NTNU
Kunnskap for en bedre verden

Department of Engineering Cybernetics

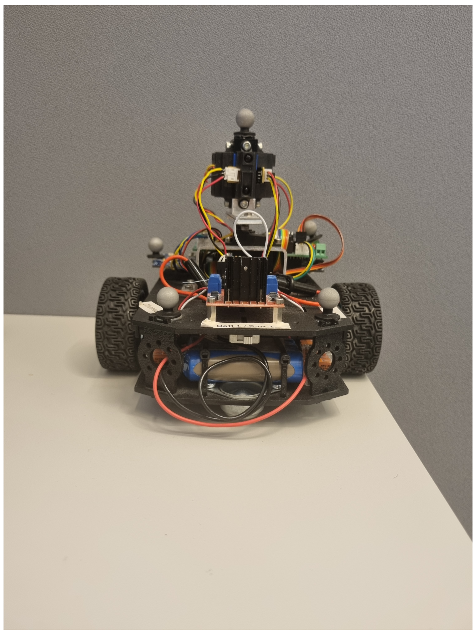
Master Thesis Spring 2023

Movement Control of Real-time Embedded Robot and Server System using MATLAB-generated C-code

Author:
Magnus Isdal Kolbeinsen

Supervisor:
Tor Onshus

Date: 12.06.2023



Preface

This thesis concludes the end of a 5-year studies of Cybernetics and Robotics. The thesis is the final 30 credit and the course TTK4900 Cybernetics and Robotics, master thesis.

To start my project I was given the software and hardware of the nRF52840 robot, java server software along with a nrf52 Dongle, and access to OptiTrac motion capture all at Gløshaugen Elektro D+B2: B333.

I would like to thank everybody involved with the project. This includes all student colleagues and persons at ITK workshop. I would also like to thank Ellen Beate Hove for help with a week extension of deadline on such short notice.

Moreover, I would also like to thank my supervisor Tor Onshus for always being available for guidance and help when it was needed.

1 Problem Description

The main goal of this project is to implement a satisfactory control system that is able to take commands from the previously implemented java server and travel to the given destination. Another part of this project was to implement the control system using MATLAB to generate C code for the robot. Additionally, a side goal of this year's students was to make a Github Organization with all necessary code to run the robots.

This project is a continuation of the specialization project Kolbeinsen 2022. Some of the imperfections that was identified in the specialization project needs to be improved:

- The robot's ability to move straight and not diverge.
- The robot's ability to reach its end destination after several commands.
- The robot had not yet been fully integrated with the server.

The project can be divided into the following subtasks:

1. Contribute to moving code from SLAM project to Github organization
    - Code cleaning, make naming convention
    - Make overview of code
    - Contribute to wiki about software
2. Develop movement control system in MATLAB and generate C code.
    - Fix issues revealed in specialization project.
    - Implement slowdown algorithm.
    - Implement IMU into the controller.
    - Implement PID control.
    - Tune PID controller.
    - Test the system performance.
3. Modify vApiTask() to work with MATLAB generated Code.
    - Initialize all required variables.
    - Calibrate gyroscope measurements from IMU.
    - Make sure it is compatible with the java server.

2 Summary and Conclusion

The system is a two wheeled robot that communicates with a java server. It takes set point commands from the server and travels to the given location. This master thesis was a continuation project from the specialization project Kolbeinsen 2022. The master thesis contains necessary information about how the robot system works and about the implementation of a new movement control system.

MATLAB was used to generate C code. controller_api() is the MATLAB generated function that calculates the input for each wheel robot and estimates the position of the robot. controller_api() is a function that contains an estimator() and regulator(). To control the robot towards a setpoint it, the robot will first rotate towards the target and then move straight. Two core additions to the system are the implementation of gyroscope readings to the rotate_robot() function and a PID regulator in the move_forward() function of the regulator(). Additionally, a focus has been put on software quality to make it easier to understand and maintain in the future.

vControllerApiTask(), previously called vApiTask(), has been modified such that it works with the java server. All the variables required to run the new MATLAB generated code have been initialized and works together with this task. The gyroscope readings have been calibrated such that it does not drift, but this calibration implementation stands as a possible source of error.

The implemented system have been tested during development. These tests includes; initial tests for gyroscope and system, integration tests for PID controller and server integration tests for the whole system. Ground truth pose was captured with Opti Track Motive and test procedures such as line tests and square tests were conducted.

Contributions have been made to the Github Organization. Collaborative work includes code cleaning sessions and discussions about name conventions and the development of a documentation wiki. Moreover, the newly developed matlab-robot-code has been uploaded as its own repository to the organization. It contains a README for setting up the MATLAB robot, relevant reports, robot-application C code and MATLAB files developed in this project.

The robot are now able to move in a straight line and can be sent several commands in succession with good accuracy. This is due to the implementation of a PID controller for moving straight and gyroscope measurements for feedback while rotating. However, because the robot has no aiding measurement, it will over time lose track of where it is. This is especially the case if the robot moves in a path involving several curves or slips on the wheels.

3 Sammendrag og konklusjon

Systemet er en tohjulet robot som kommuniserer med en java server. Den tar imot posisjonskommandoer fra serveren og beveger seg mot den lokasjonen. Denne masteroppgaven er en fortsettelse på spesialiseringsprojectet Kolbeinsen 2022. Rapporten inneholder nødvendig informasjon om hvordan robot systemet fungerer og om den nye implementasjonen av bevegelseskontrollsystemet.

MATLAB generert kode er brukt. controller_api() er den MATLAB genererte funksjonen som kalkulerer pådrag for hvert av de to hjulene og estimerer robotens posisjon. controller_api() inneholder en estimator() og en regulator(). For å styre roboten mot et mål, vil den først rotere mot målet for så å bevege seg rett. Implementasjonen av gyroskopmålinger i rotate_robot() og PID-regulator i move_forward() er viktige deler av utvkilingen i denne rapporten. I tillegg, har det vært satt fokus på programvarekvalitet slik at det blir enklere for nye utviklere å vedlikeholde koden.

vControllerApiTask(), tidligere kalt vApiTask(), har blitt endret slik at den virker sammen med java serveren. Alle variabler som hører til controller_api() har blitt initialisert og virker sammen med denne tasken. Gyroskopmålingene har blitt kalibrert slik at de ikke drifter når man leser av, men implementasjonen av dette er en mulig kilde for feil.

Det implementerte systemet har blitt testet. Disse testene inkluderer; innledende tester for gyroscope og system, integrasjonstester for PID-regulator og server integrasjonstester for hele systemet. Nøyaktig posisjon har blitt logget med Opti Track Motive og test prosedyrer bestående av firkant tester og linje tester har blitt utført.

Bidrag har blitt gitt til GitHub Organisasjonen. Sammen med de andre studentene har det blitt utført koderengjøringsøkter, diskusjoner om navn konvensjoner og utvikling av en wiki. Videre, har den nye matlab-robot-koden blitt lastet opp som et eget repository i organisasjonen. Det inneholder en README-fil for å sette opp matlab-roboten, den modifiserte robot-applikasjonskoden i c og MATLAB-filer utviklet i dette prosjektet.

Roboten klarer nå å bevege seg i en rett linje. Den kan også motta flere kommandoer på rad med god presisjon. Dette er grunnet ny implementasjon av en PID-regulator for å kjøre rett frem, og gyroskopmålinger til estimator ved rotasjon. Siden roboten ikke har noen måte å rette opp posisjonsestimatet hvis den begynner å drifte, vil den over tid miste oversikten over hvor den er. Dette er spesielt tilfellet hvis roboten tar en sti til målet bestående av mange kurver eller sklir på hjulene.

Table of Contents

1 Problem Description **ii**

2 Summary and Conclusion **iii**

3 Sammendrag og konklusjon **iv**

List of Figures **vi**

List of Tables **vii**

4 Introduction **1**

5 Previous Work **2**

6 System Overview **4**

7 Theory **10**

7.1 GitHub Organization 10

7.2 Opti Track 11

7.3 Real-time Embedded System 12

7.4 Software Quality 13

7.5 PID Control 15

8 Robot Project Collaborative Work **16**

9 Method and Implementation **24**

9.1 Test Setup and Environment 24

9.2 Software Quality 27

9.3 Movement Control System Design 28

9.4 Movement Control System Implementation 30

9.4.1 controller_api() 30

9.4.2 estimator() 31

9.4.3 regulator() 34

9.4.4 rotateRobot() 38

9.4.5 moveForward() 39

9.4.6 vControllerApiTask() 41

10 Results **42**

10.1 Initial Tests 42

10.2 Initial Testing Gyroscope 44
10.3 Integration Tests Gyroscope and Slowdown Algorithm 45
10.4 Integration Test P-regulator 47
10.5 Integration Testing PI-regulator 48
10.6 Integration Testing PID-regulator 50
10.7 Integration Test Server 54

11 Discussion 56

12 Further Work 59

Bibliography 61

Appendix 62

A README-files Github Organization 62

A Matlab-robot-README 62
B EKF-SLAM-README 63
C Python-robot-control-README 64

B vControllerAPiTask() 65

C vControllerApiTesterTask() 68

List of Figures

1 Robot server system. 4
2 Top view robot. T. Andersen 2022 5
3 Bottom view robot. T. Andersen 2022. 5
4 Side view robot. T. Andersen 2022. 6
5 General system overview after specialization project. Kolbeinsen (2022). 8
6 One of the camera used to capture 6DOF frame. 11
7 NRF3 Opti Track Motive Software. 11
8 PID controller 15
9 Zip file handout structure. 18
10 First GitHub repository structure. 19
11 Github Organization structure 21
12 README file Java Server. 23
13 Testing environment in project.Room B333. 24

14 Settings used for tracking in Opti Track Motive. 25
15 OLED display used for debugging while driving without USB. 26
16 Overview of the system today. 28
17 Overview of controller_api() funciton 29
18 Robot moving forward. 31
19 Robot rotating, turning. 32
20 Robot pose and orientation towards target. 35
21 Moving foward 39
22 300 mm Position Square Test. 42
23 Initial Position Line Test. 43
24 Plot of angular velocities from gyroscope while standing still. 44
25 Gyroscope and Slowdown Algorithm Position Square Test 300 mm. 45
26 Position Line Test 1000 Slowdown Algorithm and Gyroscope. 46
27 Position Line Test 3000m P Regulator 47
28 Position Line Test 3000mm P = 20. PI-regulator 48
29 PI-reg plot Zoomed at end destination. 49
30 PID regulator instability with derivative Term. 50
31 Line Test Position PID-regulator P = 20, I=10, D=0.5. 51
32 Line Test Position PID-regulator P = 20, I=10, D=0.5 Zoomed. 52
33 Square Test Position PID-regulator P = 20, I = 10, D = 0.5. 53
34 Server integration test 3000mm. 54
35 Server integration test square test 1000mm. 55

List of Tables

4 Introduction

Robots in industries are becoming more and more common. A robot that can move on its own can be an asset for humans in the future. By sending a robot into a hazardous locations that does not need to be controlled by a human, you can reduce the risk and danger for humans.

The SLAM-project, short for simultaneous localization and mapping, was started in 2004 by the Department of Engineering Cybernetics. Initially, it was called the LEGO-project since the robots were assembled using LEGO MINDSTORM kits. However, in later years the hardware has changed and now nRF52840 chips alongside new electronics are being used. The SLAM-project is carried on by Tor Onshus and is often used as topics for specialization projects and master thesis in Department of Engineering Cybernetics at NTNU, Gløshaugen. The end goal of the SLAM-project is to make multiple robots work together to create a map of the environment around them.

This master thesis is an continuation of the specialization project in fall 2022. The focus of the thesis is about making a satisfactory control system of the matlab-robot. Another focus of the thesis is to further explore the possibilities of using MATLAB to generate C code. The motivation behind using MATLAB, is to be able utilize MATLAB functions library and thus you make it easier to develop more complex systems. Moreover, this is reason why the robot is referred to as the matlab-robot.

The thesis contains information about the development of the movement control system of the robot. The previous work section contains a brief description of what modules were already implemented. The system overview section contains information about the hardware, firmware and software and limitations of the robot. The theory section briefly introduces topics that discussed in the report. Robot project collaborative work shows the work that has been done towards improving the overall SLAM-project. It describes the collective effort of the group of this year and the individual contributions. Method and implementation contains information about development tools, test environment and implemented code for the movement control system. Results aims to show the process of testing the system. It contains intial testing, integration testing for the PID with gyroscope measurement and server integration testing. The discussion section contains reflections about the system and its development. Further work aims to describe what can be done to improve the system and alternative ways.

5 Previous Work

From the start of the SLAM project, students have contributed to its development. The main parts of the project that relates to this project are; the robot, the java server and the robot control system in MATLAB.

The Robot

The current hardware version of the robot was developed by Eivind Jølsgård, Jølsgård 2020. He built three robots for the SLAM project (NRF1,NRF2,NRF3) that was based on nRF52840 development kit. A brief overview of the structure of the robot hardware will be described in System Overview later in the report.

Before starting this project, a functional nRF52 robot was given. The robot had implemented drivers for IR Sensors, IMU, wheel encoders, OLED display and DC motors into software. However, some hardware flaws had to be fixed in the specialization project. This mainly included the left wheel encoder and IR sensors.

Furthermore, these were integrated into a program for running the robot. The program had a structure for communicating with each hardware component and included tasks such as; *display_ task, microsd_ task, vApiTask,vMainSensorTowerTask, vMainCommunicationTask, vARQTask.* A brief overview of how the system software will be described in System Overview later in the report.

Java Server

The java server is a server that contains a user interface to send commands to the robots and monitor them. It also creates a map based on the IR sensor output and estimate from the robots. It also contains a simulator that can be used to simulate robot behaviour.

The Java server application was created by Rødseth and T. E. S. Andersen 2016. It is based on a previous implementation in MATLAB that was first introduced in Syvertsen 2006. The simulator inside the Java server was created by Thon 2006. The BLE communication to the server was developed by Ese 2016. Later the a new version of BLE was implemented by Gilje 2022 on the matlab-robot.

Robot Control System MATLAB

The specialization project was a continuation of Gilje 2022 control system. Her code was based on Berglunds control system, Berglund 2021. It included regulator.m, estimator.m, managerTask.m, api.m. A few issues was identified with Gilje's MATLAB implementation:

- Not able to perform multiple commands.
- Robot would rotate around its own axis after doing one command.
- Robot diverged from a straight line.

In the specialization project these issues were addressed and the control system was remade. To fix these issues, a restructuring and modification of the implemented MATLAB generated C code and in the vApiTask() functions was completed.

Specialization Project

Although the specialization project fixed some of the issues, some remained or was discovered.

- Robot would diverge from a straight line over longer distances.
- Accuracy of rotation controller was limited.
- Robot was not fully integrated with the robot server.

The control system from the specialization project did not utilize IMU signals for estimation and did not have an implemented PID regulator for moving forward.

6 System Overview

This section briefly describes the system. It contains information about robot hardware, robot firmware, robot software, java server and known limitations.

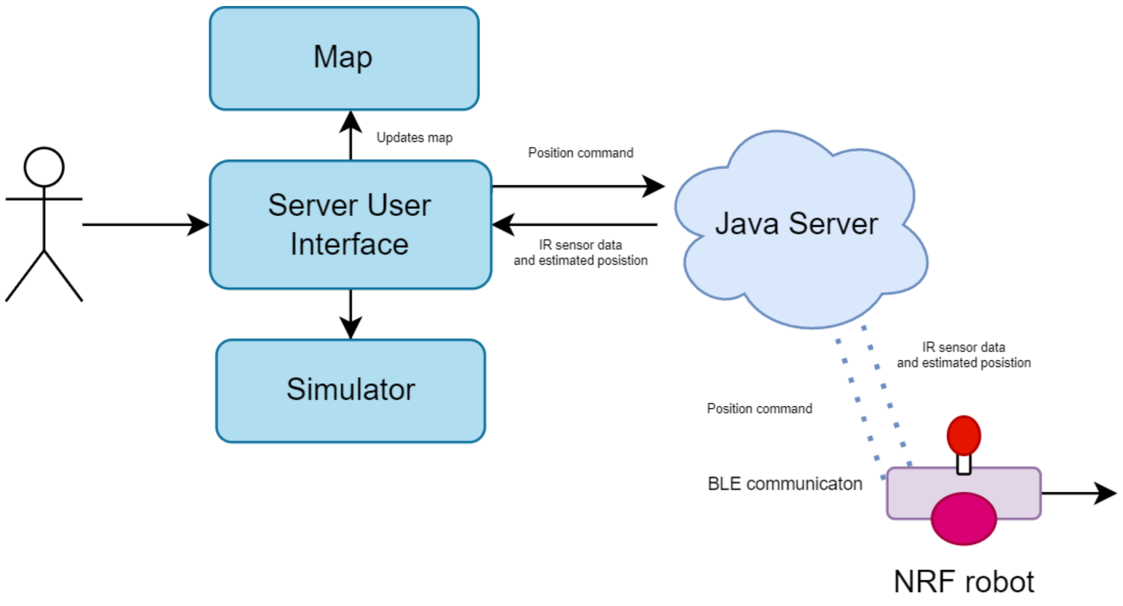


Figure 1: Robot server system.

Figure 1 shows how the user interacts with the system. The user will open the java server. It contains a user interface for connecting and controlling the robot. After connecting the robot with the java server, the user is able to send position commands to the robot. The robot will first rotate towards the target and then move to that position while simultaneously sending its estimated position and IR sensor data back to the server. The server will then try to make a map of this data. The server also contains a simulator.

Robot Hardware

A brief overview of the robot hardware is described in this section. A more detailed description of the robot hardware can be found in Jølsgård 2020. Figure 2, Figure 3 and Figure 4 shows the different components of the robot.

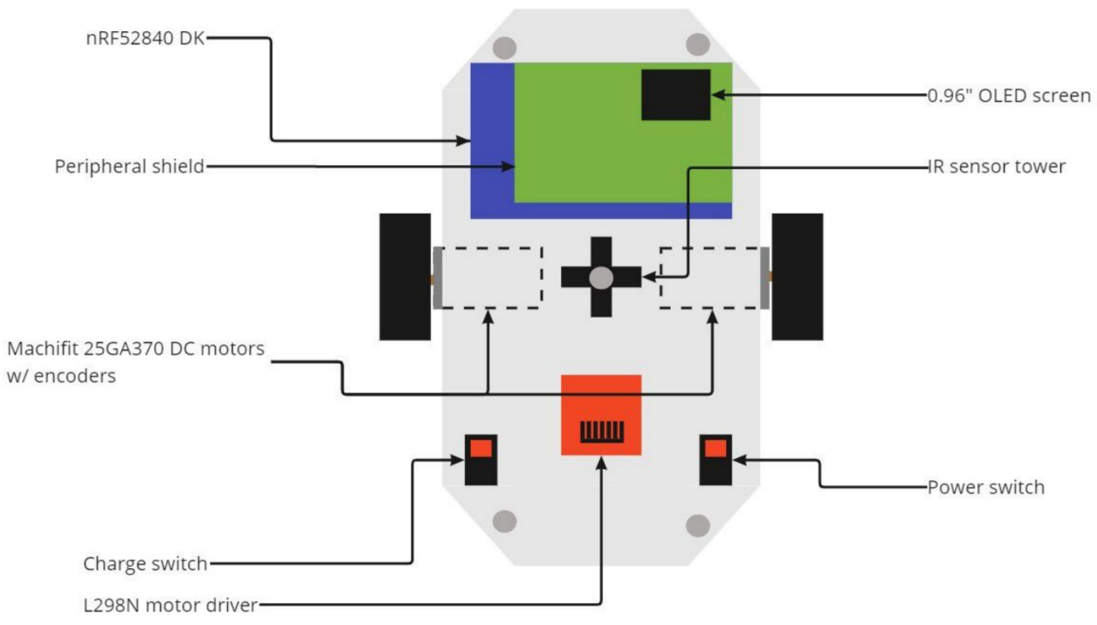


Figure 2: Top view robot. T. Andersen 2022

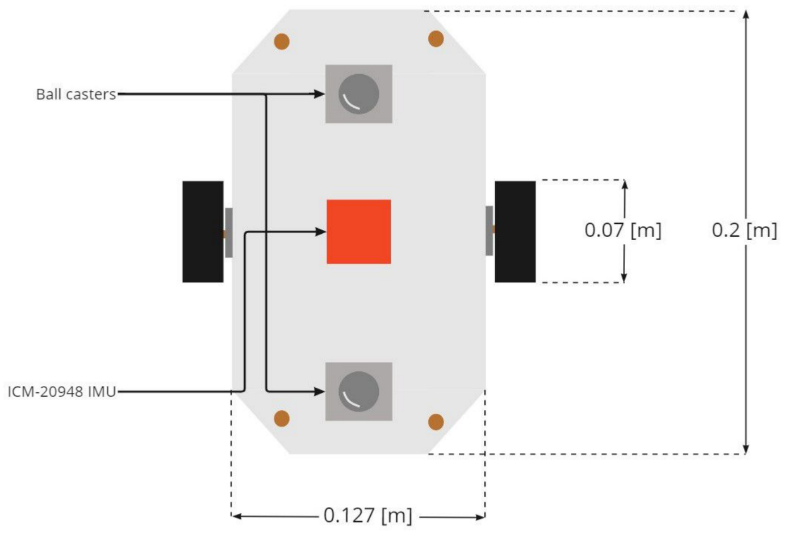


Figure 3: Bottom view robot. T. Andersen 2022.

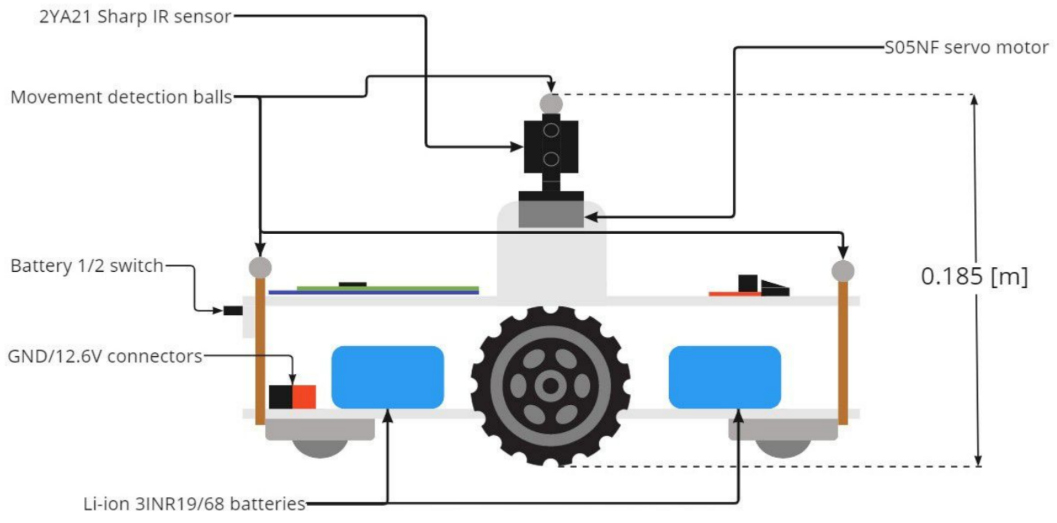


Figure 4: Side view robot. T. Andersen 2022.

The robot contains :

- 4 x IR Sensors
- 1 x Servo Motor
- 2 x DC motors
- 1 x Motor H bridge
- 1 x IMU
- Movement Detection Balls
- nRF52840 DK
- Custom NRF3 hardware shield
- 2 x 2600 Ah Batteries
- 1 x OLED Display

The IR sensors are mounted together in a tower on top of the robot. They are mainly used by the server to map the surroundings of the robot, but are also used for collision avoidance and obstacle detection.

The servo motor is used to rotate the IR sensor tower in order to gather more IR sensor data. The configuration of the wires to the sensor tower does not allow for a rotation larger than 90 °.

The DC motors are used to rotate and move the robot. They contain encoders that measure angle displacement and angular velocity.

The H bride motor driver is used to allow for bi-directional control of the DC motors.

The intertial measurement unit (IMU) contains a gyroscope and an accelerometer. The gyroscope measures angular velocity and the accelerometer measures specific force. It is mounted under the robot.

The movement dectection balls are not used by the robot, but attached to the robot so that it can be detected by Opti Track Motion Capture software.

The nRF5840 Development kit (DK) is a development kit developed by Nordic Semiconductor. It is a programmable chip that makes it possible to control digital pins and flashing of C code.

The custom hardware shield was developed by Eivind Jølsgård. It is placed on top of the nRF52840 DK and connects the nRF52840 DK to the hardware such as the DC motors, servomotor and IR sensor tower.

The OLED display is mounted on top of the hardware shield and can be used for debugging.

Robot Firmware

The robot firmware includes nRF5 Software Development Kit version 15.0.0 and the SoftDevices s140 version 6.0.0. The SDK firmware has implemented functions that makes it easier to control the nRF52840 chip.

Robot Software

An important standard library is the freeRTOS library. It is a real-time operating system library that allows for task schdeuling for microcontrollers and microprocessors. It is distributed freely under MIT open source licence.

The software that has been developed by students over the years are:

- Drivers
- vARQTask
- api()
- vApiTask
- MainCommunicationTask
- MainSensorTowerTask

Figure 5 shows a general overview how the different modules interacts with each other.

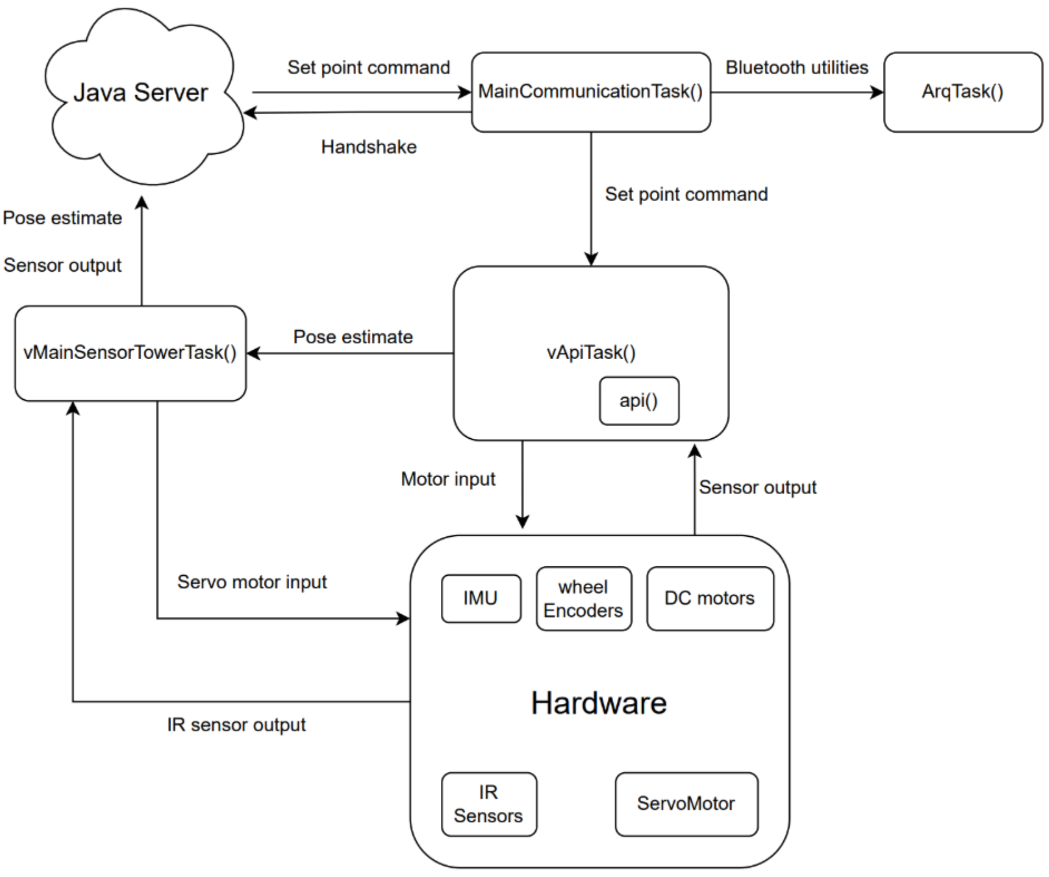


Figure 5: General system overview after specialization project. Kolbeinsen (2022).

The driver software includes motor drivers, OLED-display and readings of IR sensors, wheel encoders and IMU.

vARQTask are communication software contains utility functions for communicating over BLE.

The api() function is a MATLAB generated C code function that was given before the specialization project started. It has now been renamed to controller_api(), to make it intuitive what it does and reconstructed.

The vApiTask is a task that gathers sensor data and sets the controller output to the motors. It has now been renamed to vControllerApiTask() and reconstructed.

vMainCommunicationTask is a task used to maintain connection with the Java Server.

vSensorTowerTask is a task to control the IR sensor tower appropriately. It gathers data from the IR sensors and forwards pose estimate to the Java Server.

A small guide on how to setup the robot can be found in the specialization project Kolbeinsen 2022.

Java Server

The Java Server code contains software for setting up a server that can send and recieve data to the robots over BLE.

To start the server, you need the nRF52-dongle, NetBeans IDE, java development kit (JDK), java run time environment (JRE) and the java server source code that can be found in the java-server repository in the GitHub organisation.

A small guide on how to setup the Java server can be found in the specialization project Kolbeinsen 2022, or by following the steps in the README-file on the java-server GitHub repository. The GitHub repository also contains a manual made by the creators of the java server.

Limitations and Known Problems

The NRF3 robot has a known hardware bug where it wont start without a kickstart from the USB. In other words, if you click on the power button alone, you will not start the robot code. To start the robot code you will have to turn the power button ON and be connected to USB. However, it is able to maintain power after first starting the program once, and will not shutdown when you click the reset button.

There is a known issue with the connection with the java server. It will disconnect abruptly, and the reason behind it is unknown.

One of the limitations with this project is the robot hardware. The control system that is dependent of the accuracy of the hardware provided. Small inaccuracies of sensors, computation delay and other physical constraints will impact the system behaviour.

MATLAB generation is another limitation of this project. There are differences between MATLAB and c that makes it impossible to generate certain functions to C. This will require workarounds. An example of this was in the specialization project, where a custom modulus function had to be created because the mod() function did not have extended capabilities for C code generation due to floating point differences.

7 Theory

7.1 GitHub Organization

A GitHub organization (GHO) is a container for shared work. It typically contains multiple repositories and gives the work a unique name and brand. GHOs allow administration of multiple teams and repositories Github 2023.

Within an organization you assign different roles to collaborators. Owner, billing manager, security manager, app manager, member, moderator, outside collaborator. This makes it easier to control access in the organization. Another feature of a GHO is the teams feature. It is a feature that can be used to easier manage permissions for certain groups in the organization. Teams have dedicated pages where you can see team members, team profile, child teams and so on. It may be very helpful in larger organizations as it allows for private discussions within the teams. In smaller projects however, it may not be as interesting if everyone needs to be aware of everything.

One of the biggest benefits of using a github organization is that it more easily allows for modularity. By dividing a project into repositories that makes sense, it is easier to work on different modules. An example would be to separate a server from a robot code. If you are going to work on the server, you do not need all the robot code and vice versa. It also makes it easy for newer members to locate what is needed.

Another feature that a GitHub organization provides is the GitHub wiki. It can be used to provide extra details that you do not want to add to the README-file. This feature is only available for GitHub Free users if the repositories are public and would require a pro version if you want a private repository.

Good practises for managing a GHO is plan of continuity, use of teams, MFA for all, backups, reviewing access and code, control periodically, tracking progress and keeping lean git branches. Plan of continuity involves having atleast 2 owners ensure that the organization is able to maintain ownership continuity if one of the owners lose access. MFA is a security measure (multi-factor authentication) for all members. This is to prevent malicious attacks towards the organization. Backups of important data as a mean of a recovery plan for critical repositories. Reviewing access and code is the means of regulating changes and who has access to what. This is important for security. Tracking progress involves creating a way to track improvements and bug fixes in the project. A feature called GitHub project board can be a way to do this. This feature is a Kanban style project board to track what is being worked on. Keeping lean branches means having people work on dedicated branches for each feature or bug. This is to avoid merge conflicts and overwrites Segura 2023.

7.2 Opti Track

Opti Track Motive is a motion capture system based on camera vision. NTNU has built a Opti Track system in B333 at EL bygget. It contains a set amount of cameras that are integrated with the Opti Track Motive software. It tracks six degrees of freedom (6DoF) data of objects in real time by identifying the movement detection balls on an object Motive 2023b. The positional accuracy of the system is $\pm 0.2mm$ Motive 2023a.

In picture Figure 6 you can see one of the cameras in B333. The cameras are mounted close to the ceiling in a square shape pattern. In Figure 7 you can see how the program looks after creating a body.

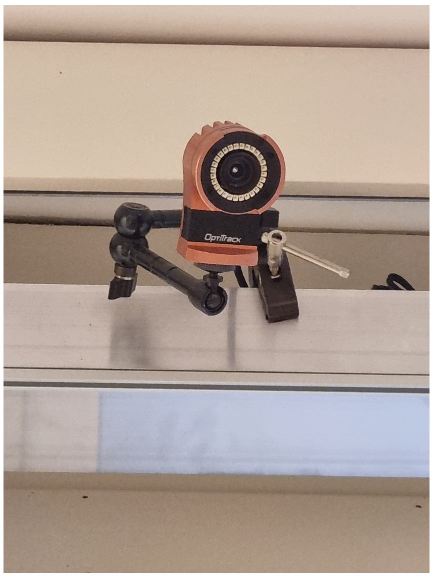


Figure 6: One of the camera used to capture 6DOF frame.

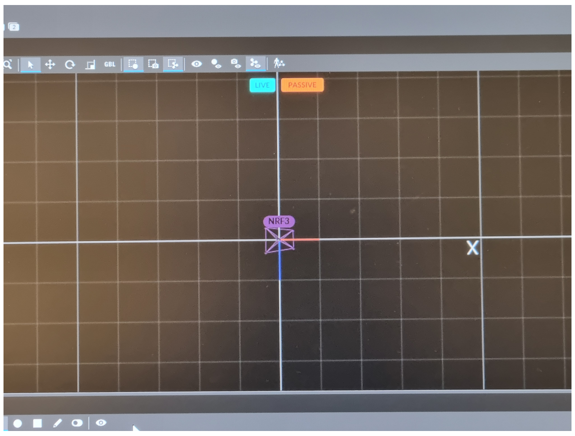


Figure 7: NRF3 Opti Track Motive Software.

7.3 Real-time Embedded System

The robot system is an embedded system. It is a combination of computer hardware and software designed for a specific function. The function is to move the robot around and ultimately map its environment.

The system is also contains RTOS (Real Time Operating System), speficially freeRTOS, which is used to schedule its program execution. It processes information from sensors, performs real time functions and responds to events. This makes the system a real-time embedded system.

There are different categories of real-time embedded systems; Hard, where all timing constraints must not be violated and no errors are acceptable. Examples of this are airplane sensors and self driving control systems. Firm, where violation of timing constraints can happen occasionally, but is undesirable. It loses performance of such failures. Examples of these are manufactoring robots. Soft, where deadlines can be exceeded. Although response failures diminishes user experience, it does not reduce the overall performance. An example is a TV router box Sakovich 2023.

Based on these definitions, we can define the robot system as a firm or hard real-time embedded system. A reponse failure may damage the system an its overall performance. If the system fails to meet the deadline for bluetooth connection with the server, it requires a manual reconnection to the server. It is also dependent of its sensor to meet the deadlines otherwise the system performance will be damaged.

Some of the challenges of developing real-time embedded systems are; physical constraints, timing constraints, task scheduling. Physical constraints may include size of the device, spatial constraints, limited memory, power consumption and environmental conditions during operation. Timing constraints refer to the specific times where the system has to respond to certain instances. An example of this in this system is the bluetooth connection with the server. It has to be maintained, or it loses the connection with the server. Task scheduling is about how the system organize dataprocessing. Some systems has the ability to perform tasks in parallell meaning they can happen at the same time. However, this requires multiple cores and the robot system does not have this ability. Instead, it uses concurrency to optimize its scheduling.

7.4 Software Quality

Software qualities are measurements of how good a program operates. There are two main categories of software quality, external and internal quality. External software quality entails the characteristics that the user of the program are aware of McConnell 2004, p.463-465:

- **Correctness** The degree of which a system is free of faults in its specification, design and implementation.
- **Usability** How easy it is for users to learn and use the system.
- **Efficiency** Execution time and use of system resources.
- **Reliability** The systems ability to perform its required functions under stated conditions.
- **Integrity** Security of operation and validity of information that it provides.
- **Adaptability** The extent of which the system can be used in an environment or applications that it was not initially designed for.
- **Accuracy** The degree of what the system outputs error-free quantitative data.
- **Robustness** To what extent will the system continue to operate if it receives invalid outputs or stressfull environmental conditions.

Internal software quality on the other hand focuses on the internal characteristics that the developer are aware of:

- **Maintainability** The level of difficulty to improve the system performance and add new features.
- **Flexibility** The level of flexibility to modify the system for other use that its original design specifications.
- **Portability** Measure of how much is required to modify the system to operate in other environments that it was originally designed for.
- **Reusability** A measurement of how easy it is to use parts of the system for other systems.
- **Readability** A measurement of how easy it is read and understand the source code.
- **Testability** A measurement of how easy it is to test the system and verify that it meets its requirements.
- **Understandability** A measurement it to comprehend the system at organizational level and detailed level. Is the system cohesive?

Both internal and external software qualities are important for a system to make a good system. The qualities may overlap sometimes by improving one criteria you also improve another one. An example would be to improve the readability of the code and by doing so, you also improve its maintainability. Even though the concepts are not exactly the same, they are intertwined. Moreover, it is impossible to make a system that has high qualities in all the criteria because there is a trade-off when emphasising certain qualities. For example, in general when improving correctness of a system, the robustness decreases. It is useful to think about these characteristics when creating or improving a system whether they are mutually beneficial or antagonistic.

Naming Conventions

Naming conventions entails names of variables, routines, constants, macros, types, objects and classes. It is one of the most effective ways to improve readability and understandability of software code, and consequently, maintainability. The reason for this is that it lets you take more for granted. An example would be good naming conventions on global variables. This makes you able to take a decision of the system variable without reading much more of the code. Another benefit is that it makes it easier for new developers to work on the same source code. This is especially the case when multiple developers have been working on the same project. Instead of learning the coding style of multiple, you have conventions to make the code more consistent. It also makes it less likely that you are duplicating variables, and at last it can compensate for weaknesses in the language.

Some informal guidelines McConnell 2004, p.259-280 to naming conventions are:

- Focus on the problem rather than the solution.
- Avoid ambiguous names.
- Use longer names rather than shorter.
- Stay consistent with the conventions.
- Give Boolean variables names and use positive Boolean variable names.
- Differentiate between routine names and variables names.
- Identify global variables.

Types of names to avoid:

- Misleading names
- Names with similar meanings
- Names with different meaning but that are similar
- Names that sounds similar
- Names with numerals
- Commonly misspelled names
- Names with hard-to-read characters

Self-Documenting Code

Another important concept for software quality is documentation. There are both external and internal documentation. External documentation entails methods that are used to explain the system without the source code. Examples of external documentation methods are doxygen, UML, flowcharts and user manuals. Internal documentation entails commenting inside the code and McConnell 2004, p.777-795 writes about the concept of self-documenting code. This is a concept where you write the code in a way that it requires little to no documentation to understand it. To make a code base easily maintainable it should require little external documentation and document itself by the way it is written.

A few important guidelines to make self-documenting code are:

- Can someone look at the code and easily understand what it does?
- Do comments focus on how rather than why?

- Are comments clear and correct?
- Have magic numbers been replaced with named constants or variables instead of being documented?
- Is the purpose of each routine commented?

7.5 PID Control

A proportional integral derivative (PID) controller is a regulator used to control a process variable. It combines proportional control with integral and derivative adjustment to compensate for state changes in the system (Instruments 2023). Its purpose is to force feedback to match a setpoint and are therefore best at regulating systems that can respond quickly to control input.

A continuous PID controller can be represented like this:

$$u(t) = K_p e(t) + K_i \int e(t)dt - K_d \frac{de}{dt} \tag{1}$$

where, K_p,K_i and K_d represent the controller gains, u represents the control signal and e represents the error.

$K_p e(t)$ is the proportional part of the controller. This term will affect the control signal with a proportional gain K_p that is multiplied with the error. By increasing this term the controller becomes more aggressive by making it react more quickly to errors, but it can also cause overshoot.This term also tends to reduce steady state error, but it will not eliminate it.

$K_i \int e(t)dt$ is the integral part of the controller. It is created by adding up the steady state error over time and multiplying this integrated error by the integral gain K_i. This term can fix persistent errors by integration of the error and thereby driving the steady state error down. This can however, also cause a more sluggish system respone because it can take a while to unwind if the error changes sign.

$K_p \frac{de}{dt}$ is the derivative part of the controller. This term is created by comparing the error from a previous time in the system by the current one. This way it can help the system anticipate error. It acts as a dampening effect by reducing the control signal when it experiences large changes in error.

A typical way that a PID controller affects a system is depicted in Figure 8. A setpoint r is substracted from a measurement y, and sent as input into the PID controller which output u into the system.

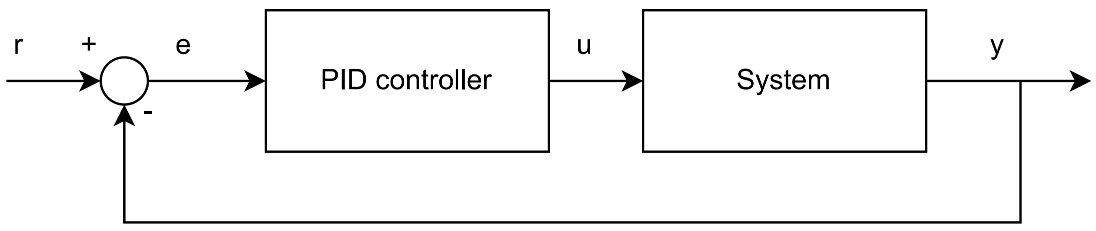


Figure 8: PID controller

8 Robot Project Collaborative Work

While working with the robots, it was decided that the students working on it should work together in order to improve the project. The participants were Emanuela Tuong Vi Thi Tran, Magnus Isdal Kolbeinsen, Marte Nordbotten Ruud Olsen and Kristian Forsdahl Berg.

My individual contributions to the GitHub organisation:

- Created repositories and uploaded code for java-server, EKF-SLAM, python-robot-control and matlab-robot-code.
- Created of README-files for GitHub Organisation repositories for java-server, python-robot-control and matlab-robot-code.

Below is a a text that was written together by all participants and it describes the work that has been done as a collective.

> “The four students working on the SLAM Robot project this semester, Spring 2023, have made an attempt to improve the overall quality of the project. The goal through this collaborative effort has been to ease the process of any future students, and to make the code easier to develop and maintain.
>
> The project contains large amounts of code with little to no regulation of code quality. Consequently, it has been regarded as hard to comprehend for newer students. This has in turn affected the progress of the SLAM-project over time. Ideally, new students should be starting on the project where the previous students left off, but this has not been the case. Students have reported to have spent a lot of time in the initial stage of learning how the robots work, which could be prevented if necessary information was easy to find.
>
> There is a large amount of information available regarding how the code works with explanations of certain features. However, there are no overview of which theses contain what information. To get a full understanding of the current code, you would have to read through many theses to figure out which ones are relevant. Additionally, it was up to each person to decide how to document their code and to write the code in any way they preferred. This lack of structure has made it unnecessarily difficult to understand the code.
>
> Upon completion of the project students have submitted their work as a zip file containing their entire project. As multiple students were working on different parts of the robot, multiple zip files of code were delivered separately with necessarily no designated main project. The following semester, new students would be presented with the question as to which project to base their work with, leaving documentation and work from the other prior projects behind. As follows, the general project would miss out on useful features and bug fixes throughout the revisions.
>
> Utilizing a GitHub organization, would open for version control and collaborative work on the projects, with the possibility of creating new repositories for experimental work, and merging these into the main project if they are decided to become a main feature. This way, new students will always have one place to look for their project material, and all documentation and code features will be preserved and accessible down the line.

Listed below are the work that has been done to improve the project and its workflow:

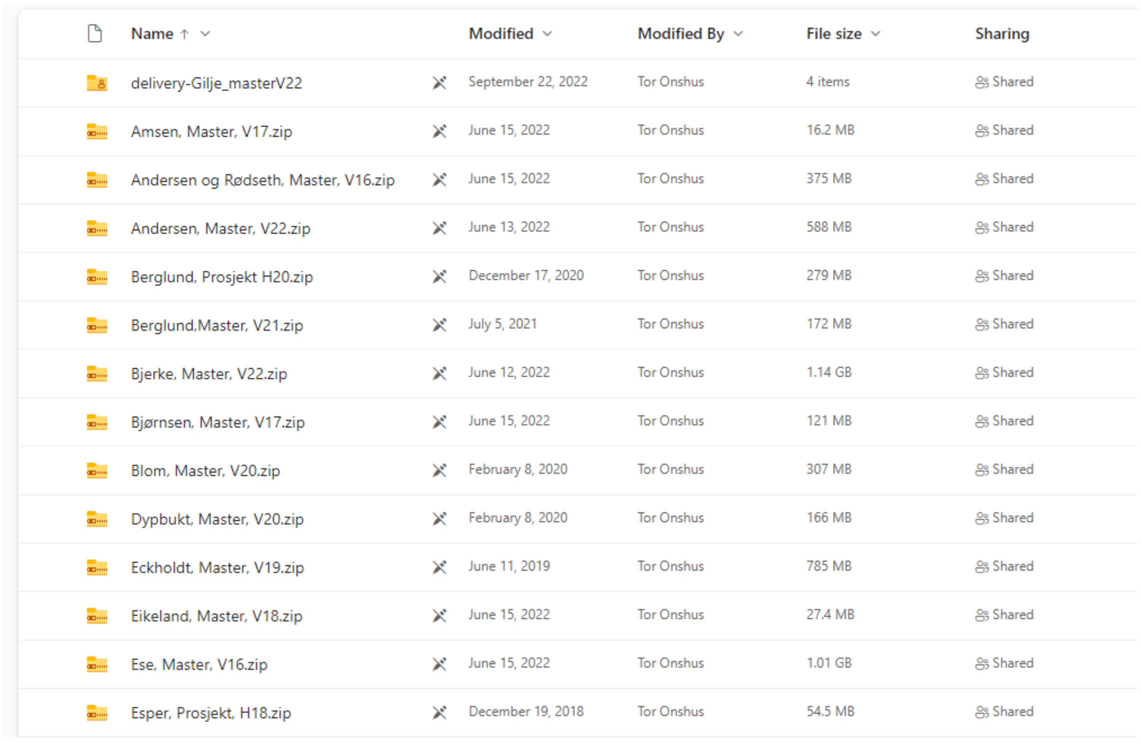
- Weekly meetings to update on workflow and the state of the project.
  - Updates and discussion about the collaborative work.
  - Updates on progress of individual projects.
  - General questions about the project.
- Set name conventions for the software running on the robots' nRF52840DK.
  - Change the code to comply with the name conventions.
  - Deleted unnecessary comments.
- Changed the file structure of the project.
  - Changed file names.
  - Divided the project into parts.
  - Changed the path of software running on the robots nRF52840DK.
- Created a Github organization.
  - Added the projects code to Github.
  - Created git ignores.
  - Deleted unnecessary and unused files in the project.
  - Created repository name.
  - Added README files to each repository.
- Created a Wiki for documentation of the project.
  - Added relevant project an master theses that has contributed the project.
  - Added information of the project in parts such as: software, hardware, known bugs etc.
  - Added HowTos:
    - How to start and end the project after a work period or semester.
    - How to create a new software project with FreeRTOS.
    - How to edit the Wiki.

Emanuela Tuong Vi Thi Tran, Magnus Isdal Kolbeinsen, Marte Nordbotten Ruud Olsen and Kristian Forsdahl Berg ”

Github Organization

The GitHub organization SLAMRobotProject was a new addition this year. It was made in order to make it easier for newer students to get familiar with the robot project. Previously, you would get access to a large amount of master thesis' zip files and had to download and read through them to find what was relevant for you.

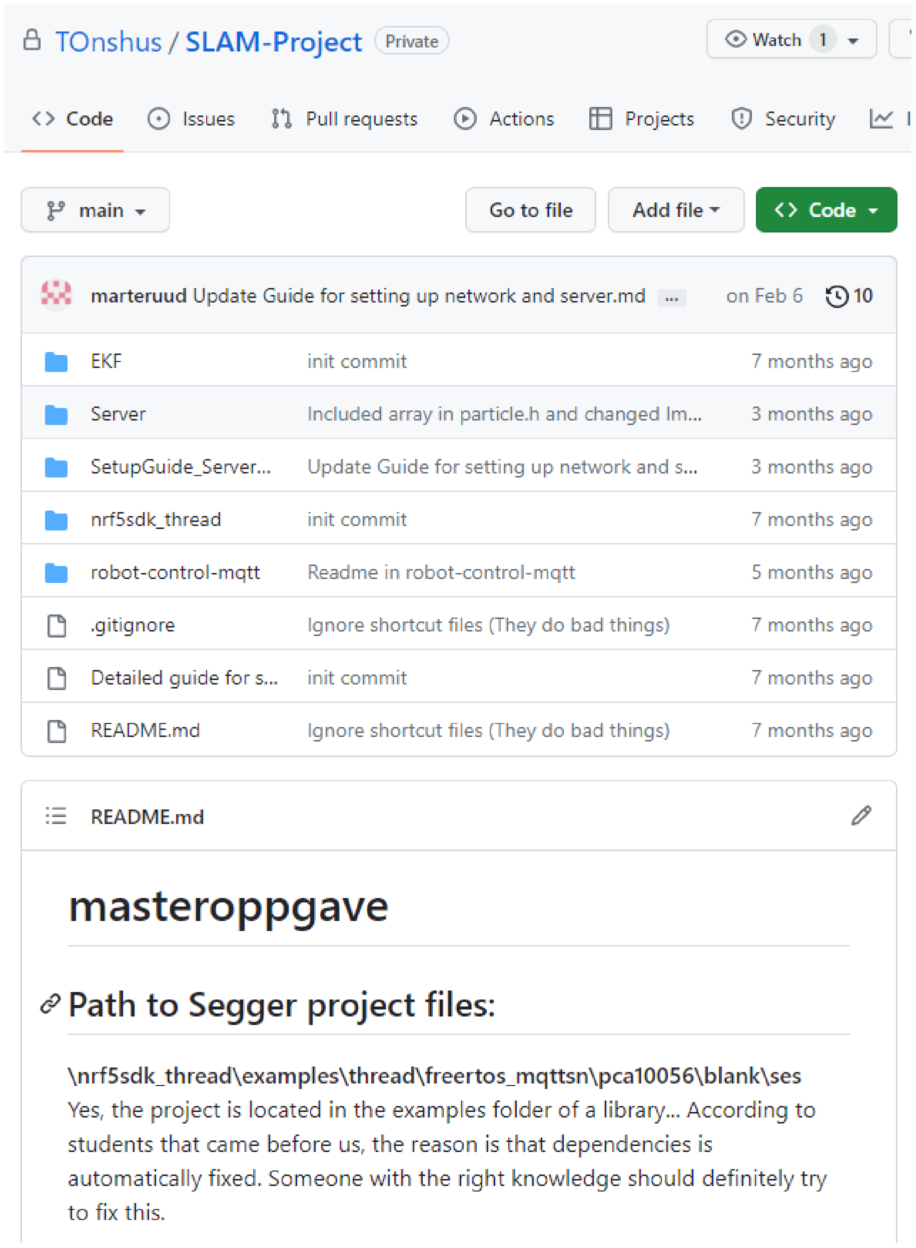
Figure 9 shows a picture of the drive containing the zip-files.



Name ↑	Modified	Modified By	File size	Sharing
delivery-Gilje_masterV22	September 22, 2022	Tor Onshus	4 items	Shared
Amsen, Master, V17.zip	June 15, 2022	Tor Onshus	16.2 MB	Shared
Andersen og Rødseth, Master, V16.zip	June 15, 2022	Tor Onshus	375 MB	Shared
Andersen, Master, V22.zip	June 13, 2022	Tor Onshus	588 MB	Shared
Berglund, Prosjekt H20.zip	December 17, 2020	Tor Onshus	279 MB	Shared
Berglund,Master, V21.zip	July 5, 2021	Tor Onshus	172 MB	Shared
Bjerke, Master, V22.zip	June 12, 2022	Tor Onshus	1.14 GB	Shared
Bjørnsen, Master, V17.zip	June 15, 2022	Tor Onshus	121 MB	Shared
Blom, Master, V20.zip	February 8, 2020	Tor Onshus	307 MB	Shared
Dypbukt, Master, V20.zip	February 8, 2020	Tor Onshus	166 MB	Shared
Eckholdt, Master, V19.zip	June 11, 2019	Tor Onshus	785 MB	Shared
Eikeland, Master, V18.zip	June 15, 2022	Tor Onshus	27.4 MB	Shared
Ese, Master, V16.zip	June 15, 2022	Tor Onshus	1.01 GB	Shared
Esper, Prosjekt, H18.zip	December 19, 2018	Tor Onshus	54.5 MB	Shared

Figure 9: Zip file handout structure.

A suggestion about moving the code over to GitHub was proposed was done in 2022. This was made by Frestad 2022 and he made a repository for the SLAM project and noted down what needed to be done with the repository. Figure 10 shows the structure of the first github repository created.



marteruud Update Guide for setting up network and server.md ... on Feb 6 10

EKF	init commit	7 months ago
Server	Included array in particle.h and changed Im...	3 months ago
SetupGuide_Server...	Update Guide for setting up network and s...	3 months ago
nrf5sdk_thread	init commit	7 months ago
robot-control-mqtt	Readme in robot-control-mqtt	5 months ago
.gitignore	Ignore shortcut files (They do bad things)	7 months ago
Detailed guide for s...	init commit	7 months ago
README.md	Ignore shortcut files (They do bad things)	7 months ago

README.md

masteroppgave

Path to Segger project files:

\nrf5sdk_thread\examples\thread\freertos_mqttsn\pca10056\blank\ses
Yes, the project is located in the examples folder of a library... According to students that came before us, the reason is that dependencies is automatically fixed. Someone with the right knowledge should definitely try to fix this.

Figure 10: First GitHub repository structure.

This was a good first step to get a better structure for a new students, and offered the possibility of version control. However, the whole SLAM project has more modules that was not added and putting the whole project into one repository would not suffice. Moreover, it was not a good solution for multiple developers to work on the same project.

Consequently, this year (2023) it was decided that a better solution would be to make it into a GitHub organization after a consultation with Anders Rønning Pettersen. This structure made much more sense because it was easier to see the different modules.

The different repositories are:

- robot-code
- matlab-robot-code
- java-server
- cpp-server
- python-robot-control
- EKF-SLAM
- ir-sensor-tower
- documentation-wiki

The robot-code repository is the robot source code to be flashed to the robot nrf52840 chip. This code a code compatible with the cpp server and it is implemented only in C.

The matlab-robot-code repository is the source code to be flashed to the robot nrf52840 chip. This robot is compatible with the java server and is implemented using both C and MATLAB generated C code.

The java-server-repository is the java server that can be used to control the matlab-robot.

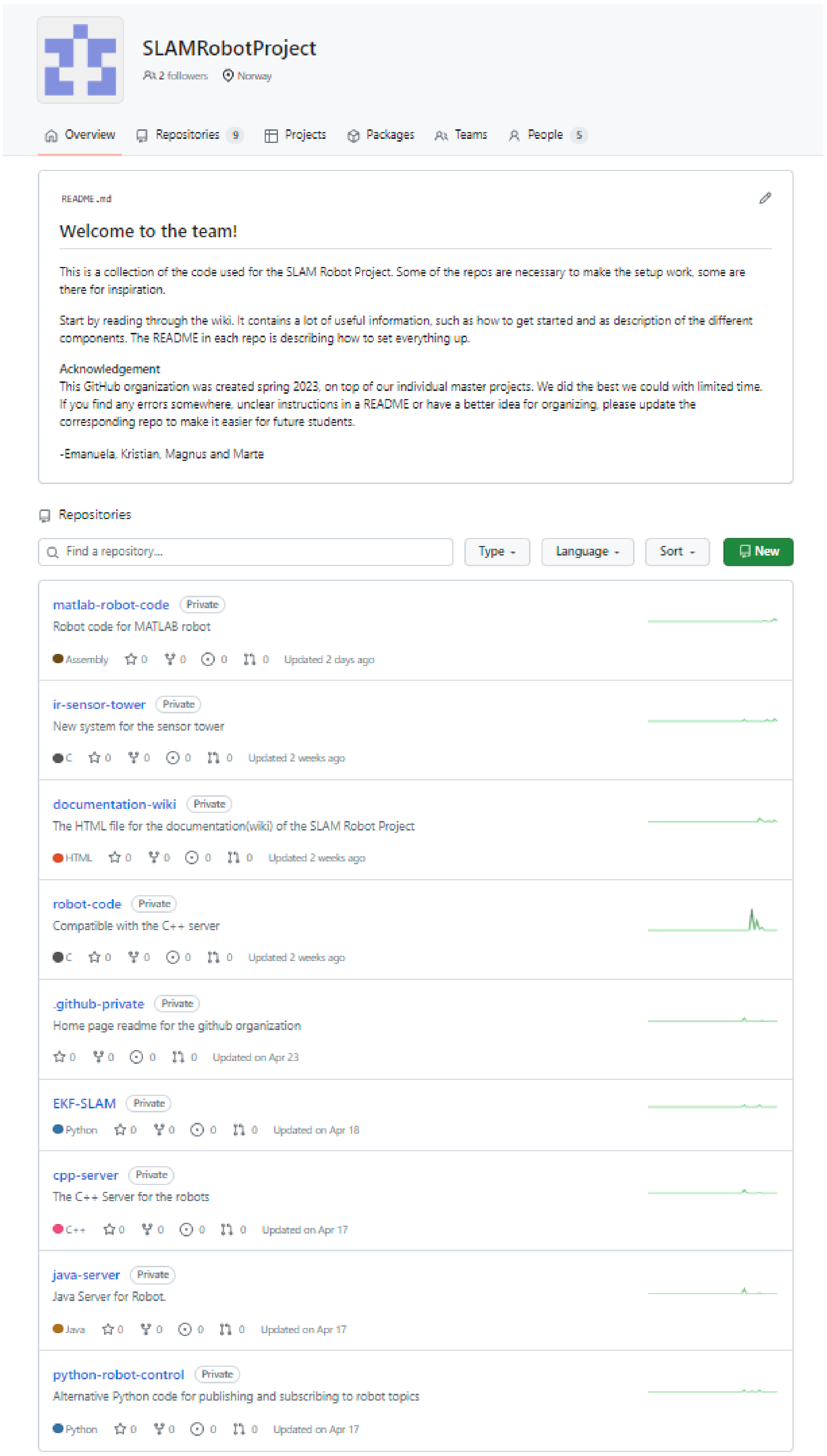
The cpp-server repository is the C++-server source code. This server can be used to control the robot using the robot-code.

The EKF-SLAM repository is the source code of a extended kalman filter SLAM algorithm implemented in python.

The ir-sensor tower repository is the source code for a new sensor tower system that was developed this year (2023).

The documentation-wiki repository contains a HTML- file for a wikipedia about the SLAM project. This was an effort of this year to provide more useful information to newer students. The reason for using this as wiki was because it required the GitHub Pro version to use a private integrated wiki in GitHub.

Figure 11 shows the new structure of the code in the GitHub Organisation.



SLAMRobotProject

2 followers · Norway

Overview · Repositories 9 · Projects · Packages · Teams · People 5

README.md

Welcome to the team!

This is a collection of the code used for the SLAM Robot Project. Some of the repos are necessary to make the setup work, some are there for inspiration.

Start by reading through the wiki. It contains a lot of useful information, such as how to get started and as description of the different components. The README in each repo is describing how to set everything up.

Acknowledgement
This GitHub organization was created spring 2023, on top of our individual master projects. We did the best we could with limited time. If you find any errors somewhere, unclear instructions in a README or have a better idea for organizing, please update the corresponding repo to make it easier for future students.

-Emanuela, Kristian, Magnus and Marte

Repositories

Find a repository... · Type · Language · Sort · New

matlab-robot-code (Private)
Robot code for MATLAB robot
Assembly · 0 · 0 · 0 · 0 · Updated 2 days ago

ir-sensor-tower (Private)
New system for the sensor tower
C · 0 · 0 · 0 · 0 · Updated 2 weeks ago

documentation-wiki (Private)
The HTML file for the documentation(wiki) of the SLAM Robot Project
HTML · 0 · 0 · 0 · 0 · Updated 2 weeks ago

robot-code (Private)
Compatible with the C++ server
C · 0 · 0 · 0 · 0 · Updated 2 weeks ago

.github-private (Private)
Home page readme for the github organization
0 · 0 · 0 · 0 · Updated on Apr 23

EKF-SLAM (Private)
Python · 0 · 0 · 0 · 0 · Updated on Apr 18

cpp-server (Private)
The C++ Server for the robots
C++ · 0 · 0 · 0 · 0 · Updated on Apr 17

java-server (Private)
Java Server for Robot.
Java · 0 · 0 · 0 · 0 · Updated on Apr 17

python-robot-control (Private)
Alternative Python code for publishing and subscribing to robot topics
Python · 0 · 0 · 0 · 0 · Updated on Apr 17

Figure 11: Github Organization structure

README-files

One of the most important files in a project is the README-file. Without a good README-file it can be very difficult to understand what the repository does. The structure of the README files that was made for the java-server, python-robot-control and matlab-robot-code repositories have this structure:

- Description of the repository
- History and acknowledgement
- Key words/phrases
- Bugs and issues
- User instructions
- Contributor instructions

The description of the repository tells the user what the purpose of the repository does. It should convey information about its utilities and motivations behind it.

History/acknowledgement should tell something about the contributors to the repository. It should convey information on how it was created and by whom.

Key words/phrases should contain important information for the user. This could be where to find information, how to use repository, what different parts of the repository does and so on.

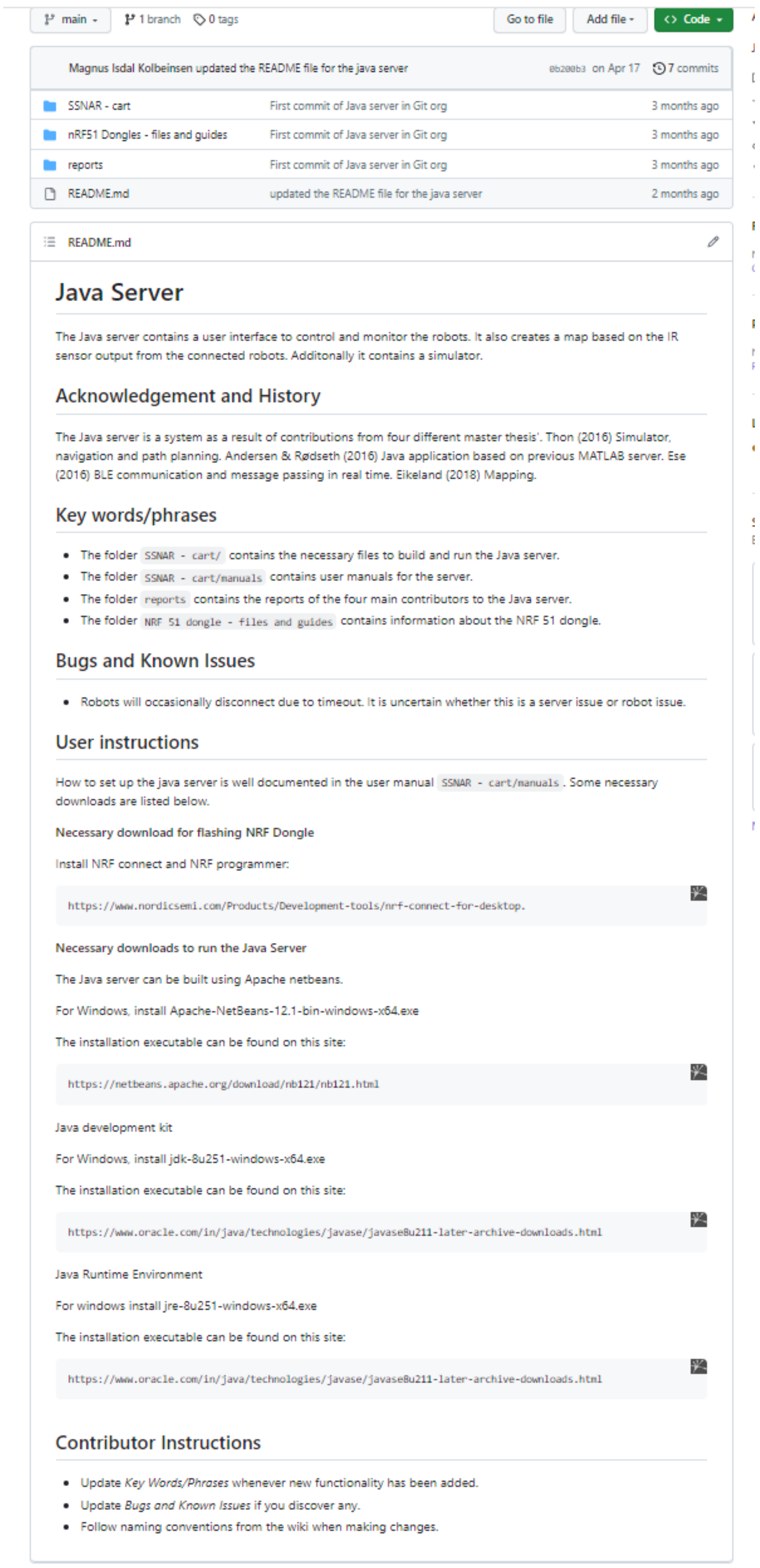
Bugs and Known Issues should contain relevant information about bugs within the repository.

User instructions should contain information about how to setup the repository.

Contributor instructions should contain information about what to do if you want to edit the repository.

An example of one of the README files that was created during this project is shown in Figure 12. Similar README-files were made for EKF-SLAM, python-robot-control and matlab-robot-code and can be found in Appendix A.

Another section that could be implemented into the README files is a visual element that shows how the repository works. This part was not added to the README-files, but can be useful for new students to see the workflow of the application.


main · 1 branch · 0 tags · Go to file · Add file · Code

Magnus Isdal Kolbeinsen updated the README file for the java server · 0b200b3 on Apr 17 · 7 commits

SSNAR - cart	First commit of Java server in Git org	3 months ago
nRF51 Dongles - files and guides	First commit of Java server in Git org	3 months ago
reports	First commit of Java server in Git org	3 months ago
README.md	updated the README file for the java server	2 months ago

README.md

Java Server

The Java server contains a user interface to control and monitor the robots. It also creates a map based on the IR sensor output from the connected robots. Additonally it contains a simulator.

Acknowledgement and History

The Java server is a system as a result of contributions from four different master thesis'. Thon (2016) Simulator, navigation and path planning. Andersen & Rødseth (2016) Java application based on previous MATLAB server. Ese (2016) BLE communication and message passing in real time. Eikeland (2018) Mapping.

Key words/phrases

- The folder `SSNAR - cart/` contains the necessary files to build and run the Java server.
- The folder `SSNAR - cart/manuals` contains user manuals for the server.
- The folder `reports` contains the reports of the four main contributors to the Java server.
- The folder `NRF 51 dongle - files and guides` contains information about the NRF 51 dongle.

Bugs and Known Issues

- Robots will occasionally disconnect due to timeout. It is uncertain whether this is a server issue or robot issue.

User instructions

How to set up the java server is well documented in the user manual `SSNAR - cart/manuals`. Some necessary downloads are listed below.

Necessary download for flashing NRF Dongle

Install NRF connect and NRF programmer:

```
https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-desktop.
```

Necessary downloads to run the Java Server

The Java server can be built using Apache netbeans.

For Windows, install Apache-NetBeans-12.1-bin-windows-x64.exe

The installation executable can be found on this site:

```
https://netbeans.apache.org/download/nb121/nb121.html
```

Java development kit

For Windows, install jdk-8u251-windows-x64.exe

The installation executable can be found on this site:

```
https://www.oracle.com/in/java/technologies/javase/javase8u211-later-archive-downloads.html
```

Java Runtime Environment

For windows install jre-8u251-windows-x64.exe

The installation executable can be found on this site:

```
https://www.oracle.com/in/java/technologies/javase/javase8u211-later-archive-downloads.html
```

Contributor Instructions

- Update *Key Words/Phrases* whenever new functionality has been added.
- Update *Bugs and Known Issues* if you discover any.
- Follow naming conventions from the wiki when making changes.

Figure 12: README file Java Server.

9 Method and Implementation

9.1 Test Setup and Environment

B333 room NTNU

The robot was mainly tested on the Opti Track Motion Capture LAB at NTNU, room B333. B333 was used instead of QualiSys at A052 because the system would not track properly because it was too high in the ceiling.

The environmental conditions for testing controlled, making them similar with every test. However, sunlight had to be blocked before testing as they interfered with the accuracy of tracking the robot. The room looks like this:

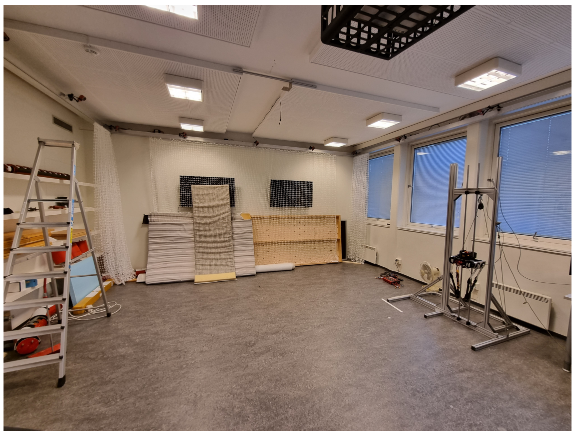


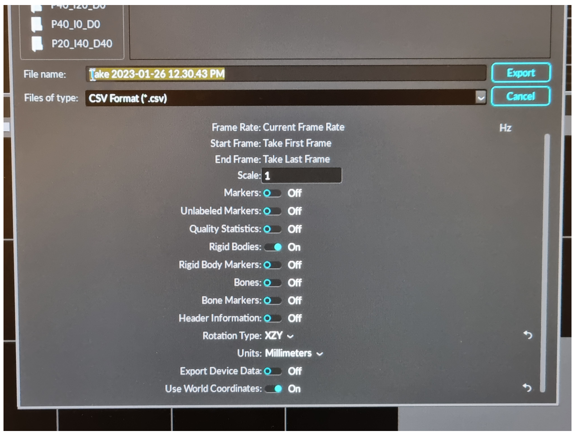
Figure 13: Testing environment in project.Room B333.

Opti Track Motive

Opti Track Motive was used to capture the robots position and orientation while running the program. This way it was possible to capture a close to ground truth value of the robot. The capture setup is the the axis are defined in a way that may not be intuitive at first. The Y axis is defined from the ceiling to the floor and using the right-hand rule to align the X axis and the Y axis.The reason behind this convention is unknown, but it made plotting a bit more cumbersome because the estimates retrieved from the encoders were had a different convention.

To capture the movement of the robot in Opti Track Motive, you need to create a rigid body inside the program. This is what the motion detection balls are for. They are placed on the robot in a pattern that makes it easy for the cameras to detect its pose. After the motion detection balls are attached to the object you want to capture, you select the given markers (movement detection balls) in the software, right-click and create rigid body.

When you want to capture, you click on the button labeled live and then click the red button. Afterwards, you click on edit. Here you are able to see the video of the robot moving. If you want to save the tracking data, you click file and export tracking data. The format that was used was .csv and the settings are shown below.



File name: Take 2023-01-26 12.30.43 PM
Files of type: CSV Format (*.csv)

Frame Rate: Current Frame Rate Hz
Start Frame: Take First Frame
End Frame: Take Last Frame
Scale: 1
Markers: Off
Unlabeled Markers: Off
Quality Statistics: Off
Rigid Bodies: On
Rigid Body Markers: Off
Bones: Off
Bone Markers: Off
Header Information: Off
Rotation Type: XZY
Units: Millimeters
Export Device Data: Off
Use World Coordinates: On

Figure 14: Settings used for tracking in Opti Track Motive.

USB COM port

USB COM port debugging was done by making a code snippet that contained useful debugging variables. This was made into a MACRO in the vControllerApiTesterTask() so that it would not interfere with the compiler. This MACRO was called DEBUG and would only compile if you the debug MACRO was true.

To use the debug macro, you need the NRF-terminal extension in visual studio code. The NRF terminal is a terminal that directly connects to the USB of the nrf52840 COM port on the short side.

This tool was very useful for retrieving the estimated positions of the robot. By simply making an array of estimated positions during one of the tests, it was possible to track the estimated positions without being connected to the java server.

Visual studio code was used during this project to develop code that was not part of the MATLAB generated code.

OLED Display

A previously implemented code called a display task was used. This task was used during this project to test the estimator(). Examples of variables that were debugged with this feature was estimated position, encoder ticks, distanceDriven and distanceRemaining.

This task was useful, but it must be used with caution as it affects the system behaviour slightly. You cannot simply use this feature without using some of the computational resources of the robot, and the display function to often affects the performance of the robot. This type of debugging was particularly useful when debugging without USB.

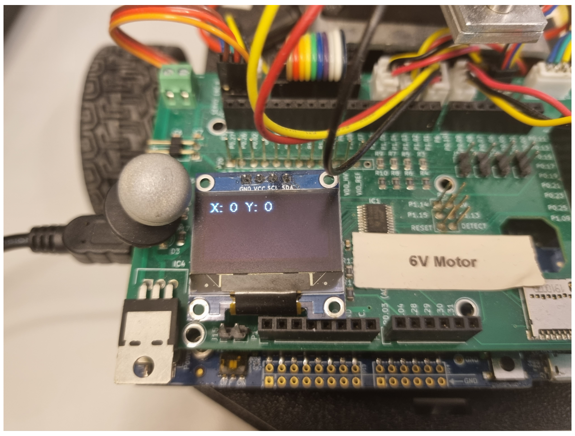


Figure 15: OLED display used for debugging while driving without USB.

Setting up the MATLAB robot

A guide on how to setup the robot can be found in the specialization project Kolbeinsen 2022.

MATLAB generated C code

A part of this project is to generate C code with MATLAB. The procedure for doing this is shown in the specialization project Kolbeinsen 2022.

One outer MATLAB generated function was made for the MATLAB generated code. This function is now called controller_api(). controller_api() interfaces the MATLAB generated code with the C code. It has a large amount of input arguments and can therefore be difficult to debug once it is compiled to C. Even though it is not ideal to have a function that has to handle that many variables, it was done like this to have all the MATLAB generated functions in one place. It reduced the amount of times you would have to generate the MATLAB functions.

Test Types

The tests that were performed during this project were:

- Manual tests.
- Line tests.
- Square tests.
- Server integration tests.

The manual tests are the tests were manual maneuvering was done.

The line tests are tests to test the robots ability to follow a straight line.

The square test is a integration test to test if the robot is able to receive and complete multiple commands. It also tells a story about its ability to keep on track after multiple commands.

The server integration test is a test where the robot takes commands from the server. This is a test to see if the system is able to not only receive commands, but also keep connection with the server while doing them.

9.2 Software Quality

Naming conventions that was made during this project:

- Global variables
- Variables
- Functions
- MACROS
- Constants

Global variables are written with a g_ in front. Example: g_xPos.

Variables in general are written in camelCase starting with a lowercase letter. Example: distanceDriven.

Functions are written with snake case and all lower case. Example: move_forward().

MACROS are written with ALL CAPS snake case. Example: NO_TEST.

Constants are written with ALL CAPS snake case. Example: COMPLETE.

Other naming conventions that was used was to try to focus on the what rather than the why. Example: turning instead of robotStateFlag.

Other means that was used to make the code more readable was to make sure that indents where placed appropriately.

For documentation, variable explanations was placed inside the code. Additionally, complex code sections were briefly justified in the code. The goal was to write the code in a way such that the code itself was self-documenting and easy to comprehend.

9.3 Movement Control System Design

The structure of the movement control system is similar to the system from the specialization project. Figure 16 shows how the system design is today. Information flow stays the same, but api() has been renamed to controller_api() and vApiTask() to vControllerApiTask(). The implementation of vControllerApiTask can be found in Appendix B.

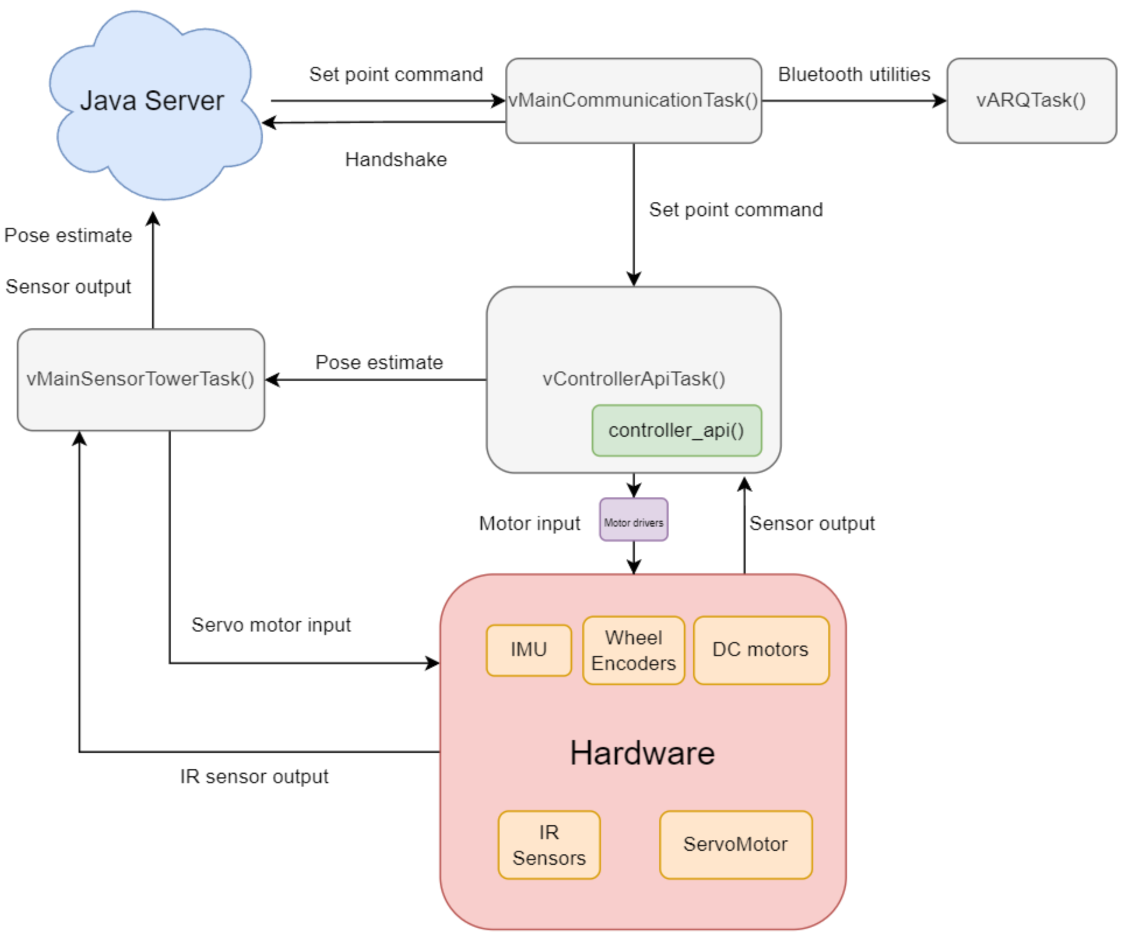


Figure 16: Overview of the system today.

The general structure of the controller_api() can be seen in Figure 17. The structure is similar to the specialization project. api() has been renamed to controller_api(), moveForward to move_forward() and rotateRobot to rotate_robot(). A PID-controller has been implemented inside move_forward() which required extra variables such as thetaIntegralError, thetaDerivativeError and sTime.

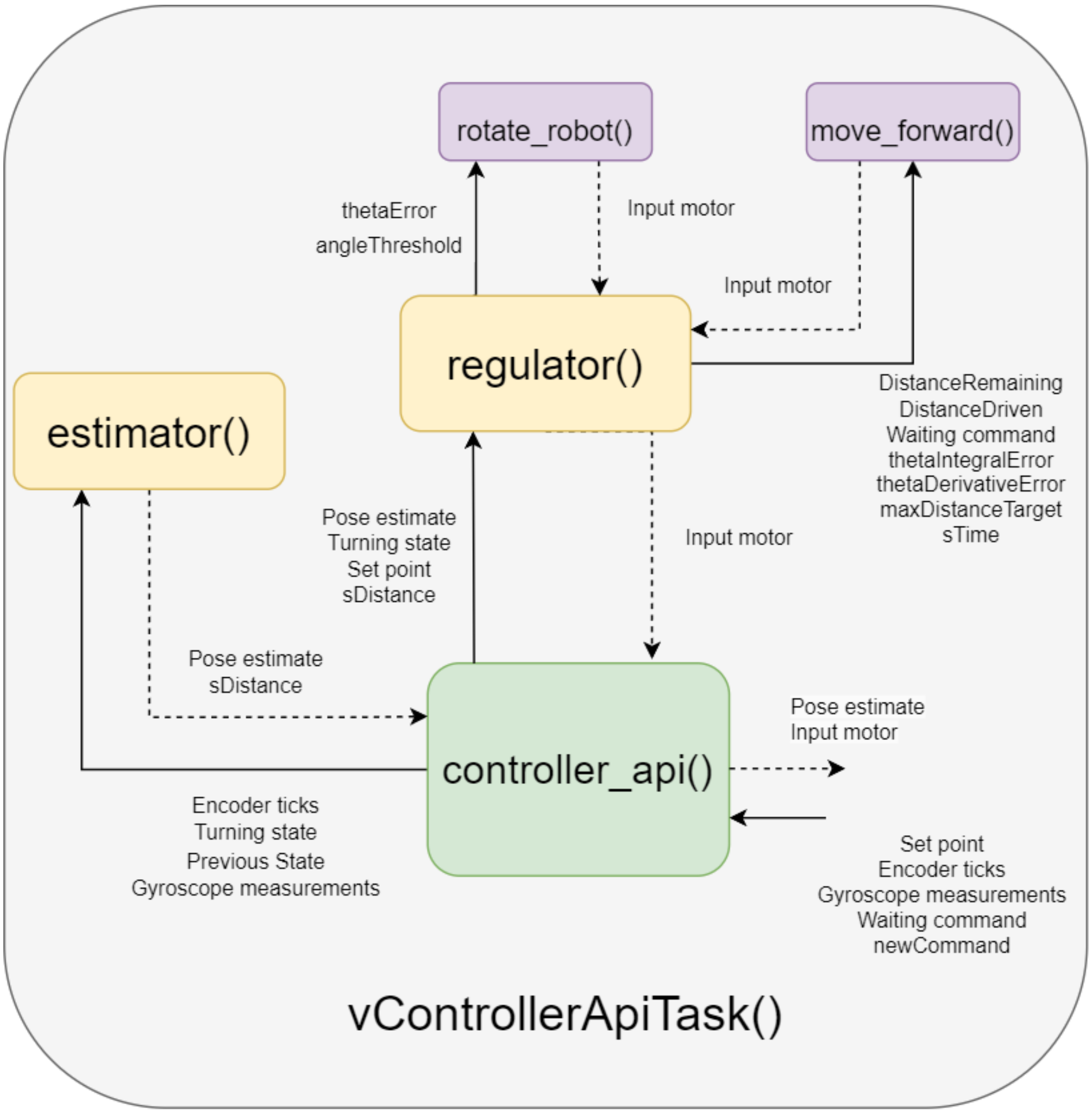


Figure 17: Overview of controller_api() funciton

You can see how the different MATLAB generated functions work together. controller_api() calls the estimator which returns the pose estimate. It then calls the regulator function which takes a decision whether to rotate or move forward and returns the input to the left and right DC motor.

9.4 Movement Control System Implementation

9.4.1 controller_api()

The controller_api() function is the function that interfaces the MATLAB code with the C code. This function is used to transfer necessary signals from vControllerApiTask() to the MATLAB generated code. It takes signals such as setpoint commands, encoder ticks and gyroscope measurements and returns input to the left and right motor, uL and uR. An effort to improve software quality has also been made by renaming the variables and adding comments inside the code.

The design of the funciton is similar to the one created in the specialization project, but new signals have been added:

- Gyroscope signal from IMU, sThetaGyro.
- Start positions when receiving new commands, ddInitX, ddInitY.
- PID error for integral action, thetaIntegralError.
- Sample time, sTime.

The functions itself contains the estimator and the regulator. The code is shown below.

```matlab
function [gX_hat,gY_hat,gTheta_hat,distanceDriven,leftU,rightU,
          turning,waitingCommand,thetaIntegralError,thetaError] =
controller_api(setpointX,setpointY,newCommand,waitingCommand,ticksLeft,ticksRight,
               distanceDriven,turning,xprev,yprev,thetaprev,ddInitX,ddInitY,sThetaGyro,
               thetaIntegralError,delta_t,thetaError)
% This function changes global states and returns input to the motors.
% FUNCTION     ~ Description
% ----------------------------------------------------------------------
% estimator() ~ makes an estimate of the position and orientation
% regulator() ~ returns input to the DC motor drivers
% ----------------------------------------------------------------------

setpoint = [setpointX,setpointY];
Encoder = [ticksLeft,ticksRight];
ddInit = [ddInitX, ddInitY];
prev = [xprev,yprev,thetaprev];

[xHat, sDistance] = estimator(Encoder,prev,turning,sThetaGyro);

[uL, uR,distanceDriven,turning,waitingCommand,thetaIntegralError,thetaError] =
↪   regulator(xHat,setpoint,turning,sDistance,distanceDriven,ddInit,waitingCommand,
          thetaIntegralError,delta_t,thetaError,newCommand);

gX_hat = xHat(1);
gY_hat = xHat(2);
gTheta_hat =xHat(3);

leftU  = uL;
rightU = uR;
end
```

9.4.2 estimator()

The estimator takes in encoder ticks and gyroscope measurements, and returns a pose to controllerApi(). The estimator calculates the pose differently depending on the movement state of the robot; turning and not turning.

Most of the estimator was implemented in the specialization project, but it is now using gyroscope measurements for orientation when the robot is tuning.

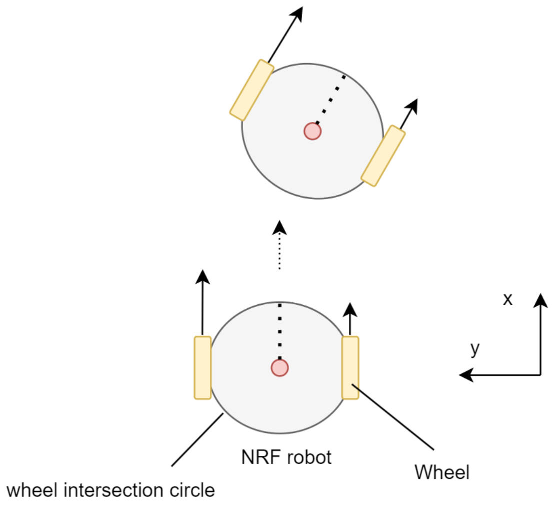


Figure 18: Robot moving forward.

The encoder ticks of the wheels are used in the move forward part of the estimator to calculate position and to calculate robot orientation. Figure 18 shows a picture of what happens when the robot moves forward with different input on each wheel. Since each wheel have different inputs the robot will do a turn, in this case to the right. Consequently, there will be a difference between the left and right encoder ticks. The idea is that the difference in amount of ticks equals a distance around the wheel intersection circle and this distance is equal to a change in orientation of the robot multiplied by a ratio involving the circumference of the wheel intersection circle. Equation (2) shows how this calculation happens.

$$sTheta = (distanceTicks/oneRevWheelBase)2\pi \tag{2}$$

where sTheta is the change in orientation of the robot each sample, distanceTicks is the distance travelled around the intersection circle and OneRevWheelBase is the circumference of the wheel intersection circle.

Afterwards, it calculates the distance driven in one sample, sDistance, which is calculated based on the average amount of ticks of both the left and right motor. Equation (3) shows how sDistance is calculated.

$$sDistance = aTicks * wheelRatio; \tag{3}$$

where sDistance is the total distance traveled that sample, aTicks is the average ticks of the left and right motor and wheelRatio is the ratio to convert ticks into a distance.

Furthermore, it then calculates the positions by using adding an increment to the previous state. Equations (4), (5) and (6) shows the calculation of the robot pose.

$$x = x_{prev} + sDistance\cos(\theta_{robot}) \tag{4}$$
$$y = y_{prev} + sDistance\sin(\theta_{robot}) \tag{5}$$
$$\theta_{robot} = \theta_{prev} + sTheta \tag{6}$$

where $[x, y, \theta_{robot}]$ is the estimated robot pose, sTheta the change in θ_{robot} that sample and sDistance is the distance driven that sample.

An assumption about stability is made when calculating sTheta, and that is that between samples the theta velocity stays constant. So when you are calculating a change in theta by adding an increment of the velocity times the change of time, between time samples you have to assume that the velocity are somewhat stable during that time period.

If the robot is in rotation mode, the estimator now uses the gyroscope measurements to calculate orientation. Figure 19 shows how the robot rotates by moving the left and right wheel in different directions.

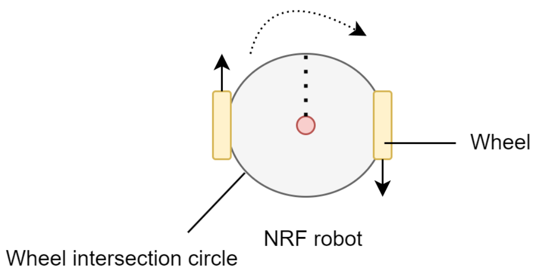


Figure 19: Robot rotating, turning.

The assumption made in this part of the estimator is that the robot is not changing its position while turning at a standstill. In other words, $x = x_{prev}$ and $y = y_{prev}$. Equation (7),(8) and (9) shows how the estimator calculates robot pose while in turning mode.

$$x = x_{prev} \tag{7}$$
$$y = y_{prev} \tag{8}$$
$$\theta_{robot} = \theta_{prev} + sThetaGyro \tag{9}$$

where $[x, y, \theta_{robot}]$ is the estimated robot pose and sThetaGyro is the gyroscope measurement from the current sample, in radians.

The implementation of gyroscope measurements into the MATLAB code was simple because of the way the system was designed in the specialization project. The sTheta variable was changed to sThetaGyro with a conversion from degrees to radians. It required more in vControllerApiTask() however, as a conversion from angular velocity to theta increment in a sample was needed to do, and additionally a calibration of the measurements to avoid drift.

The implemented code can be seen below.

```matlab
function [xHat, sDistance] = estimator(Encoder,prev,turning,sThetaGyro)
% Estimates position and orientation [x_hat, y_hat, theta_hat]
% Returns xHat
% VARIABLE   [UNITS] [RANGE]      ~  Description
% -------------------------------------------------------------------------
% Encoder   [int,int]             ~ wheel encoder ticks
% sThetaGyro[rad]                 ~ theta increment from gyroscope
% turning   [bool]                ~ is the robot in rotating mode?
% sDistance [mm]                  ~ distance travelled since last cycle
% prev      [mm,mm,rad]           ~ the previous iteration from xHat
% xHat      [mm,mm,rad]           ~ the current estimate pose [x,y,theta]
% -------------------------------------------------------------------------

%Constants  [ticks], [mm]
nTicks                    = 300;
diameterWheel             = 67;
oneRevWheel               = pi*diameterWheel;
wheelRatio                = oneRevWheel/nTicks;

diameterWheelBase         = 168;
oneRevWheelBase           = pi*diameterWheelBase;

%Encoder ticks from sample, difference of tick sample and
%average of tick
sTicksLeft               = Encoder(1);
sTicksRight              = Encoder(2);
dTicks                   = sign(sTicksRight-sTicksLeft)*abs(sTicksRight-sTicksLeft)/2;
aTicks                   = (sTicksLeft+sTicksRight)/2;

%Distance [mm] from one sample, Distance of ticks difference (used to calculate angle
↪   theta)
sDistance                = aTicks*wheelRatio;
distanceTicks            = dTicks*wheelRatio;

%Theta change orientation [rad] in one sample
sTheta                   = (distanceTicks/oneRevWheelBase)*2*pi;

if turning
    x = prev(1);
    y = prev(2);
    theta = prev(3)+deg2rad(sThetaGyro);
    sDistance = 0;
else
    x = prev(1) + sDistance*cos(prev(3));
    y = prev(2) + sDistance*sin(prev(3));
    theta = prev(3)+sTheta;

end
theta = modulus(theta+pi,2*pi)-pi; %% smallest signed angle (maps angle into [-pi,pi]
xHat = [x,y,theta];
end
```

9.4.3 regulator()

The main function of the regulator is to return the input uL and uR. It calls two different functions based on the movement state of the robot, turning and not turning. These functions are called rotate_robot() and move_forward().

The regulator also performs important calculations for rotate_robot() and move_forward().

- maxDistanceTarget
- distanceDriven
- destinationZone
- distanceRemaining
- thetaError
- thetaIntegralError
- thetaDerivativeError
- thetaSetpoint

The maxDistanceTarget variable is the distance between the target and the initial position when receiving the set point command.

The distanceDriven variable is distance the robot has driven since receiving a command. It is used in combination with maxDistanceTarget to idle the robot.

The destinationZone parameter is a constant that is set to 100mm.

The distanceRemaining variable is the distance between the robot and the target in a straight line. It is used in combination with the destinationZone parameter to tell the robot it has now reached its target and can stop.

The thetaError variable is the difference between the orientation of the robot and the orientation towards the end destination. This variable is used in both move_forward() and roate_robot().

The thetaIntegralError variable is the integrated version of thetaError. It is mainly used in the PID-controller in move_forward().

The thetaDerivativeError variable is the rate of change of the thetaError. It is used in the PID-controller in move_forward().

The thetaSetpoint variable is orientation towards the end destination in a straight line. Figure 20 shows the robot pose relative to the end destination.

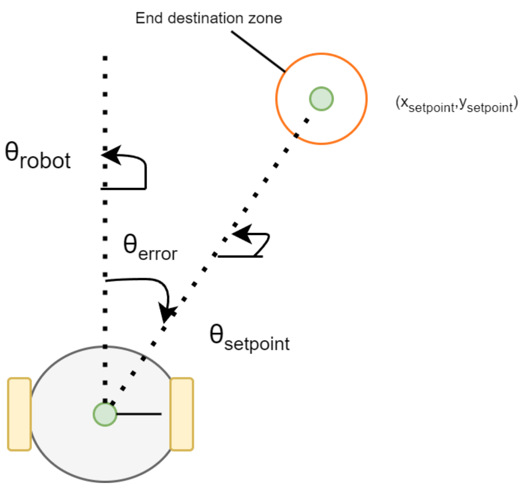


Figure 20: Robot pose and orientation towards target.

The main differences between this regulator and the specialization project in this are:

- Proportional Error.
- Integral Error.
- Derivative Error.
- The input slowdown algorithm.
- The maxDistanceToTarget variable.

The proportional error is calculated by using smallest signed angle on the difference between the orientation of the robot and the correct angle towards the target. The smallest signed angle was an important implementation in the specialization project as it mapped angles into $[-\pi, \pi]$ and therefore preventing a 270° rotation.

The integral error is calculated by integrating the theta error samples and then saturating the max theta integral error to avoid too large of oscillations due to integrator windup. Integrator windup is also handled in the C code where theta integral error is set to 0 every time the robot receives a new command.

The derivative error is calculated by comparing the current theta error with the theta error from the previous sample and then dividing by the loop time of the system.

The maxDistanceTarget variable is calculated based on where it started when it received its command. This was implemented by remembering the initial value of x and y when it received a new command and then comparing it to the desired target positions, calculating max distance to target. This variable is implemented as a fail safe to prevent the robot from crashing if it does not hit its target. It can only travel this distance before it shuts down and cancels the command.

The input slowdown algorithm is implemented by making the base input to the motors while moving forward dependent of the distance to the target. In other words, if the robot is far way from the robot, it will exert higher duty cycle on the each wheel. Equation (10) shows how the base input to each motor was manipulated. 10 was set as the minimum each motor would move because otherwise the robot would not exert enough force to move and any values lower than this would damage the system behaviour. Additionally, the maximum was set to 30, giving the robot the ability to regulate with 20 since the max duty cycle was 50%.

$$u = min(30, (10 + distanceRemaining/200)) \tag{10}$$

The implemented regulator() function can be seen below.

```
function [uL,uR,distanceDriven,turning,waitingCommand,thetaIntegralError,thetaError] = regulator(xHat,setpoint,turning,sDistance,distanceDriven,ddInit,waitingCommand,thetaIntegralError,sTime,
%Returns the inputs to the Left and Right motors
% VARIABLE          [UNITS] [RANGE]   Description
% ----------------------------------------------------------------
% xHat              [mm,mm,rad]       = [xEstimateNow, yEstimateNow, thetaEstimateNow]
% setpoint          [mm,mm]           = [xSetpoint, ySetpoint]
% ddInit            [mm,mm]           = [ddInitX, ddInitY] ~ the position estimate at the time of receiving set point command
% turning           [bool]            ~ is the robot in rotating mode?
% waitingCommand    [bool]            ~ the robot is in an idle state waiting for a new command?
% sDistance         [mm]              ~ distance moved this sample
% sTime             [s]               ~ sample time this cycle
% xDiffTarget       [mm]              ~ difference between setpoint x and current x position estimate
% yDiffTarget       [mm]              ~ difference between setpoint y and current y position estimate
% thetaError        [rad] [-pi,pi]    ~ difference between orientation of robot and orientation towards setpoint
% distanceDriven    [mm]              ~ distance driven since last setpoint command
% distanceRemaining [mm]              ~ distance from current position towards target
% maxDistanceTarget [mm]              ~ max distance to travel before stopping
% destinationZone   [mm]              ~ radius around end destination
% angleThreshold    [rad]             ~ stop condition for rotation mode
% uL,uR             [%, %] [0,50]     ~ input duty cycle to left and right motor
% ------------------------------------------------------------------------
%xHAt
xEstimateNow           = xHat(1);
yEstimateNow           = xHat(2);
thetaEstimateNow       = xHat(3);

%Setpoint positions
xSetpoint = setpoint(1);
ySetpoint = setpoint(2);

%Difference in posistion towards the target
xDiffTarget = xSetpoint - xEstimateNow;
yDiffTarget = ySetpoint - yEstimateNow;

%Conditional Thresholds
angleThreshold       = deg2rad(3);
maxDistanceTarget    = sqrt( (xSetpoint-ddInit(1))^2 + (ySetpoint-ddInit(2))^2 );
destinationZone      = 100;
distanceRemaining    = sqrt(xDiffTarget^2+yDiffTarget^2);
distanceDriven       = distanceDriven+sDistance;

%Angle towards target
thetaSetpoint = atan2(yDiffTarget,xDiffTarget);

%Proportional Error
thetaErrorPrev = thetaError;

thetaError = modulus(thetaSetpoint-thetaEstimateNow+pi,2*pi)-pi; %%smallest signed angle
rad2deg(thetaError)

%Derivative Error
thetaDerivativeError = (thetaError -thetaErrorPrev)/sTime;
if newCommand
    thetaDerivativeError = 0; % To prevent a unintended high contribution from derivative term after rotation.
```

```
end

%Integral Error
thetaIntegralError=min(thetaIntegralError+thetaError,deg2rad(20)); % saturation of
↪  integralerror to +-20 degrees
thetaIntegralError=max(thetaIntegralError,deg2rad(-20));

if turning
    u = 12;
    [uL, uR,turning] = rotate_robot(thetaError,angleThreshold,u);
    distanceDriven = 0;
else
    u = min(30,(10+distanceRemaining/200)); %Input slows down depending on distance
↪   remaining to target
    [uL, uR] = move_forward(thetaError,distanceRemaining,distanceDriven,destinationZone,
↪   maxDistanceTarget,u,waitingCommand,thetaIntegralError,sTime,thetaDerivativeError);
    if (uL == 0) && (uR == 0) % Needed to idle the robot
        waitingCommand = 1;
    end
end
```

9.4.4 rotateRobot()

After receiving a command, the robot will first rotate towards its target. If the robot is in this turning mode, it will call rotate_robot(). This function takes in theta error, angleThreshold and the input to rotate with, u. It is a simple algorithm that uses the sign of the thetaError to decide in what direction to rotate. Moreover, it changes the status variable, turning, if thetaError becomes small enough. rotate_robot() can be seen in the listing below.

```
function [uL,uR,turning] = rotate_robot(thetaError, angleThreshold, u)
% Rotates robot based on its current position and threshold for rotating
% Returns input to DC motor drivers and manipulates turning variable
% VARIABLE    [UNITS][RANGE]          ~  Description
% -----------------------------------------------------------------------
% turning           [bool]            ~ is the robot in rotating mode?
% thetaError        [rad]             ~ difference between robot orientation and
↪  orientation towards target
% angleThreshold    [rad]             ~ stop condition for rotation mode
% uL,uR,u           [%, %,%] [0,50]   ~ input duty cycle to left and right motor
% -----------------------------------------------------------------------
if abs(thetaError)>angleThreshold
    turning = 1;
    if (thetaError)>0 %% rotate counterclockwise
        uR = u;
        uL = -u;
    else %% rotate counter clockwise
        uR = -u;
        uL = u;
    end
else
    turning = 0;
    uR = 0;
    uL = 0;
end
```

9.4.5 moveForward()

After rotating towards the target, the regulator() calls moveForward() to move the robot towards the target. This function uses PID-controll to regulate each wheel. It uses the thetaError, thetaIntegralError and thetaDerivativeError to regulate the robot towards the end destination. If the robot has a negative thetaError it will compensate by exerting more force on the right wheel. If the robot has a positive thetaError, it will exert more force on the left wheel. Figure 21 displays how the robot would behave if it was tilted towards the right, i.e., having a negative thetaError.

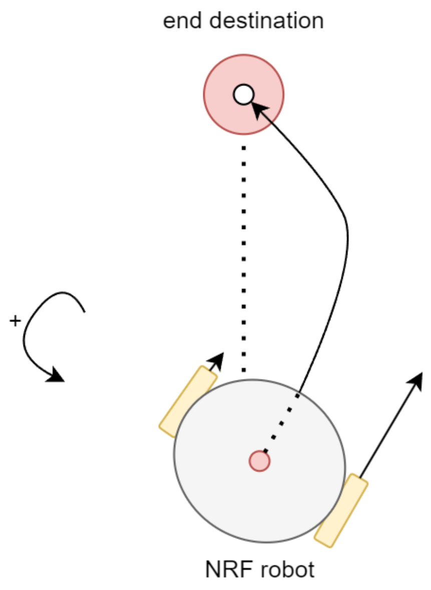


Figure 21: Moving foward

The function contains:

- PID gains.
- PID saturation limits.
- PID control terms for each wheel.

The PID gains are used to tune the pid controller.

The PID saturation limits were implemented in order to avoid control issues due to saturation.

The PID-controller were implemented in a way such that each wheel is regulated based on where the robot is heading. If it is going left and the target is to the right, it would add more input to left wheel. This way of controlling the robot added extra complexity, but was solved by using $(1 - sign(error))/2$ and $(1 + sign(error))/2$ for left and right motor, respectively.

The resulting input to each wheel can be seen in Equation (11) and (12)

$$\begin{aligned} uR &= u + dirProportionalControlRight \cdot uP + dirIntegralControlRight \cdot uI \\ &\quad - dirDerivativeControlRight \cdot uD \qquad (11) \\ uL &= u + dirProportionalControlLeft \cdot uP + dirIntegralControlLeft \cdot uI \\ &\quad - dirDerivativeControlLeft \cdot uD \qquad (12) \end{aligned}$$

where u is the base input to the motor, dirProportionalControlRight, dirIntegralControlRight and dirDerivativeControlRight are conditional control terms for the right motor, dirProportionalControlLef, dirIntegralControlLeft and dirDerivativeControlLeft for the right motors. uP, uI and uD are the input to the motor from the PID-controller. The full implementation can be seen below.

```matlab
function [uL,uR] = move_forward(thetaError,distanceRemaining,distanceDriven,destinationZone,maxDistanceTarget,u,waitingCommand,thetaIntegralError,sTime,thetaDerivativeError)
% Moves the robot towards the target. PID controller to controll each wheel input.
% Returns the input uL and uR to the DC motors drivers
% VARIABLE          [UNITS] [RANGE]     Description
% ------------------------------------------------------------------------
% thetaError        [rad] [-pi,pi]      ~ difference between orientation of robot and orientation towards setpoint
% distanceRemaining   [mm]              ~ distance towards the destination
% distanceDriven      [mm]              ~ distance driven since last command
% destinationZone     [mm]              ~ radius around end destination
% maxDistanceTarget   [mm]              ~ maximal distance to travel for a command
% waitingCommand      [bool]            ~ the robot is in an idle state waiting for a new command?
% thetaIntegralError  [rad]             ~ accumulated angle deviation
% thetaDerivativeError[rad/s]           ~ theta derivative divided by loop time
% sTime               [s]               ~ sample time
% uL,uR,u             [%, %,%] [0,50]   ~ input duty cycle to left and right motor
% ---------------------------------------------------------------------------

%PID gains
kP = 20;
kI = 10;
kD = 0.5;

%PID saturation limits
proportionalLimit   = 20;
integralLimit       = 10;
derivativeLimit     = 10;

%PID corrections with saturation
uP = min(kP * abs(thetaError),proportionalLimit);
uI = min(kI * abs(thetaIntegralError)*sTime,integralLimit);
uD = min(kD * abs(thetaDerivativeError),derivativeLimit);

% Conditional Control Terms depending on direction
dirProportionalControlLeft  = (1-sign(thetaError))/2; % 1 if tilted left, 0 if tilted right, 0.5 if at target direction
dirIntegralControlLeft      = (1-sign(thetaIntegralError))/2;
dirDerivativeControlLeft    = (1-sign(thetaDerivativeError))/2;

dirProportionalControlRight = (1+sign(thetaError))/2;
dirIntegralControlRight     = (1+sign(thetaIntegralError))/2;
dirDerivativeControlRight   = (1+sign(thetaDerivativeError))/2;


if distanceRemaining>destinationZone && distanceDriven<maxDistanceTarget && not(waitingCommand)
   uR = u+dirProportionalControlRight*uP+dirIntegralControlRight*uI-dirDerivativeControlRight*uD;
   uL = u+dirProportionalControlLeft*uP+dirIntegralControlLeft*uI-dirDerivativeControlLeft*uD;
   uR = round(uR); % to minimize type casting error to int later
   uL = round(uL);
   % [uL,uR] must not exceed MAX value [50,50].
else
    uR = 0;
    uL = 0;
end
```

9.4.6 vControllerApiTask()

vControllerApiTask() is the task were the controller_api() function is implemented. This task was renamed and modified to work with the new robot controller. The reason for this convention was that all the previous tasks in the code implementation had this convention for tasks. vControllerApiTask is a better name than vApiTask() because it conveys more about what this task does and what it is. Additionally, a tester task was made, vControllerApiTesterTask(). This task can be swapped out with vControllerApiTask() if you want to debug or do tests without the server.

vControllerApiTask() was mainly modified such that:

- The robot was able to receive commands from the server.
- The gyroscope measurements was calibrated upon startup.
- Variables from controller_api() was initialized.
- controller_api() worked together with this task.

vControllerApiTesterTask() contains:

- MACROS to interchange between different tests (square,line).
- MACRO to debug via USB.
- MACRO to log.
- Display task.

The implementation of vControllerApiTask() can be seen in Appendix B.

The implementation of vControllerApiTesterTask() can be seen in Appendix C.

MACRO

MACROs was used actively in code development. As mentioned before it was used to debug the code via USB, but it was also used to make different sequences for testing. This made it much faster to test different sequences of the code.

The MACROS that was made was TEST MACRO, DEBUG MACRO and LOG MACRO.

The debug MACRO was simply a true or false. You either wanted to debug or not.

The test MACRO had the values NO_TEST, LINE and SQUARE.

The no test MACRO would make the robot stay still and not move and would be used in combination with the debug macro often to check values on the robot.

The square and line test were used so frequently in the testing of the robot code that it was necessary to find a way to interchange between these tests, and that is what the LINE and SQUARE marco does.

The log MACRO is used in combination with LINE or SQUARE to log the estimated positions.

MACROS were very useful in this case. It sped up development time by a lot and allowed for a more trial and error approach to code development. By simply changing a variable value, you could change between a different compilations of the code. For a new person using the code, it might be a bit confusing to see the MACROS. This is not something that necessarily improves readability of the code, but it certainly makes it easier to maintain and test.

10 Results

10.1 Initial Tests

In order to improve the system of the specialization project, it was needed to test its current system behaviour again. Two tests were performed on the system, a square test of 300mm and a line test of 1000mm. These two tests were completed early on and without checking the estimates of the robot system.

The square test was conducted to test the systems' ability to follow four consecutive commands and its ability to rotate. The test shows the four way points given to the robot, the starting and end point of the robot and the path it took. Its main purpose was to test its precision of orientation, θ_{robot}. The test was conducted by running a program that gave the robot four different position commands and using Opti Track Motive to track its 3D position. Figure 22 shows the result of this test.

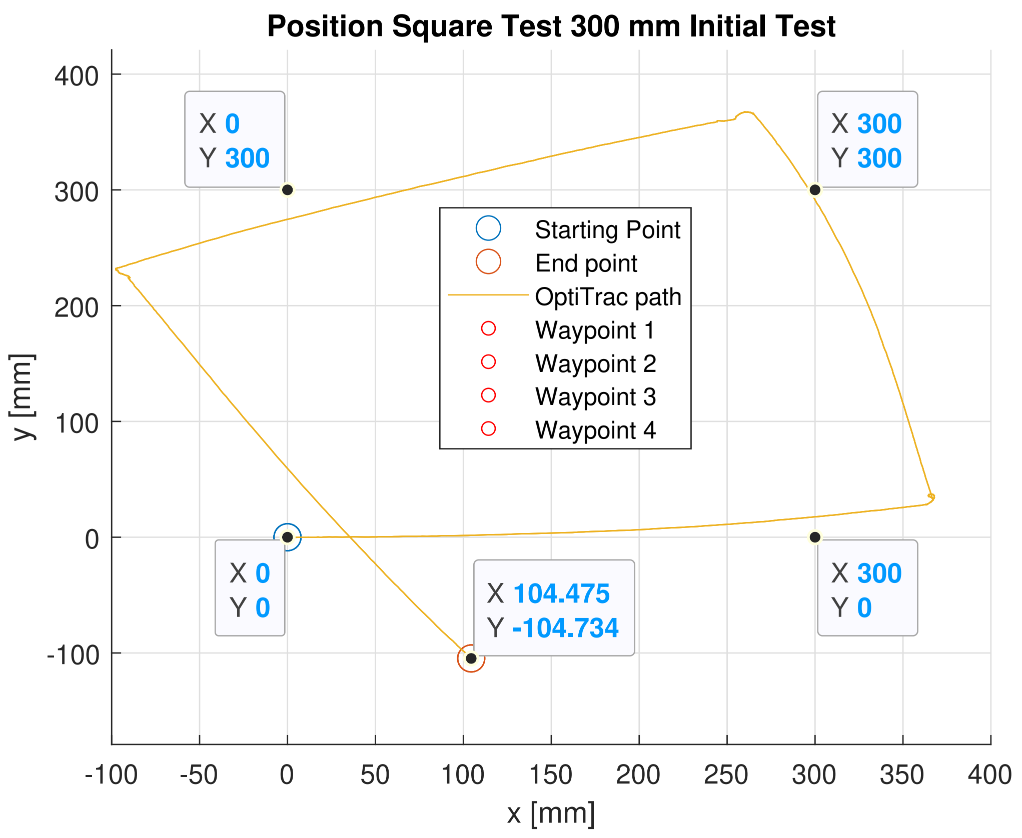


Figure 22: 300 mm Position Square Test.

It can be seen that the robot diverges slightly from the first waypoint (0,300) and overshoots the waypoint with about 70 mm in the x direction. Moreover, it rotates towards the second waypoint with good accuracy. However, it travels further away from the waypoint. This drift from the target goal increases with multiple waypoints.

The line test was conducted to test how much the robot would diverge from a straight line. The test shows the waypoint given to the robot with a destination zone that was implemented in the program. This zone at the time of the test was a circle with 50 mm radius. The test was conducted by running a program that sent the robot a position command to (1000,0) and Opti Track Motive was used to track its 3D position. Figure 23 shows the result of the first line test.

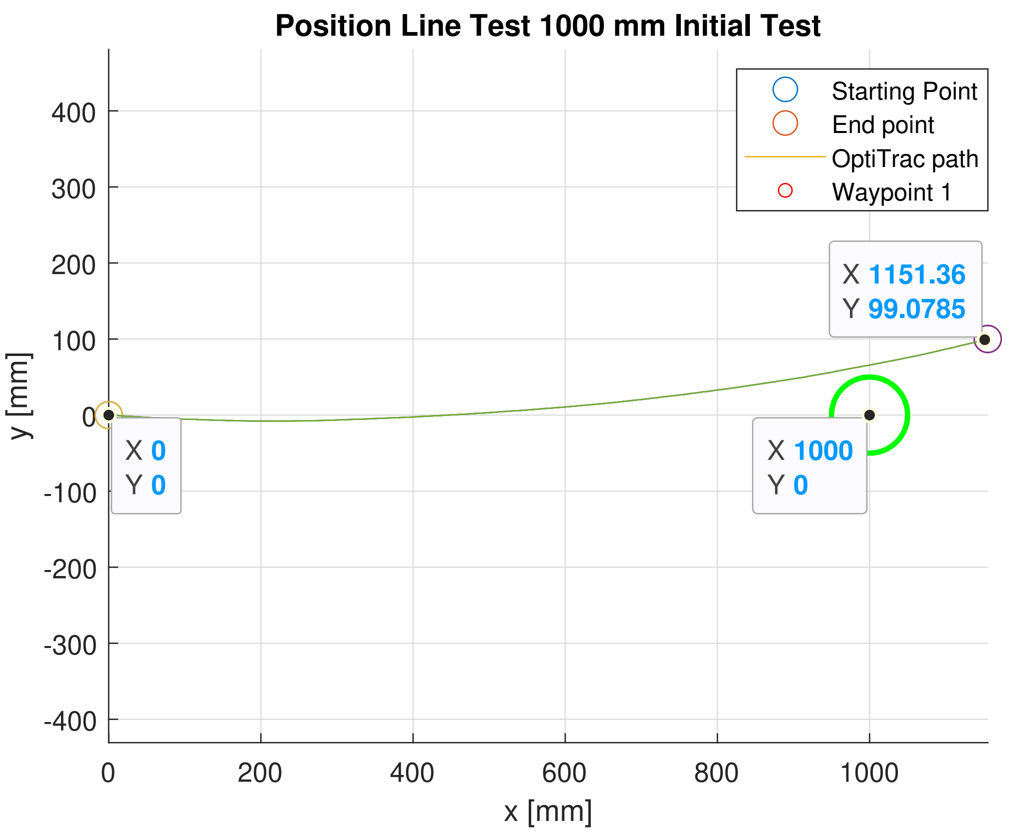


Figure 23: Initial Position Line Test.

The robot diverges from the position command and overshoots. The end position is about 180 mm away from the target position. However, it only slightly misses the end destination zone. Consequently, it is likely it hit the previously implemented max distance fail safe before stopping.

10.2 Initial Testing Gyroscope

The initial square test made it clear that using encoders was not a reliable way to measure orientation after several commands. As a way to improve the system, implementation of gyroscope measurements into the estimator was suggested. To do this a test of the gyroscope measurements were completed.

The gyroscope test was performed by keeping the robot at a standstill. The output of the gyroscope was then read by using a USB cable on the short end of the nRF52840 chip. It was conducted to confirm how accurate the values from the gyroscope was. This test was also conducted during the specialization project, and was an important step towards implementing the gyroscope into the estimator.

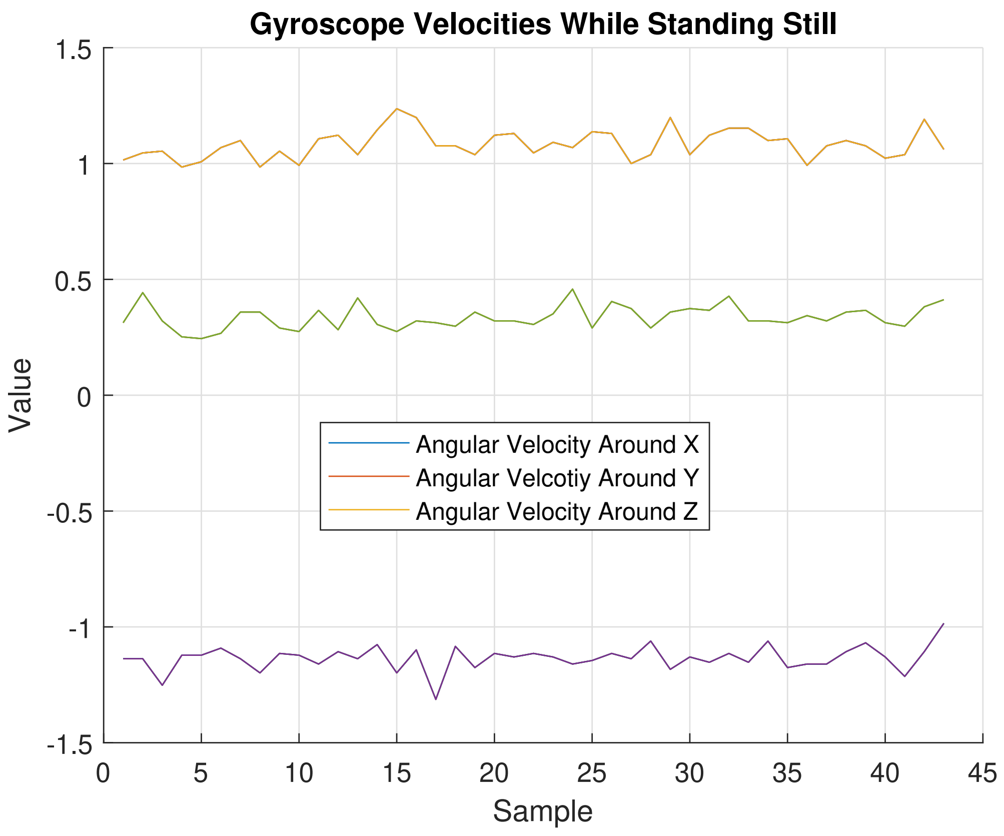


Figure 24: Plot of angular velocities from gyroscope while standing still.

In the plot Figure 24 you can see the angular velocities measurements of the gyroscope. The gyroscope is mounted on the bottom of the robot with the z axis pointing upwards and xy plane aligned with the right-hand rule. The important variable in this project is the angular velocity around z. This is the variable that is being used as measurement for orientation in the xy-plane. From the plot you can see that this value varies between $[1 - 1.25)]$. In other words, an offset of $\dot{z}_{offset} = 1.125$ and a deviation of $\sigma_{\dot{z}} = \pm 0.125$. These values was later used as initial values for calibrating the gyroscope and prevent it from drifting.

10.3 Integration Tests Gyroscope and Slowdown Algorithm

The initial tests of the system revealed that there were inaccuracies in the orientation estimate while moving. Consequently, gyroscope measurements were implemented into the estimator. More specifically, into the turning mode of the estimator. Additionally, due to overshoots, a slowdown was implemented into the controller. This addition to the estimator allowed the system to have lower stopping threshold, also called angleThreshold, while rotating. It was reduced from 5° to 3°.

Two tests were performed on the implementation of gyroscope measurements into estimator and slowdown algorithm, a square test and a line test.

The square test was conducted mainly to test the gyroscope measurements, but it contained the slowdown algorithm implementation. It was conducted by sending the same four waypoints to the system as in the initial tests. Figure 25 shows the results of the test. It includes the four way points, the starting and end point of the robot and the 2D positions obtained using Opti Track Motive.

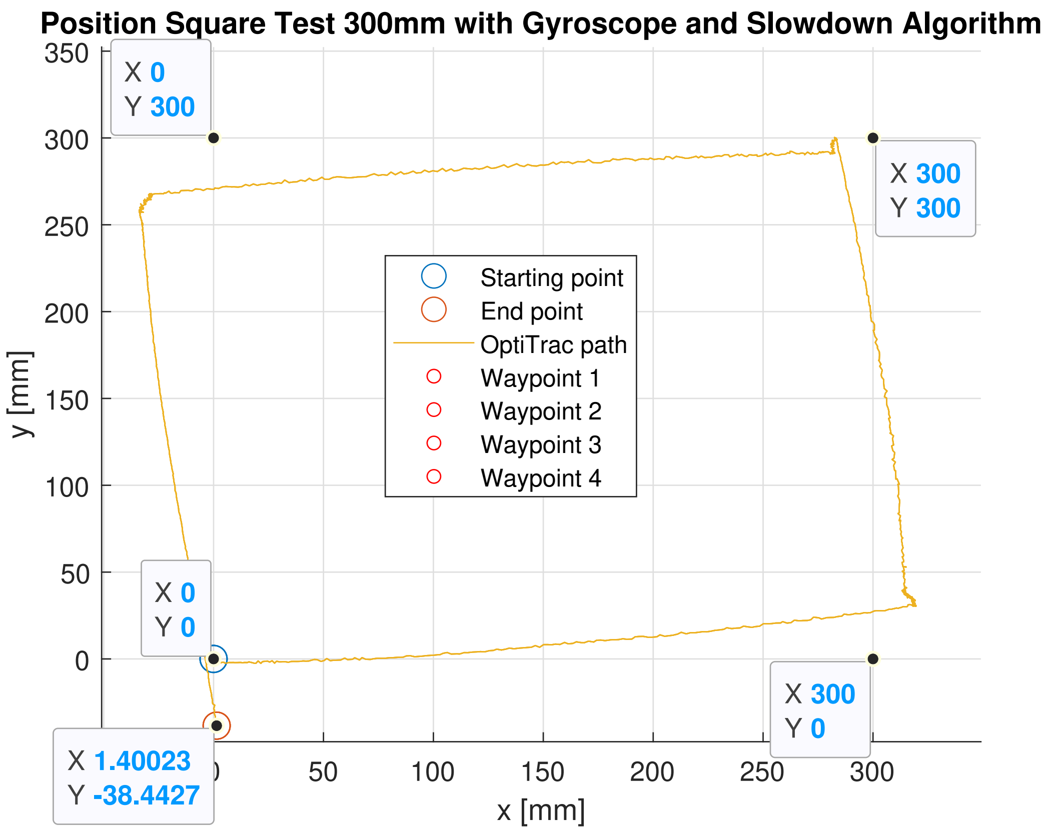


Figure 25: Gyroscope and Slowdown Algorithm Position Square Test 300 mm.

In the plot you can see that the path is much more precise than in Figure 22. The robot overshoots much less with every waypoint and this is likely due to the slowdown algorithm. Moreover, it rotates in a more consistent way. The shape of the path resembles a square much more so than in the initial test square test in Figure 22. It also reaches its end destination within the 50mm radius. However, the robot still diverges from a straight line while moving.

The line test at this stage of development was conducted to see the differences in overshoot due to the slowdown algorithm. The system had the gyroscope implemented into the estimator, but not into the move forward part. In other words, since the robot already had the correct orientation at the start of the test, this part of the implementation should not affect the results. The result of the line test is shown in Figure 26. It contains the starting and end point, the way point command and the 2D positions tracked by Opti Track Motive.

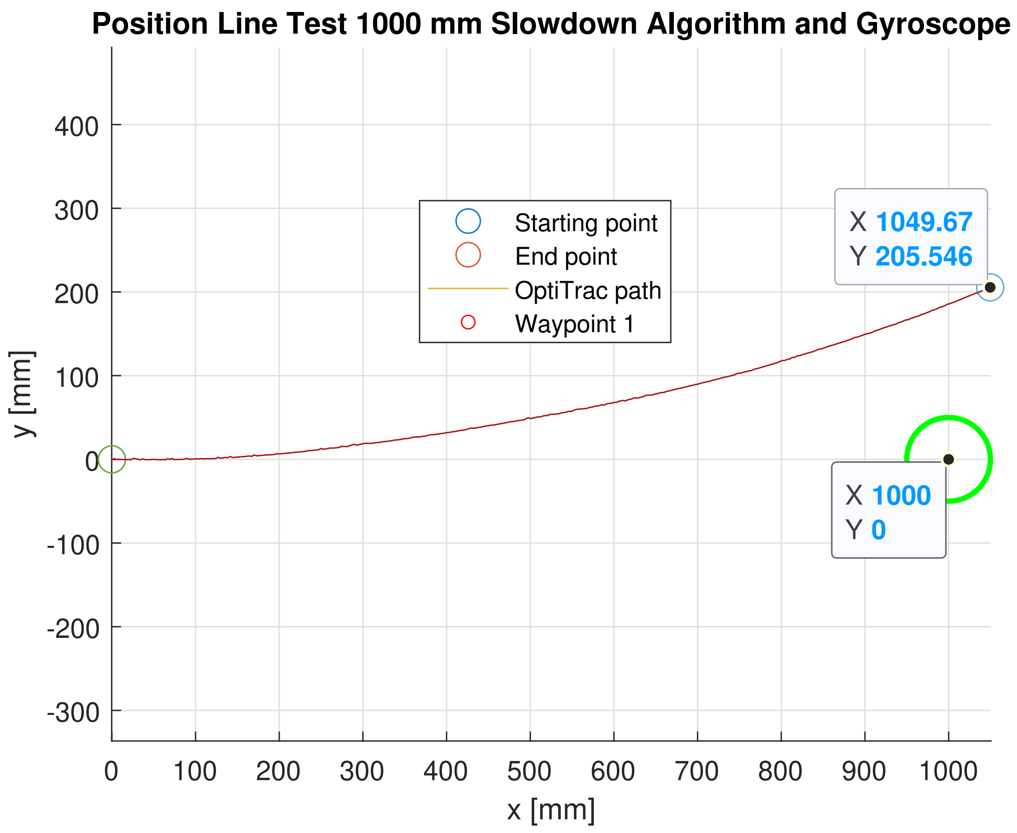


Figure 26: Position Line Test 1000 Slowdown Algorithm and Gyroscope.

In the plot you can see that the slowdown algorithm actually affected the system behaviour in a negative way when it came to line tracking. The end destination is further away from the target than in the initial system with about 210 mm. It also missed the end destination zone by a larger margin. However, it overshot less in x direction. From this plot you could argue that the slowdown algorithm weakens the robots ability to move straight, but no PID-controller had been implemented yet. At this point the only regulation for moving in a line was an offset calibration made due to the left motor being a little bit stronger than the right motor.

10.4 Integration Test P-regulator

To prevent the robot from diverging from a straight line, a PID controller was to be implemented. The first step towards this task was to implement the proportional term so that the robot would act if it deviated its goal orientation.

Tests were done on the magnitude of the proportional term P. Additionally, the bias on each motor were removed to make the P-regulator do the regulation on its own. This bias equates to +1 duty cycle %. on the right motor. The test were performed in the same way as the previous line tests, but they were 3000mm long. This was done to get a higher resolution of error. Three tests were done on the P-regulator with proportional gains 0,20 and 40. In this plot estimates was been added. The end destination zone was also increased to a radius of 100mm. The results of the P regulator test is shown in Figure 27.

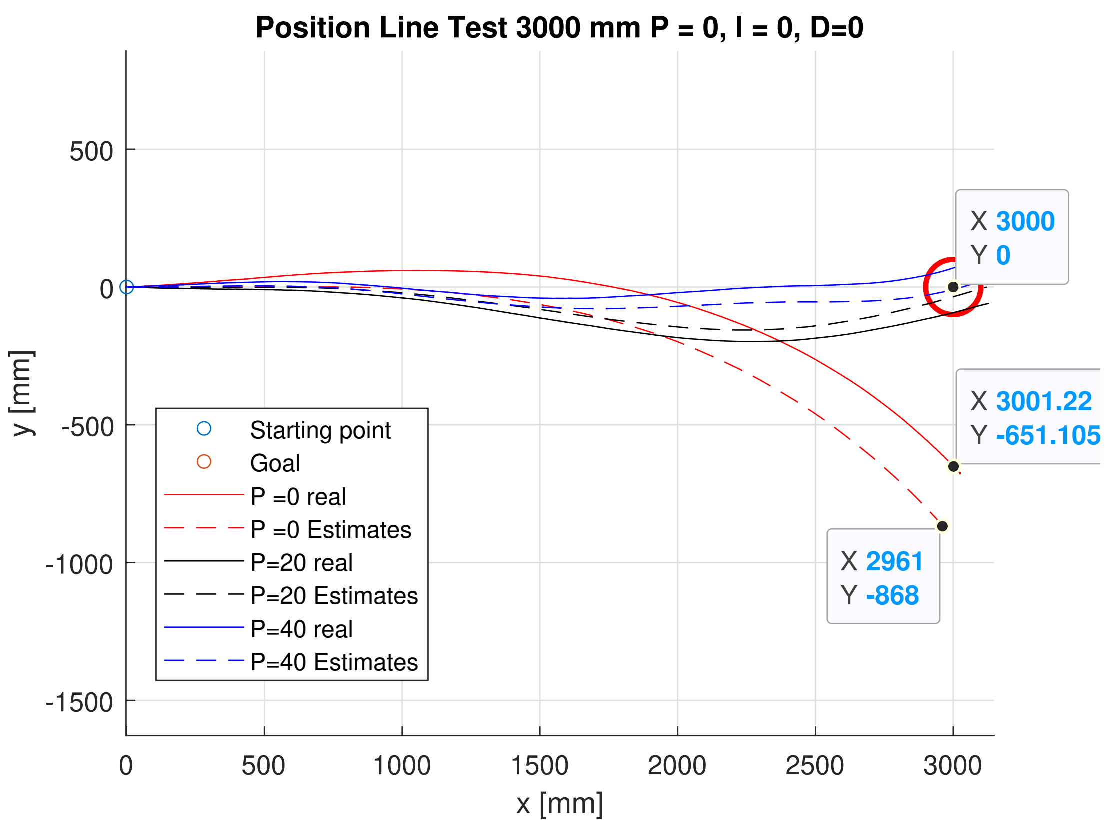


Figure 27: Position Line Test 3000m P Regulator

In the plot you can see that the P-regulator improves the system by regulating the amount of input to each wheel. When the proportional gain was 0, it ended about 650mm from its target destination. By adding a proportional gain of 20 or 40, the robot managed to cross inside the end destination zone. However, it is seen that the estimates deviates a little bit from its actual position. This deviation partly was caused by a bug in the estimator that will be discussed later.

10.5 Integration Testing PI-regulator

A P-regulator will not be able to reach set point perfectly without a stationary deviation caused by non-linear effects. To reduce an error like this you can use integral action. This is the difference between a PI- and P-regulator.

After the integral action was implemented and the bug involving the estimator was fixed, new 3000mm line tests were conducted to test the effect of integral action. The tests were performed in the same manner as the line test of the P-regulator. The P term was set to 20 and the integral term tested 0,20 and 40. The results of the tests are shown in Figure 28 and Figure 29.

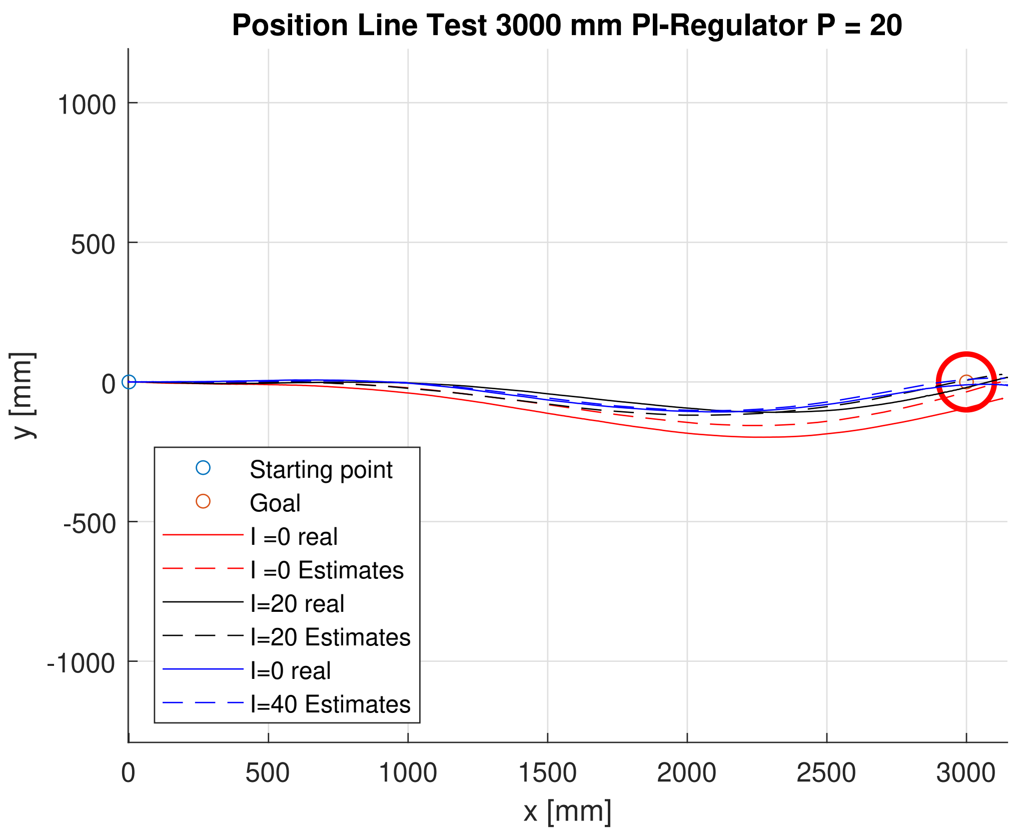


Figure 28: Position Line Test 3000mm P = 20. PI-regulator

Figure 28 shows the whole path the robot takes along with its estimates. It is clear that the integral action improves the system behaviour. The graph with no integral action is the graph that deviates the most from its end destination, and the graph with $I = 20$ (blue) seems to give the best response. However, the robot deviates quite a bit from around 1000mm to about 2000m in the y direction. With Integral action being a slow effect it manages to correct this deviation over time.

Figure 29 shows a zoomed version of the end destination zone.

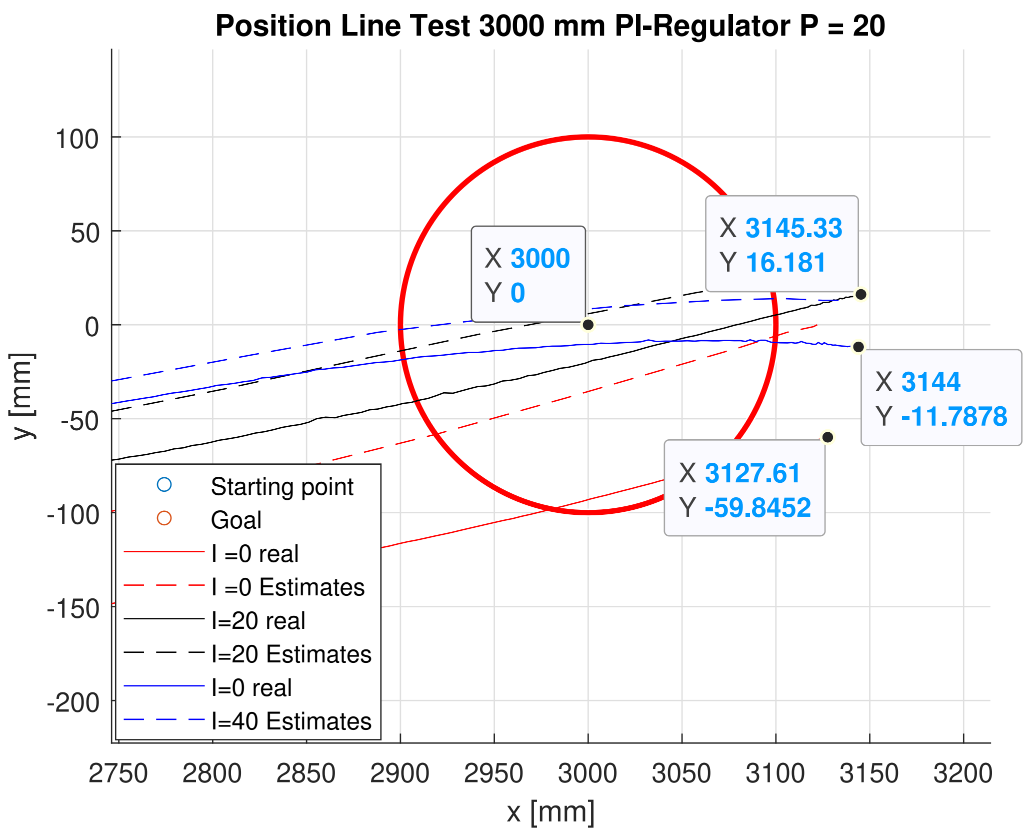


Figure 29: PI-reg plot Zoomed at end destination.

In this figure it is easier to see the end destination positions. It is close between $I = 20$ and $I = 40$, but $I = 40$ seems to get closer to the centre of the end destination. Additionally, it seems to settle better than $I = 20$.

10.6 Integration Testing PID-regulator

A PI controller is not able to tell how fast it is going by its own. In most cases it will overshoot its target if sudden changes happens. To avoid this kind of behaviour, a derivative term can be implemented. The derivative term can help anticipate the error. The derivative term is the difference between a PI-regulator and PID-regulator. PID is in most cases better than a PI-regulator. However, if the derivative term is not tuned properly it can cause the system to become unstable.

After PID-controller was implemented, a three tests were conducted with the derivative term. The first tests were 3000mm line tests with $P = 20$, $I = 40$, and $D = 20$ and $D = 40$. The results of the tests can be seen in Figure 30. The second test was a line test 3000mm. It was conducted after tuning in a satisfactory response with the derivative term, and is shown in Figure 31 and Figure 32. The third test was a square 1000mm test of the tuned PID. It is shown in Figure 33.

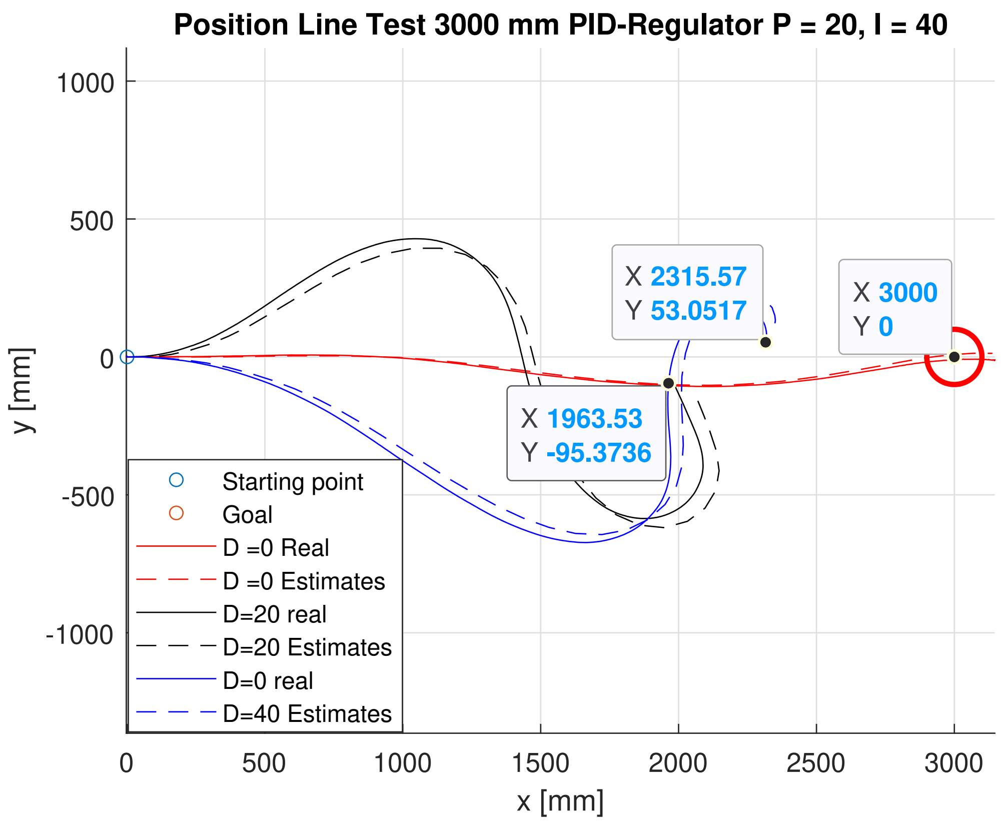


Figure 30: PID regulator instability with derivative Term.

In the plot you can see that the system becomes unstable because of the derivative term, indicating that the derivative term is too large. It never reaches its end destination because it takes oscillates too much around the target angle.

An attempt to tune the PID terms was conducted. The tuning was done using a trial and error method, and trying to find values that worked together. A tuning that worked is shown in Figure 31 and Figure 31.

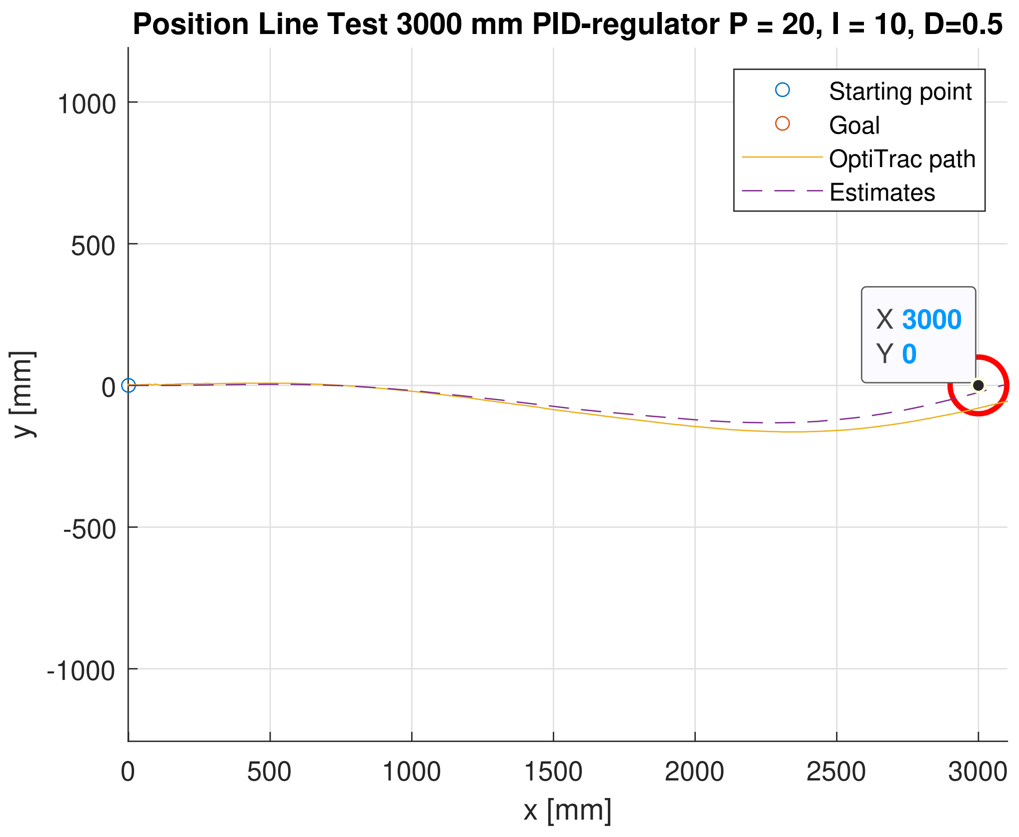


Figure 31: Line Test Position PID-regulator P = 20, I=10, D=0.5.

At first glance it does not look like it is much of an improvement to the tuned PI controller. However, it stops much closer to the end destination. The derivative term works as a damping effect. It is better shown in Figure 32.

Figure 32 shows a zoomed in version of Figure 31 at the end destination.

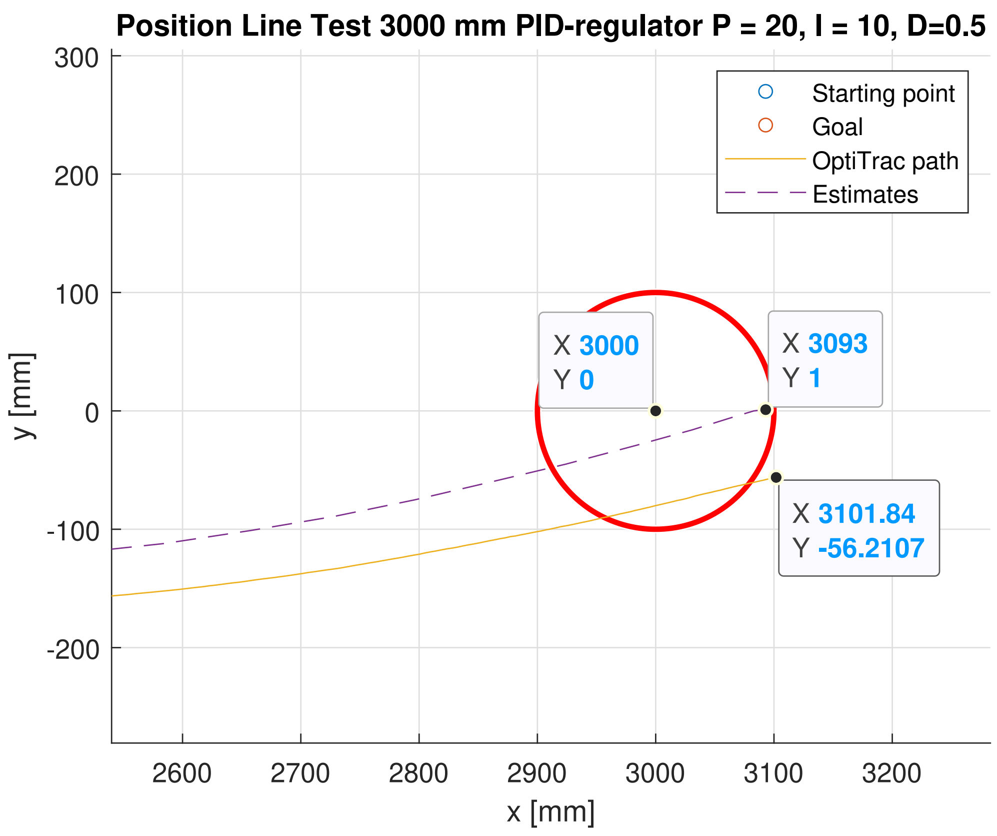


Figure 32: Line Test Position PID-regulator P = 20, I=10, D=0.5 Zoomed.

In the plot you can see that both the estimate and the Opti Track value enters the end destination zone and stops very close to it. The estimate manages to stop inside the zone while the Opti Track Motive real value does not. It is also seen that there is an error between the estimate and the real value.

Figure 33 show the last integration test of the tuned PID that was conducted, a 1000mm square test. The test was shows all the waypoints given to the robot, the real values from Opti Track Motive and the estimates.

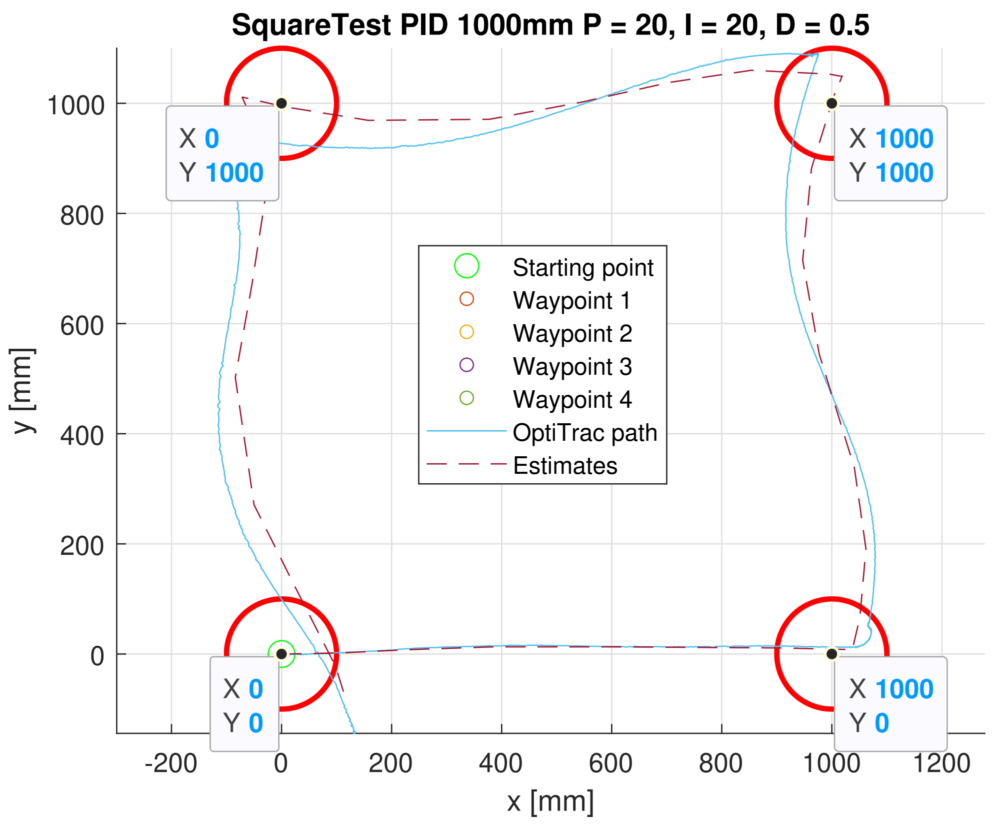


Figure 33: Square Test Position PID-regulator P = 20, I = 10, D = 0.5.

In the plot you can see that the line movement of the robot is very accurate going towards the first way point. Additionally, it lands inside the end destination zone. However, when going towards the second waypoint it struggles a bit to find the angle towards the target, hence the little bump at the second waypoint. It then tries to regulate for the error that occurs after rotating and manages to swing into the destination end zone. The shape of path reveals that the robot is regulating movement to correct for error, but the lines are not very straight.

10.7 Integration Test Server

After the implementation of the control system, the last part of the project was to confirm that it worked together with the server. This required modification of vApiTask, later renamed to vControllerApiTask(). To confirm that the implemented code was working together with the Java server, two server integration tests were performed. The tests was performed by sending the robot set point commands from the server. Two tests were done with the server to confirm it was working with the server. The estimate of the robot was not logged in this instance and the ground truth value was not logged with opti track. The server was st in manual drive and initialized in $X = 0$, $Y = 0$ and $\theta_{robot} = 0$

The first test was as line test 3000mm. The robot was send a command to position (3000mm,0mm). The server was then used to track the position. Figure 34 shows the results of the test. The red X is the starting point and the red circle is the NRF 3 robot.

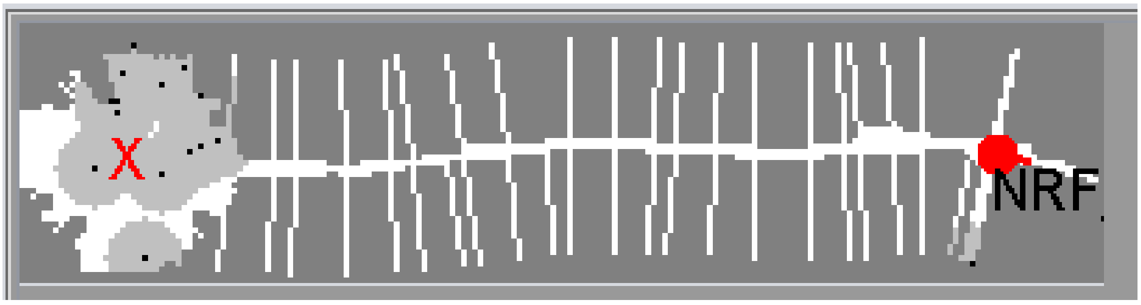


Figure 34: Server integration test 3000mm.

From the picture you can see the path that the robot took. Unfortunately, the server map has not implemented a axis system to confirm the accuracy, but it can be seen that the robot moves in a straight line.

The second test was a square test. The set point commands were (1000mm,0mm), (1000mm,1000mm), (0mm,1000mm) and (0mm,0mm). The server was used to track the result.Figure 35 shows the results from the server. The blue X is the starting point and the blue circle is the NRF 3 robot. The white area is area mapped by the IR sensors to the map.

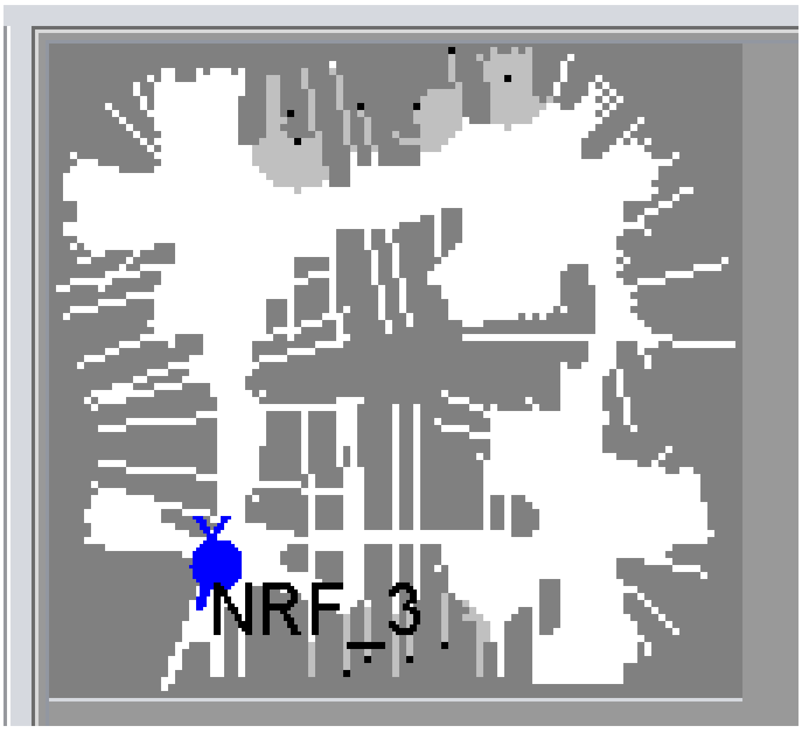


Figure 35: Server integration test square test 1000mm.

It can be seen that the robot lands very close to the starting position. It is moving in a square and seems to work as intended in this test.

11 Discussion

The GitHub organization is a great addition to the SLAM project. It makes it easier for students to develop the project going forward by version control, making the project more modular and easier to access to needed information. The README files guides new students to the information they need and hopefully this makes it easier to get started. This part of the project was not initially part of the thesis, but was discussed and added later on.

The specialization project fixed some of the initial issues with the system behaviour. It was now able to receive multiple commands and move in a predictable manner. Previously, in the specialization project there was no documentation of the ground truth values of the robot. The plotted values were from the estimates of the robot. However, this changed with Opti Track Motive which made it easier to analyze the movement of the robot by allowing for a close to ground truth tracking of the robot pose.

The robot uses its sensors to interact with the environment. It perceives by using IR sensors, gyroscope and encoders. These sensors does not provide perfect data and that is eventually affect the system behaviour of the robot. In other words after running for a while, it will drift, and would need a way to correct itself by using some sort of aiding measurement. Adding such a sensor to the robot would be an interesting addition to the robot control system.

Loop time in the system was a challenge while developing the control system for the matlab-robot. In order to implement the integral error and the derivative error this time was needed to be tracked because it was not always constant.

The robot system is designed in a way such that it rotates first then it moves straight towards the target. This way of controlling the robot is simple and intuitive but, it might not be the most efficient solution. In cases where the robot is only slightly off target, it would be able to fix this error while driving without having to first rotate towards the target. However, it was chosen because of its simplicity.

In the specialization project, a fail-safe constant that was implemented to prevent the robot from moving longer than this length between every set point command. This fail-safe has now been changed into a variable that takes the distance between position estimate at the time of receiving and the incoming set point into account. To implement this variable it was needed to save the position at the time of receiving the command until another command was received. A caveat to this fail-safe is that it can cause the robot to stop prematurely if the robot takes a path that involves a lot of curves. This behaviour can be seen in Figure 30 where the robot takes changes its orientation multiple times during its movement.

A slowdown algorithm was put in place to reduce overshoot when being closer to the target. This variable ended up weakening the system's ability to move straight initially, but after was fixed after the implemntation of PID. The reason for this is that the smallest differences in the strength of each motor matters more the slower the robot moves. The robot has a left motor that is slightly stronger than the other, and this difference becomes more noticeable at lower inputs.

The addition of gyroscope values greatly increased the accuracy of the orientation of the robot. This was seen in the integration test of the gyroscope and made it possible to implement a lower threshold for stopping while rotating than with encoder estimates. Something that was not tested during this project was to use the gyroscope for robot orientation while moving the robot. This could be worthwhile to test as the gyroscope seems to be very accurate for use in robot orientation.

The gyroscope calibration was implemented in a very simple way. For the first 400ms of the program, the robot will stay at a standstill and gather the offset values needed for it to not drift. However, since it only takes in one value, this maybe a value that is either high, medium or low. In other words, it may not represent the true mean value of the gyroscope perfectly. A way to fix this issue would be to log a larger amount of gyroscope values at the start and then take the mean values instead. This is a potential source of error for drift in the gyroscope.

A problem with the current version of the NRF52 robot is the way the drivers are written. This was discovered late while implementing the PID controller in this project and was therefore not prioritized to fix. The motor driver for the DC motors takes in integers, which by itself causes problems with resolution while developing a controller. In other words, 0.9 and 0 is the same number, which is not ideal when you want to drive the motors in a way that make them exert the same force. Moreover, these integers are saturated at 50 % duty cycle to protect the motors from taking damage. This means that the robot controller needs to take this into account when doing regulation and this creates a dependency between the controller and driver, which is not ideal. Another issue that has not been taken into account is that the robot does not move at all if the motor exerts under 9 % duty cycle on each motor. In other words the actual range for controlling the robot movement is [9,50]. In the current MATLAB generated control system these limitations have been accounted for, but a better solution to this problem is to map these values from 0 to 100 at the driver. Also increasing the resolution from 50 different values to 1000+ would make tuning and calibration much easier.

A unforeseen bug involving the estimator was revealed during testing. This bug included the orientation part of the estimator. More specifically, the dTicks variable, that is being used to calculate the change in orientation in that sample. The bug was a resolved by changing the argument inside the the sign-function from (14) to (13).

$$dTicks = sign(sTicksRight) \cdot abs(sTicksRight - sTicksLeft)/2 \tag{13}$$

$$dTicks = sign(sTicksRight - sTicksLeft) \cdot abs(sTicksRight - sTicksLeft)/2 \tag{14}$$

The control system of the robot uses two different values to regulate the position of the robot, uL and uR. In other words, the PID has multiple output. The reason for this way of implementation was that it was an intuitive way of thinking about the control problem. If the robot diverges to the left, the left motor should exert more force and vice versa. However, since there are two control input, it becomes more complex as you would have to find a way to alternate between these motor output. The solution became to use a control terms for switching between the motors, for instance dirProportionalControlLeft (15).

$$dirIntegralControlLeft = (1 - sign(thetaIntegralError))/2 \tag{15}$$

This term takes the direction of where the robot is heading into account when giving a proportional input to the robot. Terms like these were used for all the PID gains. Another reason for this implementation has to do with the non linear behaviour of the drivers. As mentioned before, the DC motors will not move the robot if they are given lower than 9%. Meaning that if the robot was driven by for example voltage difference between the motors, it would risk dropping below this value and this would cause instability. A way to avoid this problem would be to map the values between 9 and 50. Modifying the system to use voltage difference could be another solution to the problem.

The introduction of proportional term to the controller reduced the amount of error while moving towards the target. By adding this term, the robot was able to move 3000mm and land in the end destination zone. However, it did not manage to land inside the zone.

The introduction of the integral action managed to work as a fine tune term for the regulator. Integral action added extra complexity to the system, but improved the performance. It required the implementation of anti windup in the task code and a saturation term to avoid integrator windup. During testing, this term would destabilize the system as it would wind up the integral error. To fix this problem, a integral reset was implemented in the task code when receiving a new command. Additionally, a saturation on the magnitude of the input from integral action was implemented.

The introduction of the derivative term added even more complexity to the system. It is dependent on the sample time of the task and requires the controller to remember the previous theta error. It was not as easy to tune as the PI-gains because the system was very sensitive to its contribution. It was implemented to resist sudden changes in control input and when tuned with the other variables, it managed to improve the system.

A problem that occurred while testing the PID regulator was when the robot was far away from the target it would not as easily regulator for deviation. When the robot was far away from the target, the target angle was naturally very small for long. The robot would simply not react to small changes that caused drift before it was too late and erratic behaviour occurred when it got too close. This would be especially be the case if the robot moved passed the end destination zone.

The estimator deviates from the Opti Track results. This can be seen from the results. It is unknown exactly why this is, but one could speculate that it has something to do with the estimator model limitations. The model will simply not know if the robot wheel slips or if the robot starts of with a slightly different orientation than the Opti Track orientation.

One part of this project was to explore the functionality of the MATLAB to C code generation. It has its limitations in some areas where it cannot generate all of its function to C code. In the specialization project, it was needed to implement a custom modulus function in MATLAB to account for the floating point differences between C and MATLAB. Moreover, while researching PID control in MATLAB in this thesis it was discovered that the MATLAB implemented PID functions did not have the extended capabilities of being generated to C code. That is also the reason why it was implemented in the way it was.

An alternative way to design the move_forward() was considered. Instead of adding input to the motors separately, it involved using the voltage difference to regulate the orientation of the robot. However, this method was dismissed because of limitations involving the motors needing 9% duty cycle to move and was saturated at 50 %. If the robot was operating inside this range at all times, it would not be a problem, but because the robot ideally should slow down when being close to the target, one of the motors would stop abruptly.

The robot is using orientation towards the target as setpoint for its control. An alternative way to implement the PID controller could be to make a line between the position of the robot and the goal position and make the robot follow this line. This concept is called trajectory tracking. The reason for not using this method in this project, was that the method involving the target orientation set point was more intuitive to implement. However, a method involving trajectory tracking may give a better performance if implemented. It would require a refactoring of the move_forward() function.

Some time was spent to implement a method to log the estimate pose of the robot. As a short-term solution it was done by creating arrays that would log the x and y position and then reading the output after. This method worked if the test was short, but since RAM memory in the nrf52840 is 256 KB, it was not suitable as a method to log long-term. A solution to this problem may be to utilize the QSPI.

The use of TaskDelay(ms) was minimized vControllerApiTask() and replaced with TaskYield(). Previous implementation of vControllerApiTask() included excessive use of TaskDelay(ms). This caused the robot to sometimes miss deadlines that was cruicial. By replacing these functions with TaskYield(), other tasks was able to finish their code and timing issues were resolved.

The estimator() is still using wheel encoder ticks to evaluate the orientation of the robot while moving forward. Wheel play and wheel slip are possible errors that can occur because of this. An idea that was never tested during this project was to implement the sThetaGyro into the move_forward() function, i.e. completely rely on the gyroscope measurements for all robot orientation. It will not solve all the issues occurring from wheel slip, but may offer a more accurate θ_{robot}.

12 Further Work

Logging

The method that was used during this project to log estimates on the robot will not suffice for longer tests due to limitations with RAM-memory.

Implementing a way to utilize the QSPI of the nRF52840 for logging will be beneficial for the future. This will could also be beneficial for the SLAM project as a whole as it would offer the possibility to do offline-SLAM and data logging.

Java Server

The java server interface is easy to use, but needs to be updated and tested.

During this project, only one robot was used in conjunction with the server. But is the java-server able to handle multiple robots while maintaining connection?

Motor drivers

The motor driver code needs to be modified such that it displays the full range of the motors. The motor drivers should map values into the [9,50] range because of physical limitations. Additionally the resolution should be increased if possible, as of now it can only be tuned with integers. These changes will make it easier to make controllers for other developers.

PID-controller

The PID controller in move_forward() is not yet tuned to perfection. To improve the controller with its current design, one would need to perform more tuning on the controller gains, kP,kI,kD.

Estimator()

The estimator may be improved by using the gyroscope values for orientation while moving forward. It still remains to be tested. To do this, sTheta would have to be replaced with sThetaGyro in move_forward(). Another idea could be to implement a kalman-filter that would utilize both measurements. Since the encoder ticks method seem to be fairly accurate over shorter distances, one could increase the uncertainty of the encoder ticks with distanceDriven. The system behaviour is highly coupled with how good the estimator is, so making this better is very important.

Additionally, the calibration of the gyroscope measurements could be implemented in a better way by sampling a large amount of measurements and taking the mean of the whole sample size.

New features

Suggestions to additions to the system or project:

- Return to home algorithm.
- Collision avoidance.
- Aiding measurement
- Maze discovery algorithm.

A return to home algorithm would entail a method to return to the starting point if connection is lost.

Collision avoidance is partly implemented in vControllerApi(), but in a very primitive way. An ideal collision avoidance implementation would make the robot detect the obstacle and then move way from it. This feature would have to be implemented in a such that it would still reach its end target if physically possible.

An aiding measurement would be something that could prevent drift in the robot. A way to correct the estimate if it has drifted. You would need a sensor with high precision and a reference point in order to achieve this, but it could be an interesting addition to the estimate.

A way to take this project further could be to implement maze discovery in the system. The idea would be to try to make the robot solve a maze on its own. This could be done with the current system or a new iteration of the hardware. The concept of micromouse is widely known, and studying the history and development of competitions involving micromouse maze solving would be a good starting point.

Bibliography

Andersen, Thomas (2022). 'Sparse IR sensor EKF-SLAM for MQTTSN/Thread connected robot'. In: *NTNU Department of Cybernetics*.

Berglund, Gabriel (2021). 'Investigation of performanceenhancing measures for a nonholonomic differential drive robot'. In: *NTNU Department of Cybernetics*.

Ese, Erlend (2016). 'Sanntidsprogrammering på samarbeidande mobil-robotarn'. In: *NTNU Department of Cybernetics*.

Frestad, Henning (2022). 'The SLAM-Project'. In: *NTNU Department of Cybernetics*.

Gilje, Maria (2022). 'Implementing Bluetooth LE on a MATLAB controlled nRF52840 robot'. In: *NTNU Department of Cybernetics*.

Github (2023). *About organizations*. URL: https://docs.github.com/en/organizations/collaborating-with-groups-in-organizations/about-organizations (visited on 12th June 2023).

Instruments, Crystal (2023). *PID Control Theory*. URL: https://www.crystalinstruments.com/blog/2020/8/23/pid-control-theory (visited on 12th June 2023).

Jølsgård, Eivind (2020). 'Embedded nRF52 robot'. In: *NTNU Department of Cybernetics*.

Kolbeinsen, Magnus Isdal (2022). 'Set Point Control of Robot- and Server System using MATLAB generated C Code'. In: *NTNU Department of Cybernetics*.

McConnell, Steve (2004). *Code Complete 2nd edition*. Microsoft Press.

Motive, Opti Track (2023a). *Motive Optical motion capture software*. URL: https://optitrack.com/software/motive/ (visited on 12th June 2023).

— (2023b). *Rigid Body Tracking*. URL: https://docs.optitrack.com/motive/rigid-body-tracking (visited on 12th June 2023).

Rødseth, Mats Gjerset and Thor Eivind Svergja Andersen (2016). 'Arduino Robot and Server Application'. In: *NTNU Department of Cybernetics*.

Sakovich, Natalita (2023). *Real-Time Embedded Systems: What Are They and Why Do We Need Them?* URL: https://shorturl.at/tFJS6 (visited on 12th June 2023).

Segura, Thomas (2023). *GitHub Organizations Best Practices*. URL: https://gitprotect.io/blog/github-organizations-best-practices/#:~:text=Organizations%5C%20allow%5C%20administrating%5C%20of%5C%20multiple,assume%5C%20various%5C%20roles%5C%20within%5C%20it (visited on 12th June 2023).

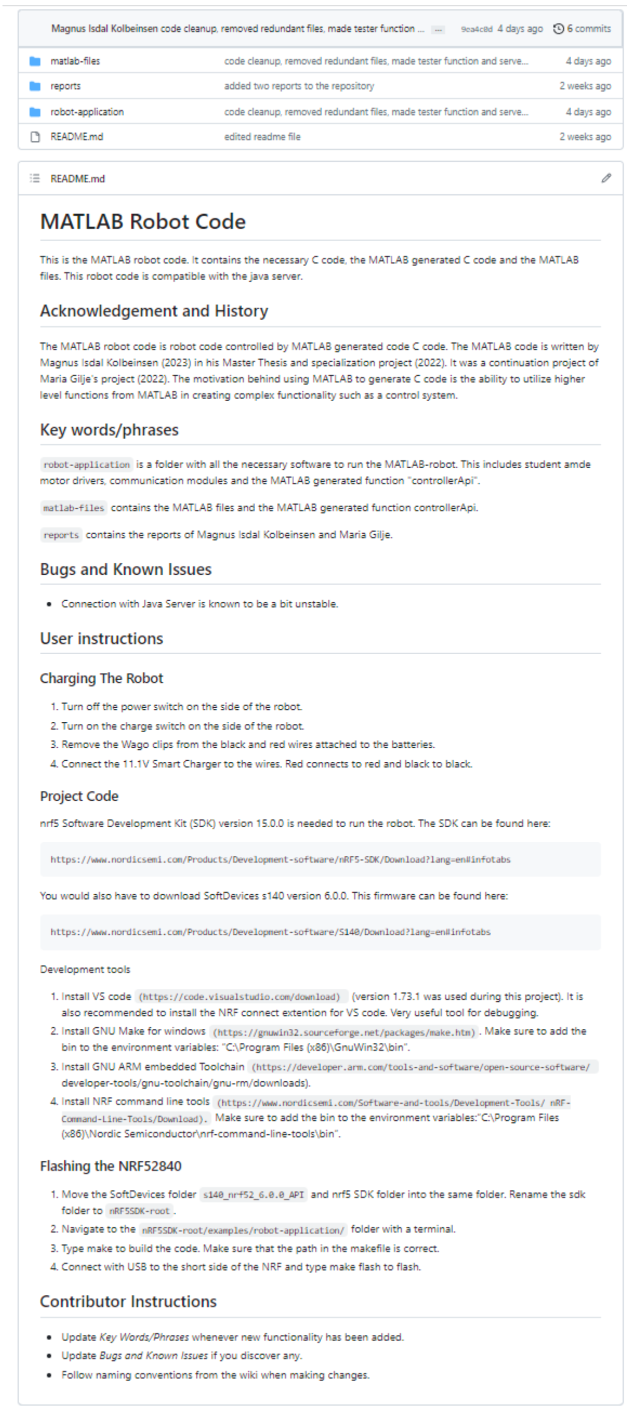
Syvertsen, Bjørn (2006). 'Autonom legorobot'. In: *NTNU Department of Cybernetics*.

Thon (2006). 'Simulation, Mapping and Naviation for Server Application'. In: *NTNU Department of Cybernetics*.

Appendix

A README-files Github Organization

A Matlab-robot-README



Magnus Isdal Kolbeinsen code cleanup, removed redundant files, made tester function ... 9ea4c8d 4 days ago 6 commits

matlab-files	code cleanup, removed redundant files, made tester function and serve...	4 days ago
reports	added two reports to the repository	2 weeks ago
robot-application	code cleanup, removed redundant files, made tester function and serve...	4 days ago
README.md	edited readme file	2 weeks ago

README.md

MATLAB Robot Code

This is the MATLAB robot code. It contains the necessary C code, the MATLAB generated C code and the MATLAB files. This robot code is compatible with the java server.

Acknowledgement and History

The MATLAB robot code is robot code controlled by MATLAB generated code C code. The MATLAB code is written by Magnus Isdal Kolbeinsen (2023) in his Master Thesis and specialization project (2022). It was a continuation project of Maria Gilje's project (2022). The motivation behind using MATLAB to generate C code is the ability to utilize higher level functions from MATLAB in creating complex functionality such as a control system.

Key words/phrases

`robot-application` is a folder with all the necessary software to run the MATLAB-robot. This includes student amde motor drivers, communication modules and the MATLAB generated function "controllerApi".

`matlab-files` contains the MATLAB files and the MATLAB generated function controllerApi.

`reports` contains the reports of Magnus Isdal Kolbeinsen and Maria Gilje.

Bugs and Known Issues

- Connection with Java Server is known to be a bit unstable.

User instructions

Charging The Robot

1. Turn off the power switch on the side of the robot.
2. Turn on the charge switch on the side of the robot.
3. Remove the Wago clips from the black and red wires attached to the batteries.
4. Connect the 11.1V Smart Charger to the wires. Red connects to red and black to black.

Project Code

nrf5 Software Development Kit (SDK) version 15.0.0 is needed to run the robot. The SDK can be found here:

```
https://www.nordicsemi.com/Products/Development-software/nRF5-SDK/Download?lang=en#infotabs
```

You would also have to download SoftDevices s140 version 6.0.0. This firmware can be found here:

```
https://www.nordicsemi.com/Products/Development-software/S140/Download?lang=en#infotabs
```

Development tools

1. Install VS code `(https://code.visualstudio.com/download)` (version 1.73.1 was used during this project). It is also recommended to install the NRF connect extention for VS code. Very useful tool for debugging.
2. Install GNU Make for windows `(https://gnuwin32.sourceforge.net/packages/make.htm)`. Make sure to add the bin to the environment variables: "C:\Program Files (x86)\GnuWin32\bin".
3. Install GNU ARM embedded Toolchain `(https://developer.arm.com/tools-and-software/open-source-software/` developer-tools/gnu-toolchain/gnu-rm/downloads).
4. Install NRF command line tools `(https://www.nordicsemi.com/Software-and-tools/Development-Tools/ nRF-Command-Line-Tools/Download).` Make sure to add the bin to the environment variables:"C:\Program Files (x86)\Nordic Semiconductor\nrf-command-line-tools\bin".

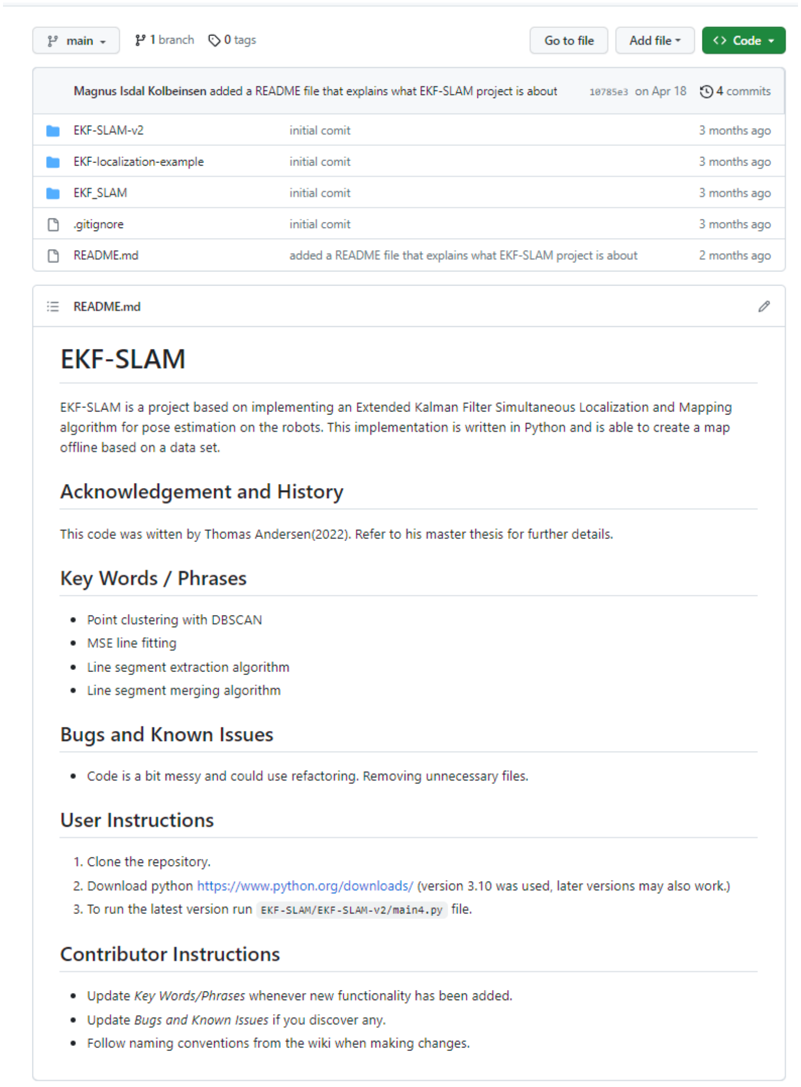
Flashing the NRF52840

1. Move the SoftDevices folder `s140_nrf52_6.0.0_API` and nrf5 SDK folder into the same folder. Rename the sdk folder to `nRF5SDK-root`.
2. Navigate to the `nRF5SDK-root/examples/robot-application/` folder with a terminal.
3. Type make to build the code. Make sure that the path in the makefile is correct.
4. Connect with USB to the short side of the NRF and type make flash to flash.

Contributor Instructions

- Update *Key Words/Phrases* whenever new functionality has been added.
- Update *Bugs and Known Issues* if you discover any.
- Follow naming conventions from the wiki when making changes.

B EKF-SLAM-README



Magnus Isdal Kolbeinsen added a README file that explains what EKF-SLAM project is about 10785e3 on Apr 18 4 commits

EKF-SLAM-v2	initial comit	3 months ago
EKF-localization-example	initial comit	3 months ago
EKF_SLAM	initial comit	3 months ago
.gitignore	initial comit	3 months ago
README.md	added a README file that explains what EKF-SLAM project is about	2 months ago

README.md

EKF-SLAM

EKF-SLAM is a project based on implementing an Extended Kalman Filter Simultaneous Localization and Mapping algorithm for pose estimation on the robots. This implementation is written in Python and is able to create a map offline based on a data set.

Acknowledgement and History

This code was witten by Thomas Andersen(2022). Refer to his master thesis for further details.

Key Words / Phrases

- Point clustering with DBSCAN
- MSE line fitting
- Line segment extraction algorithm
- Line segment merging algorithm

Bugs and Known Issues

- Code is a bit messy and could use refactoring. Removing unnecessary files.

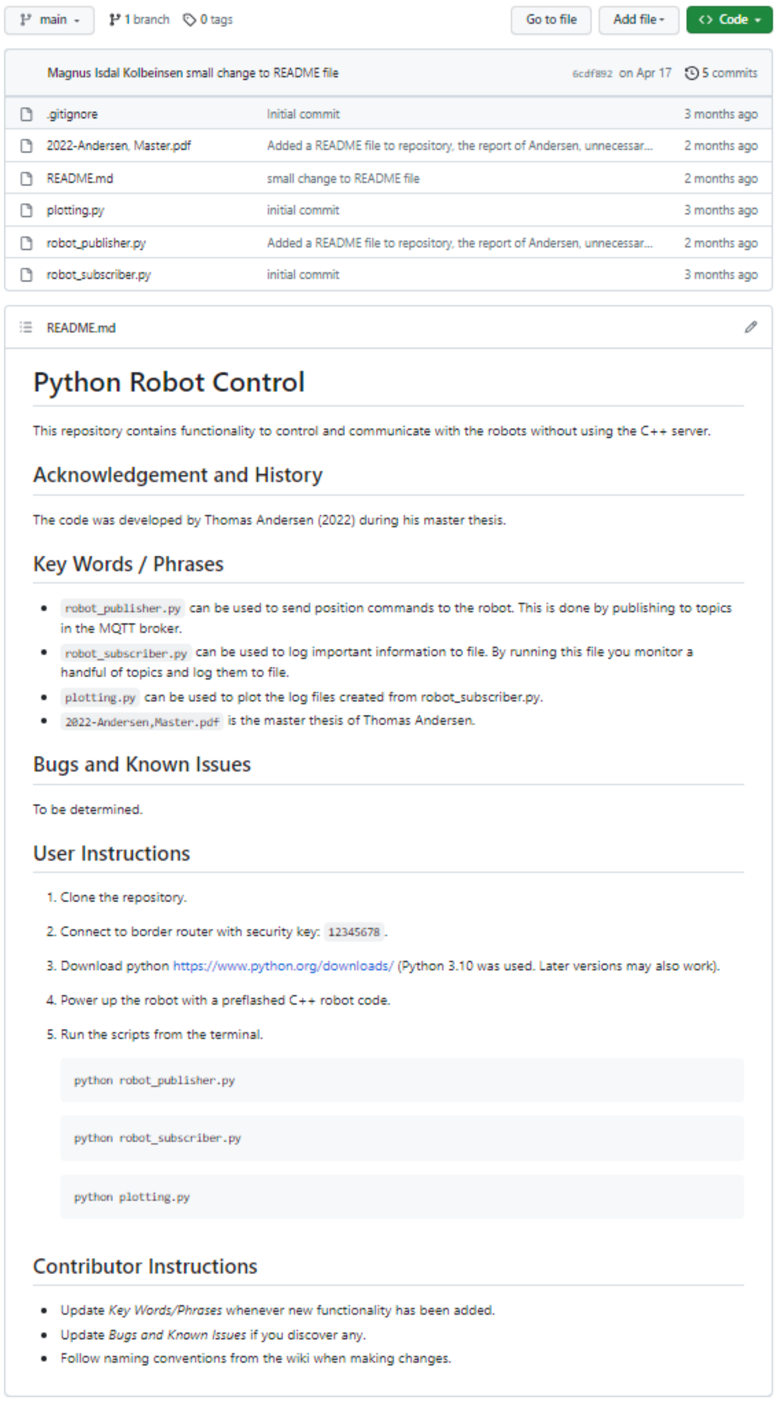
User Instructions

1. Clone the repository.
2. Download python https://www.python.org/downloads/ (version 3.10 was used, later versions may also work.)
3. To run the latest version run `EKF-SLAM/EKF-SLAM-v2/main4.py` file.

Contributor Instructions

- Update *Key Words/Phrases* whenever new functionality has been added.
- Update *Bugs and Known Issues* if you discover any.
- Follow naming conventions from the wiki when making changes.

C Python-robot-control-README



Magnus Isdal Kolbeinsen small change to README file 6cdf892 on Apr 17 5 commits

.gitignore	Initial commit	3 months ago
2022-Andersen, Master.pdf	Added a README file to repository, the report of Andersen, unnecessar...	2 months ago
README.md	small change to README file	2 months ago
plotting.py	initial commit	3 months ago
robot_publisher.py	Added a README file to repository, the report of Andersen, unnecessar...	2 months ago
robot_subscriber.py	initial commit	3 months ago

README.md

Python Robot Control

This repository contains functionality to control and communicate with the robots without using the C++ server.

Acknowledgement and History

The code was developed by Thomas Andersen (2022) during his master thesis.

Key Words / Phrases

- `robot_publisher.py` can be used to send position commands to the robot. This is done by publishing to topics in the MQTT broker.
- `robot_subscriber.py` can be used to log important information to file. By running this file you monitor a handful of topics and log them to file.
- `plotting.py` can be used to plot the log files created from robot_subscriber.py.
- `2022-Andersen,Master.pdf` is the master thesis of Thomas Andersen.

Bugs and Known Issues

To be determined.

User Instructions

1. Clone the repository.
2. Connect to border router with security key: `12345678`.
3. Download python https://www.python.org/downloads/ (Python 3.10 was used. Later versions may also work).
4. Power up the robot with a preflashed C++ robot code.
5. Run the scripts from the terminal.

```
python robot_publisher.py
```

```
python robot_subscriber.py
```

```
python plotting.py
```

Contributor Instructions

- Update *Key Words/Phrases* whenever new functionality has been added.
- Update *Bugs and Known Issues* if you discover any.
- Follow naming conventions from the wiki when making changes.

B vControllerAPiTask()

```
#include "ControllerApiTask.h"

#include "defines.h"
#include "nrf_log.h"
#include "FreeRTOS.h"
#include "functions.h"
#include "queue.h"
#include "semphr.h"

#include "server_communication.h"
#include "globals.h"
#include "ICM_20948.h"
#include "servo.h"
#include "robot_config.h"
#include "encoder.h"

#include "positionEstimate.h"

#include "motor.h"
#include "../drivers/display.h"
#include <stdio.h>

void vControllerApiTask(void *arg){

    //Constants
    #define COMPLETE 0
    #define GYRO_MIN 0.25

    vServo_setAngle(0);

    struct sCartesian Setpoint = {0, 0};
    double gX_hat = 0.0;
    double gY_hat = 0.0;
    double gTheta_hat = 0.0;
    double leftU = 0.0;
    double rightU = 0.0;
    uint8_t robotMovement = moveStop;

    //Wheel encoder
    double ticks_Left     = 0;
    double ticks_Right     = 0;
    double total_ticks_r   = 0;
    double total_ticks_l   = 0;

    //new controllerApi
    double turning=1;
    double setpointX = 0;
    double setpointY = 0;
    double newCommand = 1;
    double xprev = 0;
    double yprev = 0;
    double thetaprev = 0;
    double distanceDriven =0;
    double waitingCommand =0;
    double ddInitX = 0;
    double ddInitY = 0;
    gTheta_hat = thetaprev;
    double thetaIntegralError = 0;
    double thetaError = 0;

    //IMU calibration
    bool calibration = 1;
    TickType_t ticks_since_startup = xTaskGetTickCount();
    double offsetGyroX = 0;
    double offsetGyroY = 0;
    double offsetGyroZ = 0;
    double gyroAngleZ =0;
    double delta_theta_gyro =0;
    double sample_diff_gyro_z =0;
    double gyro_z_prev=0;


    //init test parameters
    int uL = 0;
    int uR = 0;
    float time_since_startup = 0;


    while (true) {
        taskYIELD();
        TickType_t ticks_since_startup_prev = ticks_since_startup;
        ticks_since_startup = xTaskGetTickCount();
        float delta_t = (ticks_since_startup - ticks_since_startup_prev)*1.0 /
↪   configTICK_RATE_HZ;
        time_since_startup = time_since_startup+delta_t;

        //Read IMU data
        IMU_reading_t gyro;
        // IMU_reading_t accel;
        IMU_read();
        gyro = IMU_getGyro();
        double gyro_x = gyro.x-offsetGyroX;
        double gyro_y = gyro.y-offsetGyroY;
        double gyro_z = gyro.z-offsetGyroZ;

        if (calibration){
            vTaskDelay(400);
            IMU_read();
            gyro = IMU_getGyro();
            offsetGyroX = gyro.x;
            offsetGyroY = gyro.y;
            offsetGyroZ = gyro.z;
            calibration = COMPLETE;
        }

        delta_theta_gyro = gyro_z*delta_t;

        if(!calibration && IMU_new_data() && (gyro_z>GYRO_MIN || gyro_z<-GYRO_MIN )){
            gyroAngleZ=gyroAngleZ+delta_theta_gyro;

        }

        gyro_z_prev =gyro_z;
        //double accel_x = accel.x;
        // accel = IMU_getAccel();
        // double accel_y = accel.y;
        // double accel_z = accel.z;


        if (xQueueReceive(poseControllerQ, &Setpoint, 0) == pdTRUE) {
            //New command has been received
            newCommand=true;
        }


        encoderTicks Ticks = encoder_get_ticks_since_last_time();
        ticks_Left = -(double)Ticks.left;
        ticks_Right = -(double)Ticks.right;
        total_ticks_r = total_ticks_r+ticks_Right;
        total_ticks_l = total_ticks_l+ticks_Left;

        //
        if (newCommand){
        turning = 1;
        setpointX = Setpoint.x;
        setpointY = Setpoint.y;
        waitingCommand = 0;
        newCommand = 0;
        ddInitX = gX_hat;
        ddInitY = gY_hat;
        thetaIntegralError=0;
        }


        //MATLAB generated function

        controller_api(setpointX, setpointY,  newCommand,
                &waitingCommand, ticks_Left, ticks_Right,
                &distanceDriven, &turning, xprev,
                yprev, thetaprev, ddInitX,
                ddInitY, delta_theta_gyro,
                &thetaIntegralError, delta_t,&thetaError,
                &gX_hat, &gY_hat, &gTheta_hat,
                &leftU, &rightU);


        // temp values of global states
        xprev     = gX_hat;
        yprev     = gY_hat;
        thetaprev = gTheta_hat;

        uL = (int)leftU;
        uR = (int)rightU;

        vMotorMovementSwitch(uL,uR);
        taskYIELD();

        xSemaphoreTake(xPoseMutex, 15);

↪   set_position_estimate_heading(gTheta_hat);
        set_position_estimate_x(gX_hat/1000); // convert from mm to m
        set_position_estimate_y(gY_hat/1000); // convert from mm to m
        xSemaphoreGive(xPoseMutex);
        taskYIELD();
        if (checkForCollision() == true){
                motor_brake();
                robotMovement = moveStop;
                xQueueSend(scanStatusQ,&robotMovement,0);
        }

        else{
            robotMovement = moveForward;
            xQueueSend(scanStatusQ,&robotMovement,0);
        }

    }
}
```

C vControllerApiTesterTask()

```
#include "ControllerApiTesterTask.h"

#include "defines.h"
#include "nrf_log.h"
#include "FreeRTOS.h"
#include "functions.h"
#include "queue.h"
#include "semphr.h"

#include "server_communication.h"
#include "globals.h"
#include "ICM_20948.h"
#include "servo.h"
#include "robot_config.h"
#include "encoder.h"

#include "positionEstimate.h"

#include "motor.h"
#include "../drivers/display.h"
#include <stdio.h>

void vControllerApiTesterTask(void *arg){

    //Constants
    #define COMPLETE 0
    #define GYRO_MIN 0.25

    //MACRO variables
    #define SQUARE 1
    #define LINE 2
    #define NO_TEST 0

    //MACROs for running different sequences
    #define DEBUG 0
    #define LOG 1
    #define TEST_TYPE SQUARE // SQUARE | LINE | NO_TEST

    // SQUARE test parameters
    #if (TEST_TYPE==SQUARE)
        int squareTestInterval = 150;
        int xarrayPos[150] = {0};
        int yarrayPos[150] = {0};
        double xWaypoint1 = 1000;
        double yWaypoint1 = 0;
        double xWaypoint2 = 1000;
        double yWaypoint2 = 1000;
        double xWaypoint3 = 0;
        double yWaypoint3 = 1000;
        double xWaypoint4 = 0;
        double yWaypoint4 = 0;
        double xWaypoint  = xWaypoint1;
        double yWaypoint = yWaypoint1;
    #endif

    // run straight line test
    #if (TEST_TYPE==LINE)
        double xWaypoint = 3000;
        double yWaypoint = 0;
        int lineTestInterval = 150;
        int xarrayPos[lineTestInterval];
        int yarrayPos[lineTestInterval];
    #endif

    #if (TEST_TYPE==NO_TEST)
        double xWaypoint=0;
        double yWaypoint=0;
    #endif


    vServo_setAngle(0);

    struct sCartesian Setpoint = {0, 0};
    double gX_hat = 0.0;
    double gY_hat = 0.0;
    double gTheta_hat = 0.0;
    double leftU = 0.0;
    double rightU = 0.0;
    double ticks_Left = 0;
    double ticks_Right = 0;
    uint8_t robotMovement = moveStop;

    //prehandshake
    double ticks_Left_preHandshake      = 0;
    double ticks_Right_preHandshake     = 0;
    double total_ticks_r_preHandshake   = 0;
    double total_ticks_l_preHandshake   = 0;

    //new controllerApi
    double turning=1;
    double setpointX = 0;
    double setpointY = 0;
    double newCommand = 1;
    double xprev = 0;
    double yprev = 0;
    double thetaprev = 0;
    double distanceDriven =0;
    double waitingCommand =0;
    double ddInitX = 0;
    double ddInitY = 0;
    gTheta_hat = thetaprev;
    double thetaIntegralError = 0;
    double thetaError = 0;

    //IMU calibration
    bool calibration = 1;
    TickType_t ticks_since_startup = xTaskGetTickCount();
    double offsetGyroX = 0;
    double offsetGyroY = 0;
    double offsetGyroZ = 0;
    double gyroAngleZ =0;
    double delta_theta_gyro =0;
    double sample_diff_gyro_z =0;
    double gyro_z_prev=0;


    //init test parameters
    int counter = 0;
    int uL = 0;
    int uR = 0;
    float time_since_startup = 0;

    // used to display variables on OLED screen
    char test[128];
    char test2[128];
    char test3[128];
    char test4[128];

    //Logging
    float log_loop_variable=1;
    int log_loop=0;
    int log_guard=0;

    while (true) {
        taskYIELD();
        TickType_t ticks_since_startup_prev = ticks_since_startup;
        ticks_since_startup = xTaskGetTickCount();
        float delta_t = (ticks_since_startup - ticks_since_startup_prev)*1.0 /
↪   configTICK_RATE_HZ;
        time_since_startup = time_since_startup+delta_t;
        // double X_hat = gX_hat;
        // double Y_hat = gY_hat;
        // double Theta_hat = gTheta_hat;

        //Read IMU data
        IMU_reading_t gyro;
        // IMU_reading_t accel;
        IMU_read();
        gyro = IMU_getGyro();
        double gyro_x = gyro.x-offsetGyroX;
        double gyro_y = gyro.y-offsetGyroY;
        double gyro_z = gyro.z-offsetGyroZ;

        if (calibration){
            vTaskDelay(400);
            IMU_read();
            gyro = IMU_getGyro();
            offsetGyroX = gyro.x;
            offsetGyroY = gyro.y;
            offsetGyroZ = gyro.z;
            calibration = COMPLETE;
        }

        delta_theta_gyro = gyro_z*delta_t;
        if(!calibration && IMU_new_data() && (gyro_z>GYRO_MIN || gyro_z<-GYRO_MIN )){
            gyroAngleZ=gyroAngleZ+delta_theta_gyro;

        }

        gyro_z_prev =gyro_z;
        //double accel_x = accel.x;
        // accel = IMU_getAccel();
        // double accel_y = accel.y;
        // double accel_z = accel.z;
```

```c

        #if (DEBUG) //Printed to COM port via USB
        {
            //TICKS
            //printf("\r\nTicks Left: %f Ticks Right: %f\n\r",(float)ticks_Left_preHandshake,(float)ticks_Right_preHandshake);
            //printf("\r\n total ticks R: %f \n\r",total_ticks_r_preHandshake);
            //printf("\r\n total ticks L:  %f \n\r",total_ticks_l_preHandshake);

            //INPUT
            //printf("\r\nuL: %f uR: %f\n\r",(float)leftU,(float)rightU);

            //SETPOINT
            // printf("\r\n Setpoint: x %f y %f \n\r",(float)Setpoint.x, (float)Setpoint.y);
            // printf("\r\n New Setpoint Command %d \n\r", new_setpoint_command);

            //Distance Driven between setpoint commands
            //printf("\r\n Distance driven : %f \n\r",(float)distanceDriven);
            // printf("\r\n waitingcommand : %f \n\r",(float)waitingCommand);
            // printf("\r\n turning: %f \n\r",(float)turning);

            //GLOBAL STATES
            //double radToDeg = 180/3.14;
            //printf("\r\n %f %f %f\n\r",gX_hat,(float)gY_hat,(float)radToDeg*gTheta_hat);
            //printf("\r\n gXhat: %f gYhat: %f gTheta: %f\n\r",gX_hat,(float)gY_hat,(float)radToDeg*gTheta_hat);

            //GYRO VALUES
            //printf("\r\n gyroX: %f gyroY: %f gyroZ: %f\n\r",(float)gyro_x,(float)gyro_y,(float)gyro_z);
            //printf("\r\n test angle: %f\n\r",(float)gyroAngleZ);
            //printf("\r\n%f %f %f\n\r",(float)gyro_x,(float)gyro_y,(float)gyro_z);
            //printf("\r\n%f %f %f\n\r",(float)accel_x,(float)accel_y,(float)accel_z);
            //printf("\r\n accelX: %f accelY: %f accelZ: %f\n\r",(float)accel_x,(float)accel_y,(float)accel_z);

            //scope random value
            //printf("\r\n Random value : %f \n\r",(float)time_since_startup);
        }
        #endif

        if(!gHandshook){
            counter +=1;
            encoderTicks Ticks_preHandshake = encoder_get_ticks_since_last_time();
            ticks_Left_preHandshake = -(double)Ticks_preHandshake.left;
            ticks_Right_preHandshake = -(double)Ticks_preHandshake.right;
            total_ticks_r_preHandshake = total_ticks_r_preHandshake+ticks_Right_preHandshake;
            total_ticks_l_preHandshake = total_ticks_l_preHandshake+ticks_Left_preHandshake;

            //
            if (newCommand){
            turning = 1;
            setpointX = xWaypoint;
            setpointY = yWaypoint;
            waitingCommand = 0;
            newCommand = 0;
            ddInitX = gX_hat;
            ddInitY = gY_hat;
            thetaIntegralError=0;
            }
            //vTaskDelay(100);

        //  Displaying variables on OLED screen
        // Keep in mind that this action may slow down the system affecting the time
↪   constant of the system.
        // Use it mainly for debugging.
        // sprintf(test, "X: %i Y: %i", (int)gX_hat, (int)gY_hat);
        // display_text_on_line(1, test);
        // sprintf(test2, "Left: %f", (float)total_ticks_l_preHandshake);
        // display_text_on_line(2, test2);
        // sprintf(test3, "Right: %f", (float)total_ticks_r_preHandshake);
        // display_text_on_line(3, test3);
        // sprintf(test4, "Theta: %f", (float)gTheta_hat);
        // display_text_on_line(4, test4);


            //MATLAB generated function

            controller_api(setpointX, setpointY,  newCommand,
                   &waitingCommand, ticks_Left_preHandshake, ticks_Right_preHandshake,
                   &distanceDriven, &turning, xprev,
                   yprev, thetaprev, ddInitX,
                   ddInitY, delta_theta_gyro,
                   &thetaIntegralError, delta_t,&thetaError,
                   &gX_hat, &gY_hat, &gTheta_hat,
                   &leftU, &rightU);




            //SQUARE test MACRO
            #if (TEST_TYPE==SQUARE)
            {
                if(time_since_startup>1&&time_since_startup<10){
                    if (xWaypoint !=xWaypoint1 || yWaypoint!=yWaypoint1)
                    newCommand=1;
                    xWaypoint=xWaypoint1;
                    yWaypoint=yWaypoint1;
                }

                if (time_since_startup>10 && time_since_startup<20){

                    if (xWaypoint !=xWaypoint2 || yWaypoint!=yWaypoint2){
                        newCommand=1;
                        xWaypoint=xWaypoint2;
                        yWaypoint=yWaypoint2;
                    }
                }

                if (time_since_startup>20 && time_since_startup<30){

                    if (xWaypoint !=xWaypoint3 || yWaypoint!=yWaypoint3){
                newCommand=1;
                xWaypoint=xWaypoint3;
                yWaypoint=yWaypoint3;
                    }
                }

                if (time_since_startup>30 && time_since_startup<40){
                    if (xWaypoint !=xWaypoint4 || yWaypoint!=yWaypoint4){
                        newCommand=1;
                        xWaypoint=xWaypoint4;
                        yWaypoint=yWaypoint4;
                    }

                }

                if (time_since_startup>50){
                        leftU  = 0;
                        rightU = 0;
                    }

            }
            #endif

        //Square test log MACRO
        #if (TEST_TYPE == SQUARE && LOG)
        {
        if(time_since_startup>50 && LOG && !log_guard){
                    log_guard=1;
                    int i;
                    for(i = 0; i<squareTestInterval; ++i){
                            if((float)fabs(xarrayPos[i])<10000 &&
↪   (float)fabs(xarrayPos[i])<10000){
                            printf("\r\n%i %i\n\r",(int)xarrayPos[i],(int)yarrayPos[i]);
                            vTaskDelay(10);
                            }
                    }
        }
        }
        #endif



        //LINE test LOG MACRO
        #if(TEST_TYPE==LINE && LOG)
        {
            if(time_since_startup>20 && !log_guard){
                log_guard = 1;
                int i;
                for(i = 0; i<lineTestInterval; ++i){
                        if((int)fabs(xarrayPos[i])<10000 &&
↪   (int)fabs(xarrayPos[i])<10000)
                        {printf("\r\n%i %i\n\r",(int)xarrayPos[i],(int)yarrayPos[i]);}
                        vTaskDelay(10);
            }

        }
        }
        #endif

        }

        //collision logic
        // if (checkForCollision() == true){
        //          motor_brake();
        //          robotMovement = moveStop;
        //          xQueueSend(scanStatusQ,&robotMovement,0);
        //          printf("\r\n Testing collision block\n\r");
        //      leftU  =  0;
        //      rightU = 0;
        // }

        // temp values of global states
        xprev     = gX_hat;
        yprev     = gY_hat;
        thetaprev = gTheta_hat;


        // LOG MACRO logs position every 0.4 seconds
        #if (LOG)
        {
            log_loop_variable = log_loop_variable + delta_t;
            if (log_loop_variable>0.4 && log_loop<150){
                log_loop +=1;
                xarrayPos[log_loop-1] = (int)xprev;
                yarrayPos[log_loop-1] = (int)yprev;
                log_loop_variable=log_loop_variable-0.4;
            }

        }
        #endif

        uL = (int)leftU;
        uR = (int)rightU;

        vMotorMovementSwitch(uL,uR);
        taskYIELD();


        if (gHandshook) {
            printf("\r\n Testing handshake block\n\r");
            encoderTicks Ticks = encoder_get_ticks_since_last_time();
            ticks_Left = -(double)Ticks.left;
            ticks_Right = -(double)Ticks.right;
            if (xQueueReceive(poseControllerQ, &Setpoint, 0) == pdTRUE) {
            newCommand=true;
            }

            if (newCommand){
                turning = 1;
                setpointX = Setpoint.x*10;
                setpointY = Setpoint.y*10;
                waitingCommand = 0;
                newCommand = 0;
                ddInitX = gX_hat;
                ddInitY = gY_hat;
            }

          controller_api(setpointX, setpointY,  newCommand,
                   &waitingCommand, ticks_Left, ticks_Right,
                   &distanceDriven, &turning, xprev,
                   yprev, thetaprev, ddInitX,
                   ddInitY, delta_theta_gyro,
                   &thetaIntegralError, delta_t,&thetaError,
                   &gX_hat, &gY_hat, &gTheta_hat,
                   &leftU, &rightU);

        // temp values of global states
            xprev     = gX_hat;
            yprev     = gY_hat;
            thetaprev = gTheta_hat;

```

```c
            //vTaskDelay(100);

            // Cast to int before sending to motor
            int uR = (int)rightU;
            int uL = (int)leftU;


                        xSemaphoreTake(xPoseMutex, 15);
            set_position_estimate_heading(gTheta_hat);
            set_position_estimate_x(gX_hat/1000); // convert from mm to m
            set_position_estimate_y(gY_hat/1000); // convert from mm to m
            xSemaphoreGive(xPoseMutex);

            if (checkForCollision() == true){
                motor_brake();
                robotMovement = moveStop;
                xQueueSend(scanStatusQ,&robotMovement,0);
                printf("\r\n Testing collision block\n\r");
            }
            else{
                robotMovement = moveForward; // Lowest value for a movement. To stop scanning
                xQueueSend(scanStatusQ,&robotMovement,0);
                vMotorMovementSwitch(uL,uR);
                taskYIELD();
                }
            }
        }
    }
}
```