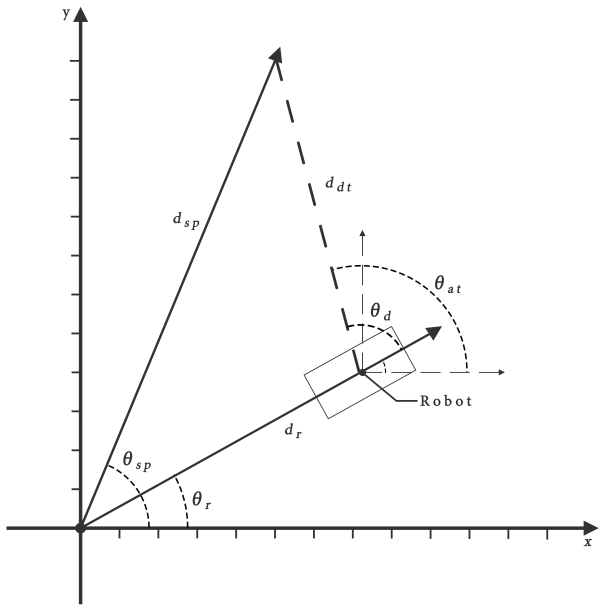

Figure 5.6: Relationship between the SP position and the position of the robot. Illustration: Myrvang 2020

.

For making the attractive field method work with the two PID controllers, some new parameters had to be defined. Figure 5.6 shows a illustration of the parameters used in the controller scheme and how the relate to each other. The set-point distance d_{sp} is the distance between the set-point coordinate and the origin. d_r is the distance traveled for the robot and θ_{at} is the distance between the set-point coordinate and estimate position. One important detail about the $\mathrm{PID}_{\mathrm{angle}}$ controller is that it uses degrees as unit. The reason is that the angle values and the related variables can be stored as an 16-bit integers. This result in less memory usage while maintain the benefit one might had from using radian as unit and floating point values.

For minimizing the the distance between the target position and the current position of the robot, the $\mathrm{PID}_{\mathrm{dist}}$ is used. For $\mathrm{PID}_{\mathrm{dist}}$, d_r is used as the measured process value and the input reference is zero. The second controller $\mathrm{PID}_{\mathrm{angle}}$ is used to minimize the deviation between the orientation of the robot θ_r and the angle to target θ_{at}. In similar fashion to the $\mathrm{PID}_{\mathrm{dist}}$, the $\mathrm{PID}_{\mathrm{angle}}$ also have a zero input reference. Δ_θ is the process value for $\mathrm{PID}_{\mathrm{angle}}$ and is the difference between

current angle of the robot θ_r and θ_{at}.

Because both θ_{at} and θ_r are computed using the $\mathrm{atan2}$ function defined in section .1, the angle can jump from -179 to 180 and vise versa. For that reason, method for handling the discontinuities when computing Δ_θ was devised. Δ_θ is defined as

$$\Delta_\theta = \begin{cases} (\theta_{at} - \theta_r) - 360 & (\theta_{at} \geq 90, \theta_r < \epsilon) \\ 360 - (\theta_{at} - \theta_r) & (\theta_{at} \leq -90, \theta_r > \epsilon) \\ (\theta_{at} - \theta_r) \end{cases} \tag{5.4}$$

where the bound ϵ is change based on the detection of a jump from -179 to 180. The jump is detected by finding the change between the previous θ_{at} and the current θ_{at}, as well as, recording the sign of the current change and the previous change. Based on this, ϵ is updated using the following rule:

$$\epsilon = \begin{cases} 0 & \mathrm{sign_k} + \mathrm{sign}_{k-1} = 0 \\ 90 & \mathrm{otherwise} \end{cases}$$

Chapter 6

Result

In the same fashion the previous chapter presented various implementation, this chapter is dedicated to present and discuss the result obtained from the project. As mention in section 5.2, the main focus was to making the robot operational, as well as, investigate a new scheme for position control. This chapter present result obtained from performance tests conducted on the motion tracking lab, for testing the new controller scheme and the system as a whole.

In the sense of system performance, two fault tolerance test were also devised. These test were made to see how well the robot respond to hardware faults and how well the robot is able to maneuver when these are present.

6.1 Position tests

For maneuverability and performance of the position controller, three position tests were devised:

1. Square test
2. Zik zak test
3. Staircase test

Each test was conducted by manually inputting set-point coordinates SP from the Java server, were the task of the robot was to reliable receive the message and maneuver to the given coordinate, within a reasonable time frame. By reasonable time frame, means that the robot should cover a distance of 1 meter within 1 minute.

Another purpose was to implicitly test the whole system. Accordingly, one require a functional communication task for communication with the server, reasonable estimate of the position, fast responsiveness and reliable interfaces for the hardware devices. The square test was carried out by inputting coordinates forming squares of 1, 1.5 and 2 meters. Table 6.1 shows the specific coordinates inputted

for each tests.

Name	SP
Square 1	$[(1000, 0), (1000, 1000), (0, 1000), (0, 0)]$ [mm]
Square 1.5	$[(1500, 0), (1500, 1500), (0, 1500), (0, 0)]$ [mm]
Square 2	$[(2000, 0), (2000, 2000), (0, 2000), (0, 0)]$ [mm]

Table 6.1: SP coordinates for Square test 1, 1.5 and 2.

The Staircase test was conducted by inputting coordinates that form the shape of a staircase. This test was used to see how well the robot maneuvered over small distances and was meant to simulate coordinates outputted by the localization algorithm on the Java server. Table 6.2 shows the specific SP coordinates

Name	SP
Staircase	$[(200, 0), (200, 200), \cdots, (1000, 1000)(0, 0)]$ [mm]

Table 6.2: SP coordinates for the Staircase test

Likewise, the Zik zak test was meant to see how well the robot respond to large deviation in the target angle and how the robot maneuver when these are present. Table 6.3 shows the specific SP coordinate

Name	SP
Zik zak	$[(1500, 0), (0, 1000), (1500, 2000), (0, 2000), (0, 0)]$ [mm]

Table 6.3: SP coordinates for the Zik zak test

Square test

The square test 1, 1.5 and 2 was conducted on the new implemented system with the use of the new position controller scheme. Figure 6.1, 6.2 and 6.3 shows the test for 1, 1.5 and 2 meters respectively. Table 6.4, 6.5 and 6.6 reports the recorded target coordinates for the robot and the time it took to complete each SP.

Square test 1			
SP	**SP value** [mm]	**Recorded** [mm]	**Start-Finish time** [s]
SP_1	(1000 , 0)	(1070 , 20.9)	$\sim 4.0s$
SP_2	(1000 , 1000)	(1095.0 , 1170.0)	$\sim 4.2s$
SP_3	(0 , 1000)	(65.5 , 1133.5)	$\sim 4.2s$
SP_4	(0 , 0)	(26.7 , 27.5)	$\sim 4.1s$

Table 6.4: Result from Square test 1

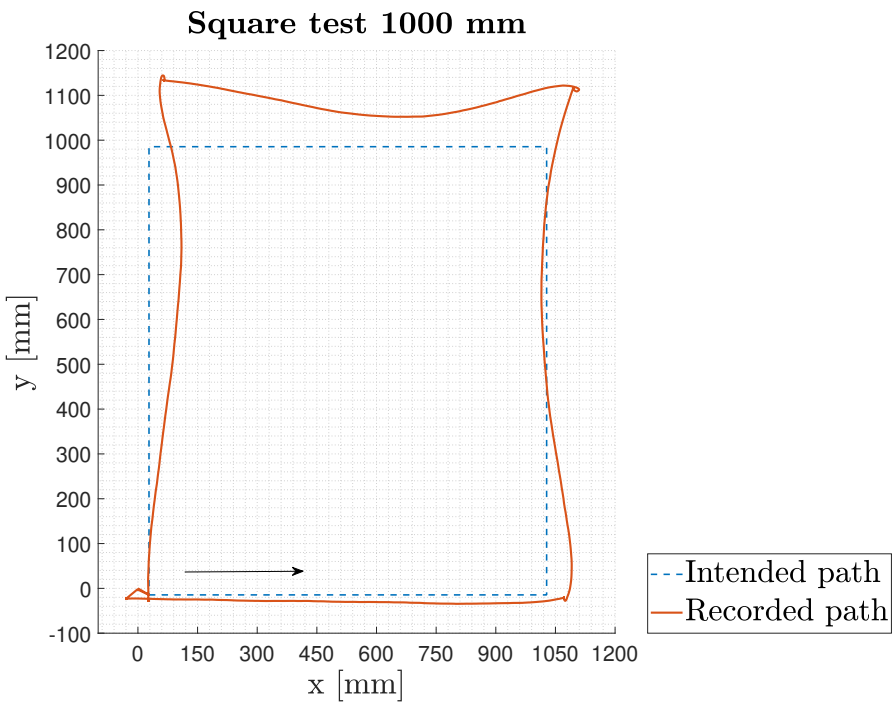

Figure 6.1: Square test 1 running the new software

.

Square test 1.5			
SP	**SP value** [mm]	**Recorded** [mm]	**Start-Finish time** [s]
SP_1	$(1500\ ,\ 0)$	$(1609.3\ ,\ -41.0)$	$\sim 4.8s$
SP_2	$(1500\ ,\ 1500)$	$(1682.2\ ,\ 1540.0)$	$\sim 5.2s$
SP_3	$(0\ ,\ 1500)$	$(-0.2\ ,\ 1642.6)$	$\sim 5.0s$
SP_4	$(0\ ,\ 0)$	$(-16.6\ ,\ 63.1)$	$\sim 5.3s$

Table 6.5: Result from Square test 1.5

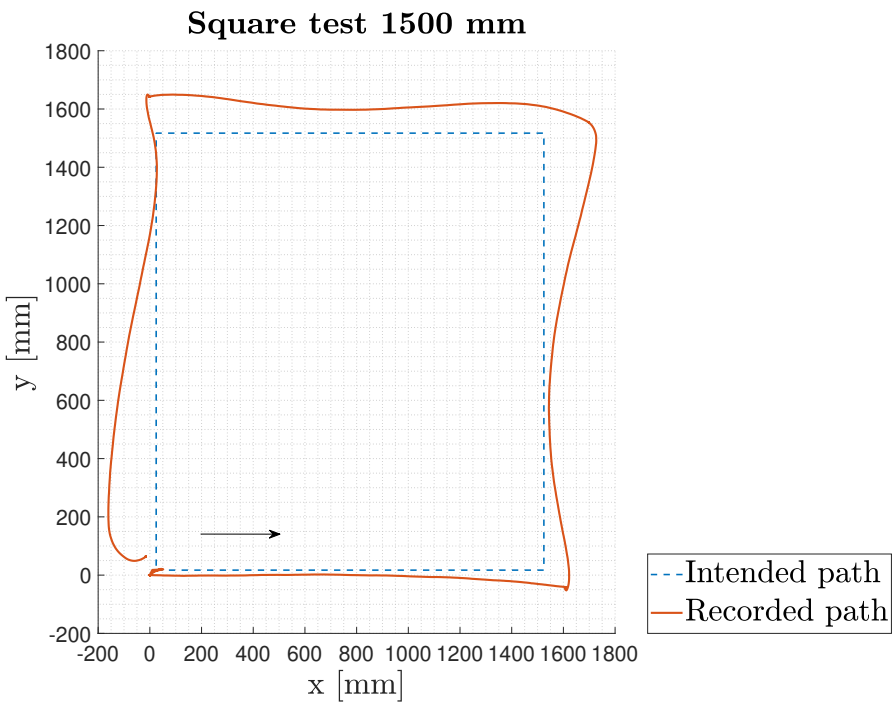

Figure 6.2: Square test 1.5 running the new software

.

Square test 2			
SP	**SP value** [mm]	**Recorded** [mm]	**Start-Finish time** [s]
SP_1	$(2000\ ,\ 0)$	$(2129.8\ ,\ -78.5)$	$\sim 6.7s$
SP_2	$(2000\ ,\ 2000)$	$(2227.0\ ,\ 2023.4)$	$\sim 6.5s$
SP_3	$(0\ ,\ 2000)$	$(46.7\ ,\ 2091.3)$	$\sim 5.9s$
SP_4	$0\ ,\ 0)$	$(-45.7\ ,\ -184)$	$\sim 6.0s$

Table 6.6: Result from Square test 2

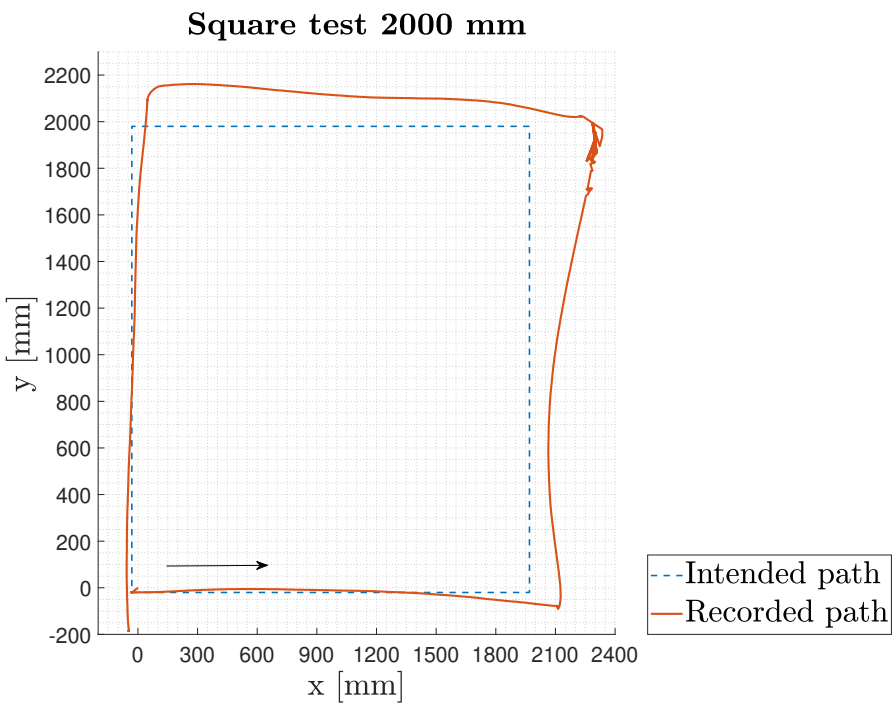

Figure 6.3: Square test 2 running the new software

.

From figure 6.1 one can see that the deviation in the final destination for each SP is in the range of 150 milliliters. To put that number in perspective, the width of the robots wheelbase is measured to 155 mm. One interesting observation, is the consistency of these values. This gives a positive indication for the controller and its ability to perform over different ranges. It is worth noting that the target threshold was set to 30 mm for each test. For each one can see the time to suc-

cessfully move from one location the the next is drastically reduced compared to the times observed from the test conducted in chapter 4. Another observation to be made, is the consistency of the time for each SP in a given test, indication that the robot is equally effective when maneuvering in each direction. One can see arc paths forming while the robot is moving, which is especially prominent in Square test 1 and 1.5. The magnitude of these paths are in the range of 100 mm and the shape is consistent for Square 1, 1.5 and 2.

The formed paths can be caused by the final orientation of the robot as it reaches the target position. This can be related to the behavior of the new controller scheme. in which the calculated angle to target will constantly change. As $\mathrm{PID}_{\mathrm{dist}}$ described in section 5.3 outputs a value corresponding how close the robot is to the target, $\mathrm{PID}_{\mathrm{angle}}$ will output a value corresponding to minimizing the error between the target angle and the orientation of the robot. As the robot closes to the target position, the angle to target will increase significantly and causes the robot to change direction as it approaches the destination. Because of this, the next path is closely related to how the robot approached the previous path. Nevertheless, the robot is able to successfully reach a position that is close to the target value and is able to return to origin within an acceptable error. This is due to the fact that the position controller tries to both minimize the distance to target as well as the angle deviation at the same time. Therefore, the robots initial orientation does not affect its ability to reach the target destination, only the path formed when maneuvering.

Staircase test

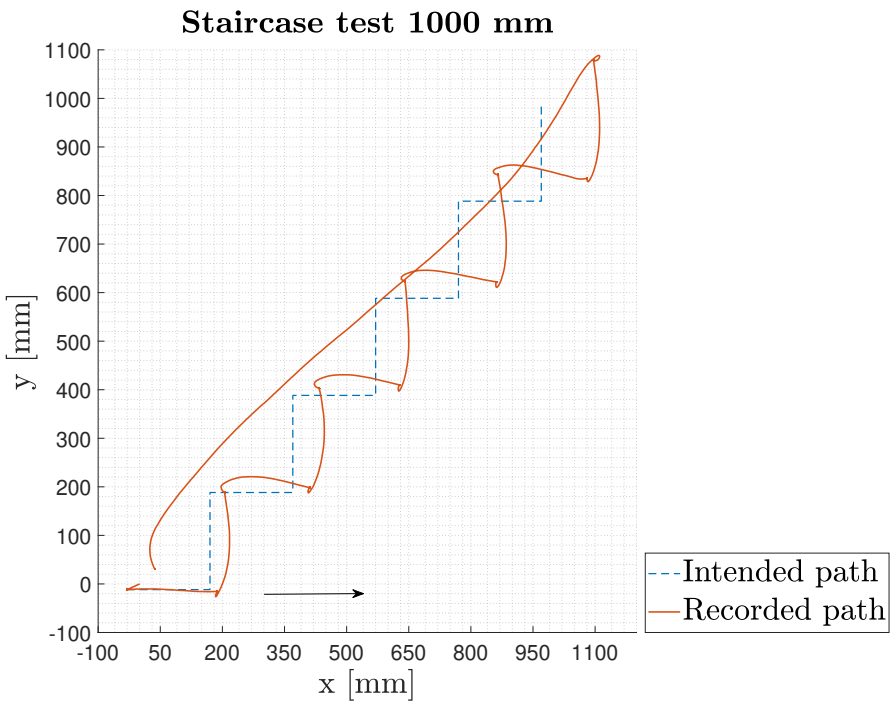

Figure 6.4: Staircase test. Length for each SP is 200 mm

.

Staircase test		
SP	**Start** [mm]	**End** [mm]
SP_1	$(200\ ,\ 0)$	$(187.8\ ,\ -15.5)$
SP_2	$(200\ ,\ 200)$	$(205.2\ ,\ 188.4)$
$\vdots$		
SP_9	$(1000\ ,\ 800)$	$(1080.3\ ,\ 834.9)$
SP_10	$(1000\ ,\ 1000)$	$(1096.2\ ,\ 1078.7)$
SP_11	$(0\ ,\ 0)$	$(36.8\ ,\ 30.4)$

Table 6.7: Result from the Staircase test

.

From figure 6.4 it is observed that the angle between two target values are marginal. In addition, from table 6.7 one can see that the robot is able to maneuver back to the origin with in a error equal to the target threshold. However, one can see a steady and increasing offset in the position of the robot, as it progress to the target positions. This shows the importance of the test, which is designed to test the robots ability to cover small distances while change direction for every setpoint values. It is also worth noting that the final recorded position have similar deviation as Square test 1.

The offset can be caused by the use of the wheel encoders for calculating the position. As mention in the Square test, the robot uses a target threshold of 30 mm. By inspecting the first two target coordinates one can see that the robot stops at position that is close to this. By taking this into account, the estimated position have a relative small error, yet significant for achieving better precision.

Zik zak test

Figure 6.5 shows the result from the Zik zak test.

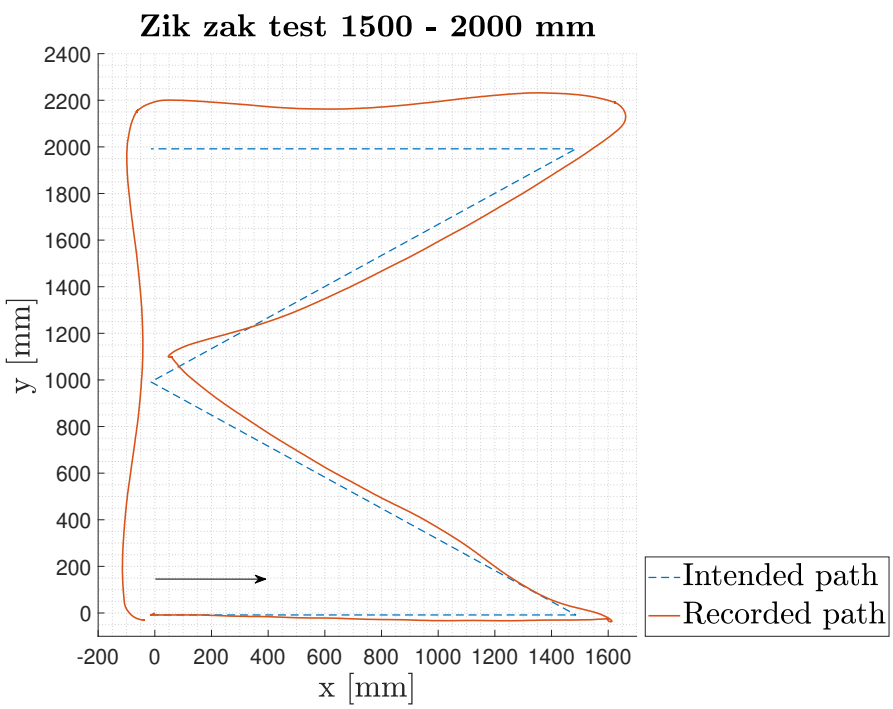

Figure 6.5: Plot of the result from the Zik zak test.

.

Zik zak test		
SP	**Start** [mm]	**End** [mm]
SP_1	$(1500\ ,\ 0)$	$(1598.9\ ,\ -24)$
SP_2	$(0\ ,\ 1000)$	$(58.7\ ,\ 1098.9)$
SP_3	$(1500\ ,\ 2000)$	$(1623.9\ ,\ 2190.5)$
SP_4	$(0\ ,\ 2000)$	$(-63.7\ ,\ 2152)$
SP_5	$(0\ ,\ 0)$	$(-39.1\ ,\ 30.8)$

Table 6.8: Result from the Zik zak test

.

The test is meant to challenge the controller scheme, as SP_2 and SP_3 require the robot to turn more than 90 degrees. As the controller uses a PID controller for the angle, one would suspect overshoot for large deviation in the target angle. From figure 6.5 one can see that this is reduced compared to that observed in each of the Square tests. This gives an indication that the tuned values for the controller gain is more optimal for larger angles.

The deviation between the target value and the recorded position of the robot is similar to that observed in the Square tests. From table 6.4, one can observe a similar start-end deviation and offset, in which can be linked to the factors mention in the Square and Staircase test.

In the tuning process for the two PID controllers, one had to balance the responsiveness of the system. To small gain K_p in the angle controller would result in a large arc forming as the robot approached the target position. The opposite, large K_p gain would result in large overshooting when turning. One could combat this by having an equal large K_d gain but this resulted in high frequency oscillation when maneuvering.

On the contrary, the behavior is partly combated by the motor controller, in which for large errors in the angle, causes the the robot to turn around by its wheelbase axis. For the sharp turn mode, the speed of each motor is determined by the $\mathrm{PID}_{\mathrm{dist}}$ controller. As with Square test 2, one can observe that the robot runs more and more smoothly as the distance increases. This can be the fact that a small integral gain is used for both the $\mathrm{PID}_{\mathrm{angle}}$ and $\mathrm{PID}_{\mathrm{dist}}$. As the distance increases the error accumulation of angle deviation increases and would eventually result in a large enough output gain.

An interesting observation can be made about the controller and its relationship with the artificial potential field. If one tuned the $\mathrm{PID}_{\mathrm{angle}}$ to act slowly by having a large integral gain, would result in a spiraling effect when the robot comes closer to the target position. The movement was similar to the motion of objects revolving around planet. The formed path had also a similar shape as a stable linear system with complex eigenvalues.[7]

Summary position test

General for each of the position tests is the deviation between the set-point coordinate and the recorded position of the robot. A common observation is that the offset increases with the travel of the robot. An interesting aspect is the Staircase test have a similar offset as Square test 1. One thing to note is that the controller operates on the estimated position of the robot, which limits how accurate it can be. A way to improve the position controller is to improve the estimation of the position and orientation, as well as, more accurate sensor readings. Another aspect is that the controller is reactive to changes in the dynamics of the robot. This is a disadvantage of the controller as changes to the estimated position and orientation would mean re-tuning of the controller.

During development and testing, it was observed that changes to, example parameters for the wheel encoders would impact the final orientation of the robot when it reach the target. As it have been observed, the controller manages to give good performance for orienting the robot in the range [-160,160] and adequate control in the critical region [-179.180]. However, the result from the Zik zak test indicates that the tuned values for the controllers are more optimal larger deviation. To improve the performance, one can employ several controller gain for separate angle regions. A challenge with this is to handle the change from one region to another. One solution is to utilise a hysteresis mechanism for switching between controller gains. A prototype for this was developed, but further testing is required.

One of the challenges for the new controller scheme was the region [-179, 180] for the target angle. Test where conducted without handling this critical region, which resulted in the robot oscillating back and forth, unable to move in the desired direction. This critical region was handled using the implemented method described in section 5.3. The method does not completely eliminate the oscillation scenario, but rather reduce the risk of it happening. Again this has to be taken into account when tuning the controllers, as to high responsiveness increases the risk of getting the robot stuck at the critical region.

6.2 Fault tolerance test

Two simple fault tolerance test were conducted. For each of the tests, the code was altered in such a way to simulated hardware fault for the two motors. As the estimator task depends on the encoders for estimating the robots position, a fault was introduced by swapping the left and right encoder pins. This was used in both of the tests. For the first test, the left motors ability to go backwards was removed. This was done by keeping the correspondingly output pin on the Arduino board constantly low. This was to simulate broken output pin or short circuiting. For the second, the speed gain of the right motor was greatly reduced, to simulate worn-out motors.

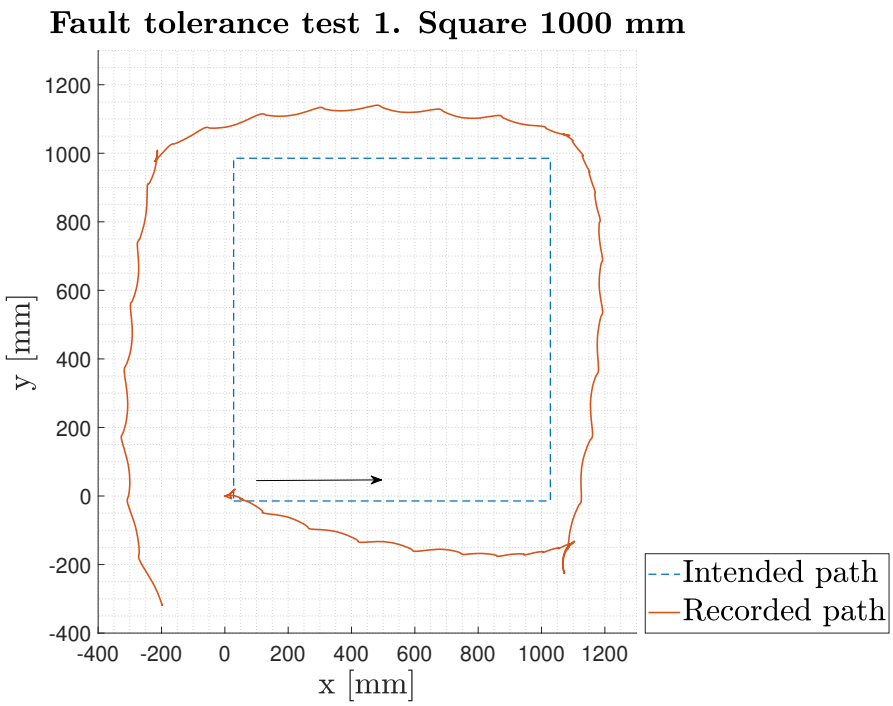

Figure 6.6: Fault tolerance test 1

.

Fault tolerance test 1		
SP	**Start** [mm]	**End** [mm]
SP_1	$(1000\ ,\ 0)$	$(1071.7\ ,\ -224.4)$
SP_2	$(1000\ ,\ 1000)$	$(1071.7\ ,\ 1055.4)$
SP_3	$(0\ ,\ 1000)$	$(-220.1\ ,\ 977.4)$
SP_4	$(0\ ,\ 0)$	$(198.4\ ,\ -317.4)$

From figure 6.7 one can see that small ripples forming as the robot is only able to exert most force on the right motor. As expected, the robot struggles to keep a straight line when maneuvering, but is able to maneuver in some degree. As mention in section 5.2 the motor controller have a sharp turning mode, when exceeding a threshold value causes one of the motors to change direction. This means that when the angle error is great enough, the $\mathrm{PID}_{\mathrm{angle}}$ will output a correspondingly value, which in turn, activates the sharp turn mode in the motor controller.

However, the test reveals a weakness with using the wheel encoders for position estimate. As the robot rock back and fourth trying to correct its course, the accumulated error in the estimate will increase. A resulting consequence is the large offset between the target position and where the robot stops.

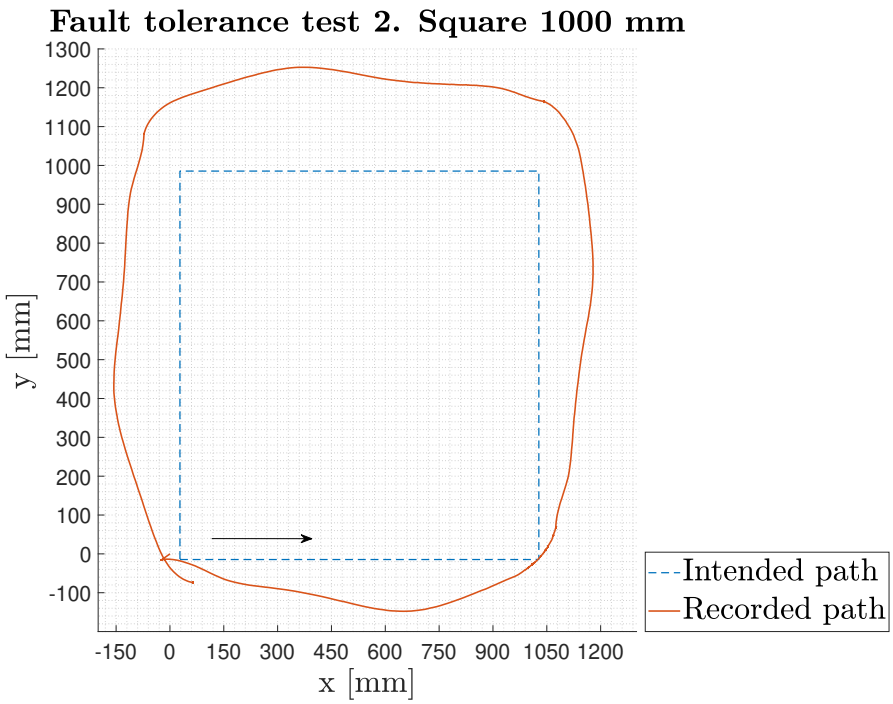

Figure 6.7: Fault tolerance test 1

.

Fault tolerance test 1		
SP	**Start** [mm]	**End** [mm]
SP_1	(1000 , 0)	(1075.1 , 67.5)
SP_2	(1000 , 1000)	(1042.7 , 1164.6)
SP_3	(0 , 1000)	(72.4 , 1081.1)
SP_4	(0 , 0)	(64.8 , 73.8)

From figure 6.7, one can see that the the robots ability to do sharp turns is greatly reduced. As the encoder pins are swapped, this would result in error in the calculated position, which in turn result in errors in the angle to target value. One also see that the arcs when maneuvering are larger when introducing the output pin error. However, as seen in table 6.2, the robot is able to reach the target position, within an acceptable error. In fact, the end position of the robot is in pair with the observed deviation in Square test 1. This indicates a strength of using only the gyro for orientation measurement. It also indicates that swapping the encoder pin

have little effect on the estimated position.

This in some sense contradict the observation made for figure 6.7. On the contrary, the robot will only stop when it detects a estimated position that is within the target threshold value. For that reason, using the encoders is still an inaccurate way of estimating the relative position of robot, as it depends to much on the particular wheel moments. To increase the accuracy of the estimate and resulting position controller one may seek other sensors that is independent of the wheels.

Chapter 7

Further work

7.1 Getting the robot fully operational

The robot is in most part operational.The result in section 6.1 showed that the robot is able to maneuver to a target location within an acceptable time frame and error margin for the position. It handles well connection and re-connection to the server, as well as, sending and receiving data. One problem that is not fully sorted is the sending of update messages. The server is able to detect the correct form of the messages but only sporadically make any response. One suspect that this is a problem on the server side, due to the long scheduling period of the communication task on the robot for reducing the interference to the rest of the system. The main issue with fixing this problem is the limitation one has to impose on the system running on the robot. However, in the latter years the focus have been shifted to using a thread network which was investigated by Petersen (Master 2019) [16]. For that reason, further improvements to the communication may be focused on making the robot compatible with the thread network.

7.2 Further improve the position controller

The newly implemented position controller is show to give smooth movement of the robot considering it only uses two sets of PID controllers. The discrete PID controller is able to respond to small and sudden changes in the speed and direction of the robot. The biggest factor hampering the performance of the controller is the critical region [-179, 180]. A better way of handling this is required for making the controller work across all angle ranges.

It was also during testing noted a significant difference between the speed of the two motors. A solution may involve utilizing the output of the $\mathrm{PID}_{\mathrm{angle}}$. This value can act as a set-point for a new PID controller that is responsible for controlling the individual speed of each motors. One hypothesizes that combining

some form of speed controller with the implemented position controller will improve the behavior of the robot. However care must be taken to avoid convoluted code that plagued the Arduino 2018 code, as well as, how it should be implemented. For measuring the speed of the wheels one can use the encoders, but one need to consider how this should be integrated into the system.

7.3 Finding dynamic model of the robot

The result gave insight that the implemented position controller related closely to the dynamics of the robot, as observed by the paths formed. Therefore, finding a complete dynamic description of the robot open up the possibility of utilizing a wast amount of established theories and methods for making the controller optimal. This is perhaps the greatest strength of the controller as it is based on a series of sequential equations, where parameters found from the model can directly be transferred to the robot.

7.4 Sensors

The The Arduino LiDAR robot mention in chapter 2 have a magnetic compass. The PBC sheild on the Arduino IR robot have the supported connection for this sensor. This can be added to further improve the position estimate and make the robot less dependent on the inaccurate reading from the wheel encoders. From the IMU sensor one can also extract linear acceleration, which can be useful in the position estimation.

7.5 Collision detection and path planning

The background section mention the notion of artificial potential field for guiding the robot to the desired destination. From this, one can also make the use of repulsive filed for avoid objects. [10] An idea that have sprung up during the project, is to utilise the four IR sensor to detect objects in the surrounding and make the robot avoid objects while moving.

The data collected by the four sensors can be stored in some data structure and be used to build up a repulsive filed, similar to that describe in section 2.1. By combining both the attractive filed and the repulsive field one can use a form of gradient descent method to find an intermediate set-point coordinate for the robot. This set-point can then be used in the implemented position controller. By doing this, one can make the robot not only stops in the present of nearby objects, but searches for a new path and change the course accordingly.

7.6 Distributed system

As the development of the robot will continue and more functionalities will be added, the limited resource of the controller becomes a mayor concern. One idea is to add a second Atmeg2560 controller that runs in parallel to the current one. This way, the task can be distributed over two devices and achieve a true parallel system. There have been also mention in the past Petersen (Master 2019) [16] of utilising a programmable nRF52 dongle [15] that is used in the thread network to distribute the workload.

Bibliography

[1] Andy Wellings Alan Burns. *Real-time Systems and Programming languages*. Pearson Education Canada, 2005.

[2] *Atmega2560 datasheet*. https://www.freertos.org/wp-content/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf. Accessed: 2020-05-08.

[3] John Catsoulis. *Designing Embedded Hardware*. O'Reilly, 2005.

[4] Allen B. Downey. *The Little Book of Semaphores*. 2008.

[5] Erlend Ese. *Master thesis: "Sanntidsprogrammering på samarbeidande mobilrobotar"*. Accessed: 2020-05-08. 2016.

[6] Dimitris K. Manolakis John G. Proalis. *Digital Signal Processing*. Pearson, 2014.

[7] Hassan K. Khalil. *Nonlinear systems*. Prentice Hall, 2002.

[8] Kristian Lien. *Master thesis: "Embedded utvikling på en fjernstyrt kartleggingsrobot"*. Accessed: 2020-05-08. 2017.

[9] *LSM6DS3 datasheet*. https://www.st.com/resource/en/datasheet/lsm6ds3.pdf.

[10] Seth Hutchinson Mark W. Spong and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons Inc, 2005.

[11] *Mastering the FreeRTOS Real Time Kernel*. https://www.scilab.org/discrete-time-pid-controller-implementation. Accessed: 2020-05-08.

[12] Simen Robstad Nilssen. *Master thesis*. Accessed: 2020-05-08. 2018.

[13] Simen Robstad Nilssen. *Project: Arduino SLAM robot*. Accessed: 2020-05-08. 2017.

[14] *nrf51 datasheet*. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.1.pdf. Accessed: 2020-05-08.

[15] *nrf52 dongle*. https://www.nordicsemi.com/Software-and-tools/Development-Kits/nRF52840-Dongle. Accessed: 2020-05-08.

[16] Kristian Fjelde Pedersen. *Master thesis: Mesh Networking for IoT*. Accessed: 2020-05-08. 2019.

[17] *Picture Arduino*. https://store.arduino.cc/arduino-mega-2560-rev3. Accessed: 02.06.2020.

[18] *Picture LSM6DS3 Breakout*. https://cdn.sparkfun.com/r/500-500/assets/parts/1/0/6/4/6/13339-01.jpg. Accessed: 02.06.2020.

[19] *Picture nrf51 dongle*. https://www.nordicsemi.com/-/media/Images/Products/DevKits/nRF51-Series/nRF51-Dongle.png?h=588&la=en&mw=350&w=350&hash=F3CC9D49A52568A5C5C48D3027913AA36B9 Accessed: 02.06.2020.

[20] Andersen & Rødseth. *Master thesis*. Accessed: 2020-05-08. 2016.

[21] *SharpG2D12 datasheet*. https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf. Accessed: 2020-05-08.

[22] *Source code Arduino 2018*. Accessed: 01.01.2020.

[23] Neil Storey. *Electronics A system approach*. Pearson, 2013.

[24] Jannicke Tusvik. *Project*. Accessed: 2020-05-08. 2009.

.1 Rigid monition

The field of rigid motions seek to describe the different methods used for representing the orientation and position for rigid objects using points and vectors in Euclidean space. Among other things, describing the method for transformation between different coordinates frames using homogeneous transformation matrices. [10] In Cartesian coordinate system a point is described in three dimensional space using the coordinates [10]

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

The Euclidean distance between two points can be found using [10]

$$||\mathbf{p}|| = \sqrt{(x_2 - x_1) + (y_2 - y_1) + (z_2 - z_1)} \tag{2}$$

For finding the correct and unambiguous angle between a point and a fixed coordinate systems x-axis, as well as, the angle between two point one can use the atan2 function defined by [10]

$$atan2(x, y) = \begin{cases} \arctan(\frac{y}{x}) & \text{if} x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if} x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if} x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if} x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if} x = 0 \text{ and } y < 0 \end{cases} \tag{3}$$

which is defined for all $(x, y) \neq 0$ and returns the angle in the range $(-\pi < \theta \leq \pi)$. For representing relative position and orientation between two or more rigid bodies and their attached frames one can use homogeneous transformation matrix defined by [10]

$$H_n^0 = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \tag{4}$$

a point can then be described in the base frame given its position in the object frame as

$$\begin{bmatrix} p_n^0 \\ 1 \end{bmatrix} = H_n^0 \begin{bmatrix} p^n \\ 1 \end{bmatrix} \tag{5}$$

where

$$p_n^0 = R_n^0 p^n + o_n^0 \tag{6}$$

For rotation around a fixed z axis, one can define the basic rotation matrix [10]

$$R_{z,\theta} = \begin{bmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$