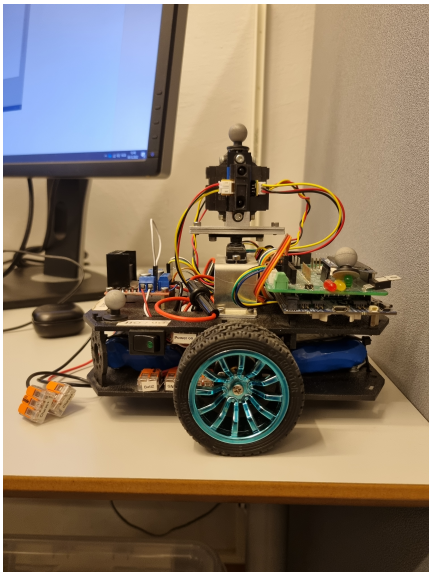
NTNU
Kunnskap for en bedre verden

Department of Cybernetics

TTK4550 - Engineering Cybernetics, Specialization Project

Set Point Control of Robot- and Server System using MATLAB generated C Code

Author:
Magnus Isdal Kolbeinsen



Supervisor:
Tor Onshus

Date 19.12.2022

Table of Contents

List of Figures

1 Problem Description

This project is a specialization project of the 15 credit course TTK4550 as part of the 5 year Cybernetics and Robotics program at Norwegian University of Science and Technology (NTNU) Gløshaugen. This course project is supervised by Tor Onshus.

The objective of this project is to make MATLAB code for controlling the robot and then generating C code. The motivation to use MATLAB is that it makes it easier to test out algorithms for control and mapping. Currently, a lot of the code for controlling the robot is written explicitly in C code and it is wanted to move this code to MATLAB.

This project is a continuation of a previous project by Maria Gilje [2]. A few issues regarding MATLAB implementation of controller was identified in Gilje's project:

- Robot could only do one command
- Robot would rotate around its own axis after doing a command
- Robot would diverge from a straight line

Ideally, these problems are solved by the end of the project. Furthermore, the project can be divided into the following sub tasks:

1. Learn how the robot- and server system works and set up the system based on previous work.
2. Perform initial testing on important hardware components. Calibrate and repair damages.
3. Develop MATLAB code for controlling the robot.
4. Generate C code from MATLAB and implement it into the robot system.
5. Perform integration testing of the implemented code.

2 Summary and Conclusion

The system is a two wheeled vehicle that interfaces its environment by using IR-sensors, wheel encoders and IMU. It receives set point commands from a Java Server in a 2D plane (x,y). Previously, software has been developed mostly in C, but also in MATLAB. In this project, an attempt to create a satisfactory control system has been done by refactoring old C code vApiTask() and making MATLAB generated C code. This report contains necessary information about the robot such as how to set up the robot, how to set up the server, and brief overview of hardware, firmware and software. Initial testing revealed some valuable aspects about the system and damaged hardware components. Faulty components has been fixed. MATLAB code has been developed for controlling the robot, generated into C code, and integrated into the system. Moreover, integration tests has been performed. The report documents the initial testing, code design and integration testing. The code design is based around how MATLAB generated C code interfaces the robot system. The implemented MATLAB generated C code is called api() which gathers all necessary input for doing set point control of the robot system. The integration tests shows that the system is now able to receive multiple commands and do multiple commands and does not rotate uncontrollably after doing a command. It still diverges from a straight lines, but not by very much (about ±5cm) of a line 100 cm. Adding a PID-regulator for moving straight would be needed to prevent this drift entirely. Moreover, the control system does estimation only based of encoder output. Including IMU measurements in estimation, especially when rotating, could increase the accuracy of the estimate.

3 Getting to Know the Problem

In order to solve the assignment, it was important to learn how the system works. The robot is a two wheel vehicle that contains a number of sensors. This section will contain a brief overview over how the robot works, how to setup the robot and the Java server up and some known limitations with the system. The robot was built by Eivind Jølsgård. More details about robot system can be found at Jølsgård [3].

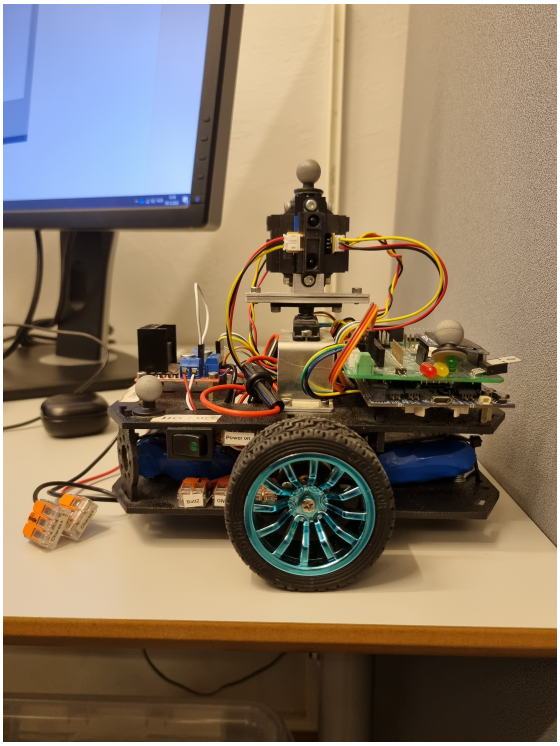


Figure 1: Picture of Robot NRF3.

3.1 Robot Description

Hardware

The main components of interest are:

- 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 NRF5 hardware shield
- 2 x 2600 Ah batteries
- 1 x OLED display

An infrared (IR) sensor is a sensor that detects and measures infrared radiation from the environment. The robot contains four IR sensors. They are active IR sensors meaning that they both emit and detect infrared radiation, and contain two parts, a light emitting diode (LED) and a receiver. A detected object would reflect light from the LED to the receiver. The IR sensors are mounted together in a tower, and are mainly used for collision avoidance and obstacle detection. They are also used by the server to map the area around it. The datasheets of the IR sensors can be found at [10].

A servo motor is a actuator that allows for accurate control of linear or angular motion. The robot has one servo motor. It is a rotary motor is used to rotate the IR sensor tower. The configuration of the IR sensor tower does not allow for a rotation larger than 90°. The datasheet of the servo motor can be found at [11].

A DC motor is a rotary electrical motors that converts direct current into mechanical energy. The robot consists of two DC motors that are attached to wheels and are used to rotate and move the robot. The motors also contain encoders to measure angle displacement and angular velocity. The product information about wheel system with DC motors, encoders and wheels can be found at [1].

An H bridge motor driver is a specialized circuit for enabling bidirectional motor driving. They are used to regulate the voltage of the DC motors to allow for speed control and directional control. The H brigde motor driver on the robot is a L298 dual H-bridge meaning that it can control up to two DC motors. Information about the motor driver can be found at [4].

An inertial measurement unit (IMU) is a device that typically consists of a gyroscope and a accelerometer. The gyroscope angular velocities and the accelerometer measures specific force. The robot has one IMU that is located under the frame of the robot. Information about the IMU can be found at [9].

The movement detection balls are not used by the robot itself, but they make it possible for other robots to detect it more reliably.

The nRF5840 Development Kit (DK) is a single board development kit by Nordic Semiconductor for Bluetooth Low Energy (BLE), Bluetooth mesh, thread and Zigbee. It is a programmable chip that allows for control of digital pins and flashing of C code. More details about can be found at [7].

The custom control chip is made by a former student Eivind Jølsgård. It is a hardware shield that is stacked on top of the nRF52840 DK and acts as a gateway to control the DC motors and servo motor as well as reading the IR sensor and wheel encoder output. For more details about the shield read the report of Jølsgård [3].

OLED display is a display mounted on top of the nRF5 hardware shield.

Firmware

nRF52840 chip contains firmware that makes it possible to run C code on it. Functions has has also been implemented to make it easier to do control inputs from the NRF52840 chip. The main components of the firmware Jlink and SoftDevice. If the NRF52840 chip ever becomes corrupt you would need to bootload this firmware on to the mass storage device again. More information about the firmware can be found at [5].

Software

Two important standard libraries are nRF5 SDK and freeRTOS.

nRF5 software development kit (SDK) is a standard library from Nordic Semiconductor to use nRF5 microcontrollers. It provides a development environment for the nRF51 and nRF52 series. The version of SDK used in this project is 15.0.0. More information about the SDK can be found here [8].

freeRTOS is a real-time operating system libary allowing task management and scheduling for microcontrollers and microprocessors. It is distributed freely under MIT open source licence. More information about freeRTOS can be found here [6].

Software has been developed over the years by students from previous years. It runs on as a task based real time system. The main parts of the implemented software are:

- Drivers
- Java Server
- BLE communication (Arq Task)
- Api
- Api Task
- Main Communication Task
- Sensor Tower Task

The driver contains functions for controlling servo motor, DC motors and LED-display, and reading wheel encoders and IR sensors.

The Java Server code is the contains the necessary software to set up the mapping server of the robots. The Java server connects to the robots and receives their position and IR sensor output and then makes a map of the environment.

The BLE communication contains utility functions regarding bluetooth low energy communications. BLE is used to communicate with the Java Server. Arq task is inside this folder.

Api contains MATLAB generated C code. Api is the part of the code that regulates the position of the robot based on the incoming command from the Server. It is a large function that returns the output to the left and the right DC motors.

Api Task is the task where control of the robot lies. It calls the api function, reads sensors, reads commands from server and adds input to motors.

Main communication tasks is the task for maintaining the connection with the Java. It also forwards the set command from the server to vApiTask().

Sensor tower task controls the servo motor and reads the output from the IR sensors, and forwards pose estimate and IR sensor data to the Java server.

A general visual illustration of how the tasks operates can be seen in Figure 8.

3.2 How to set up the Robot

This section is a quick guide to set up the robot.

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

The project code contains nFR5 Software Development Kit (SDK) version 15.0.0. This should be included in the project code zip file, but if not it can be found here:
https://www.nordicsemi.com/Products/Development-software/nRF5-SDK/Download?lang=en#infotabs
You would also have to download SoftDevices. It is in the nRF5 SDK folder\examples that the robot-application folder is and should be.

Development kits

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 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. Navigate to the robot-application folder with a terminal
2. Type *make* to build the code. Make sure that the path in the makefile is correct.
3. Connect with USB to the short side of the NRF and type *make flash* to flash.

3.3 How to set up the Server

This is a quick guide in how to set up the Java server. A more indepth guide is provided in the Java Server folder.

Programs and packages

1. Install NRF connect https://www.nordicsemi.com/Products/Development-tools/nrf-connect-for-desktop.
2. Install jdk from the Java server folder (Careful not to do any updates, it needs to be version 1.8).
3. Install jre from the Java server folder.
4. Install Apache Netbeans from the Java Server folder 12.1.

Flashing the Dongle

1. Open NRF Connect.
2. Open Programmer.
3. Connect the NRF51 Dongle to USB.
4. Select device from the drop-down menu.
5. Drag and drop the peripheral hex file from the nRF51 Dongles Folder to the device.
6. Erase and Write.

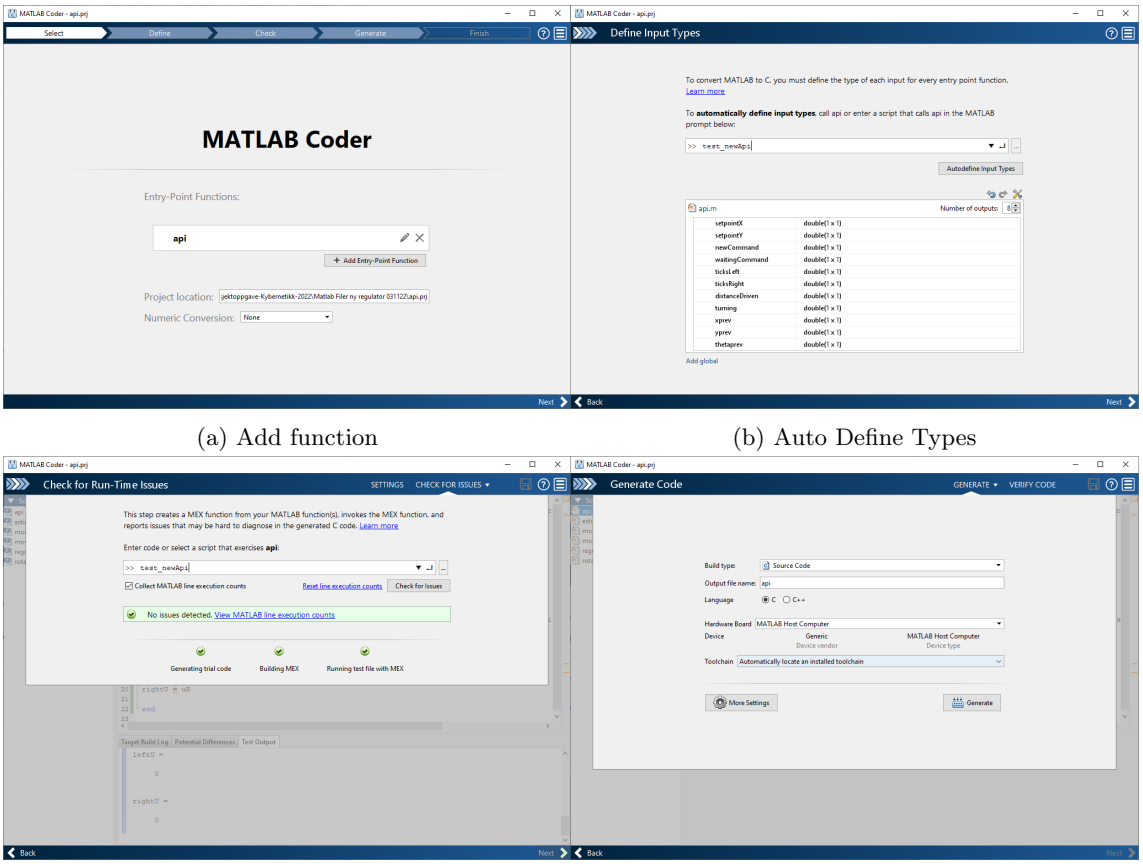
Setting up Server

1. Open Dypbukt SSNAR - cart folder in Netbeans.
2. Relink all .jar libraries.
3. Connect Dongle to USB.
4. Clean and build project.
5. Click Run.
6. Connection is established with the Dongle if the light is green.
7. Click real world simulation, if your robot should show up if correct code is flashed.

3.4 Generating C code from MATLAB

In this project the MATLAB coder function was used to generate C code. This functionality can be found in the Apps section in MATLAB. The procedure of generating code is depicted in Figure 2, and is quite simple:

1. Add function you want to generate. Click Next.
2. Add a function that calls the function. Click Autodefine Input Types. Click Next.
3. Check for issues. If any issues, resolve them else Click Next.
4. Generate.



(a) Add function

(b) Auto Define Types

(c) Check for Issues

(d) Generate

Figure 2: Generating C Code from MATLAB

3.5 How the Robot Moves

The robot moves by giving input to the wheel motors. Positive input on both DC motors makes the robot go forward as shown in Figure 3.

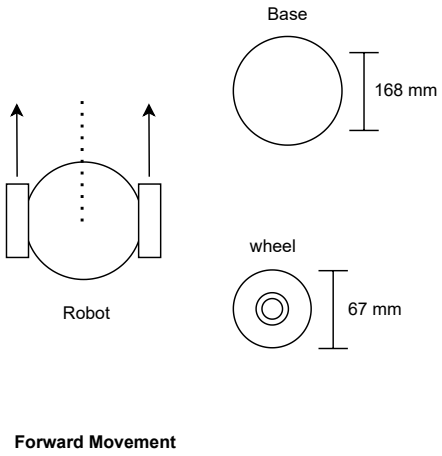


Figure 3: Forward movement, base and wheel diameter. Robot has two wheels and a base. Arrows indicate the wheel force.

To turn the robot you could move either wheel separately or simultaneously in opposite directions. The latter is the fastest and does not change the position of the base in any significant way. This is the method used to rotate the robot. Rotary movement can be seen in Figure 4.

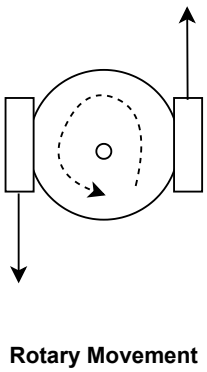


Figure 4: Rotary movement. It shows two wheels and a base. Arrows indicate wheel force.

3.6 Limitations and Known Issues

NRF 3 has a known hardware bug. It will not start solely based on the battery packs. It needs a kick-start from USB to be able to start. However, it can maintain energy with the battery packs. Meaning that you will have to plug it into USB and then turn on the power switch if u want to test the robot. Additionally, only one of the batteries work and there are no display of battery level.

MATLAB coder is a limitation within itself, as not all functions from MATLAB can be generated as C code. This will require workarounds.

The robot is a task based real time system, which means there will be timing issues and delays that might affect performance.

4 Initial Testing and Repairs

After getting the robot up and running along with previously implemented software. A few inital tests was needed to see if every component worked as intended. This was to see if this was causing any unforeseen problems, and the testing revealed a few issues that had to be fixed.

4.1 IR Sensor Testing

The IR sensor tests consisted of reading the output data from the IR sensor tower while no motors running. This testing revealed a problem where two out of four sensors were not properly connected. To fix the problem, it was needed to solder them back together.

4.2 Wheel DC Motor Testing

The DC motor testing consisted of testing the input to the motors. The goal of these tests was to find the threshold of input to make the motors move and reveal some initial limitations with the motors.

First test was to determine how much input was needed to make the robot move. The test was first done with no friction from the ground and it showed that the motors would move with about 6 duty input to the *motorswitch-function*. Furthermore, while testing on the ground it needed 9 duty input.

The second test consisted of running both motors with the same input (12) to *motorswitch-function* for 15 cycles with about 200ms delay in vApiTask. This revealed that giving the same input to the motors would not make the robot move straight. The input to the *motorswitch-function* was integers, which made it impossible to tune perfectly.

The third test consisted of running the motors in opposite directions to see if the robot would rotate in a fixed position, and it did.

4.3 Wheel Encoder Testing

The wheel encoder testing consisted of five tests. The first three tests were to read the encoder output on while moving the wheels around manually; each wheel individually, moving the robot forward and rotating the robot. The second was to read the encoder output while running the motors in the same direction with same input. The third test was to read the encoder output while doing the rotary movement in Figure 4. During testing, vApiTask() was alterd accordingly.

The first test revealed that the encoders would not start to read ticks instantly. It had $\pm 5^\circ$ wheel play each wheel. It is also revealed that the left encoder had a loose connection meaning it would sometimes not read ticks. To solve this issue it was needed to fix the cable connecting the encoder to the PCB. Additionally, it also revealed that the encoder was wrong on according to the hardware PCB board. In other words, the left encoder was connected to ENCR and vice versa. Even though it seemed to have been handled in the software previously, it was decided to fix it so that it matched the PCB layout. In Figure 5c and Figure 5d the tick output from the encoders are shown from one of the manual revolution tests. The amount of ticks for one rotation seemed quite consistent around 300 ticks ± 5 for a rotation.

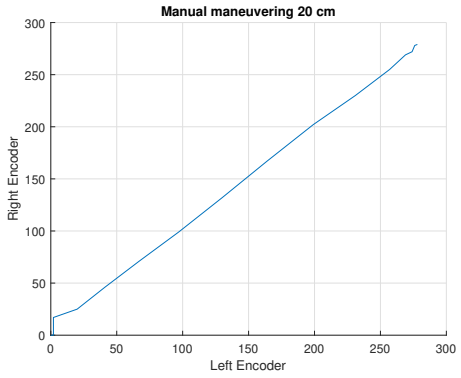
The second test consisted of moving the robot in a linear motion about 20 cm and reading the ticks accumulated on each wheel. This test is shown in Figure 5a. From this test you can see that while moving straight, the ticks on each encoder are very similar indicating consistency between the encoders.

The third test consisted of rotating the robot 90° and reading the output from the encoders. This test is shown in Figure 5b. The robot starts at (0,0) ticks and ends at (-153,187). This test was

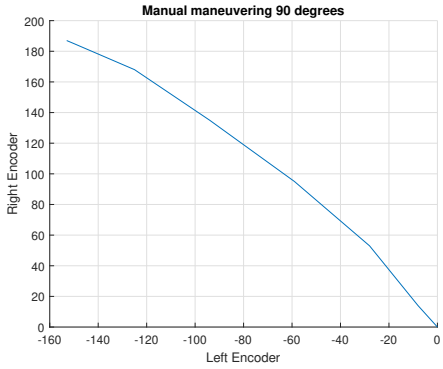
taken several times and it was not very consistent. In the test depicted the difference between the ticks was 34 while rotating 90°. This indicates irregularities between rotation and encoder ticks.

The fourth test consisted of running both motors at the same time with same input for 15 cycles. The test can be seen in Figure 5f. The encoders started at (0,0) and ended (953,1003). During the test it was also seen that the robot diverged from moving in a straight line. So even though there is a 50 tick difference at the end, it does not mean that using the encoders for straight line estimation is bad. However, it further concluded that the robot needs some sort of tuning on the duty input on each motor individually.

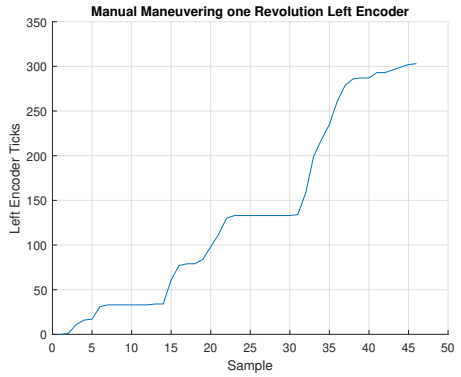
The fifth test revealed some irregularities with encoder ticks while rotating. The test can be seen in Figure 5e. The encoders started on (0,0) and ended on (-823,690) which is a 133 difference in amount of ticks after only 15 cycles of rotating. The robot rotated almost a full rotation during the 15 cycles.



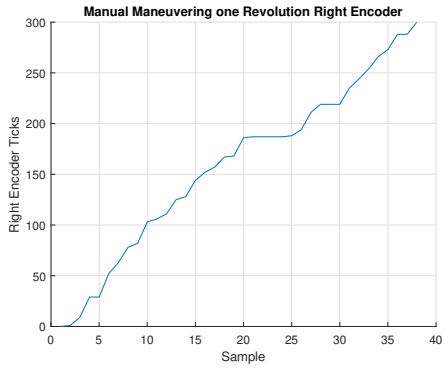
(a) Test for manually moving the robot 20 cm. Encoder ticks of each encoder shown as points (Left,Right).



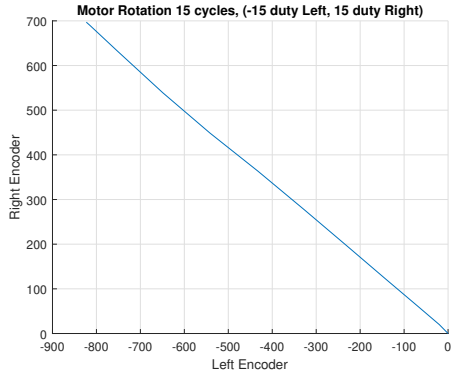
(b) Test for manually rotating the wheels. Encoder ticks of each encoder shown as points (Left,Right).



(c) Test for manually the left wheel. Encoder ticks of each encoder shown as points (Sample,Right).



(d) Test for manually the right wheel. Encoder ticks of each encoder shown as points (Sample,Right).



(e) Test for rotating with motors. 15 duty on right motor and -15 duty on left motor. Encoder ticks of each encoder shown as points (Left,Right).

(f) Test for moving straight with motors. 15 duty input on both motors. Encoder ticks of each encoder shown as points (Left,Right).

Figure 5: Initial encoder tests.

4.4 IMU Testing

The IMU testing consisted of reading the IMU values while running the robot. The IMU seemed to work properly while testing. Gyro output values can be seen in Figure 6 and the accelerometer output can be seen in Figure 7. These are standstill plots to see how accurate the estimates are, and they seem to be quite accurate.

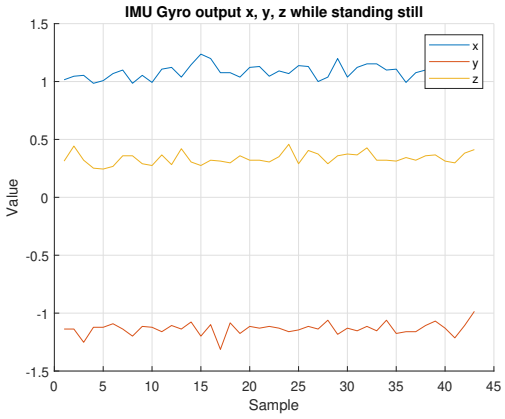


Figure 6: Gyro output x, y and z.

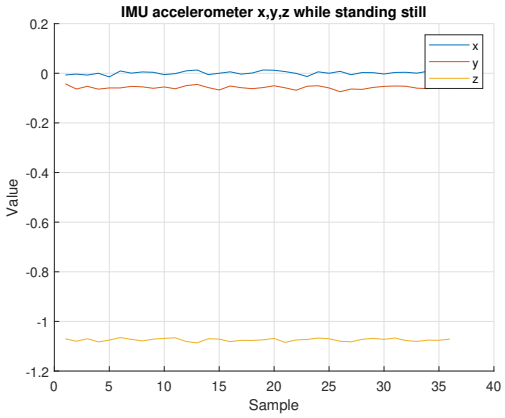


Figure 7: Accelerometer output x, y and z.

5 Controller Design and Modules

This section contains an overview of how the MATLAB generated C code interfaces the system, considerations while designing the code and an overview of the implemented code api().

5.1 Controller Api Design

The api function gathers all input regarding the pose then returning pose estimate and input to the motors. In Figure 8 you can see a general visualization of how the api function interfaces the robot system. It is only called by vApiTask(), but is the main function for controlling the robot and estimating pose.

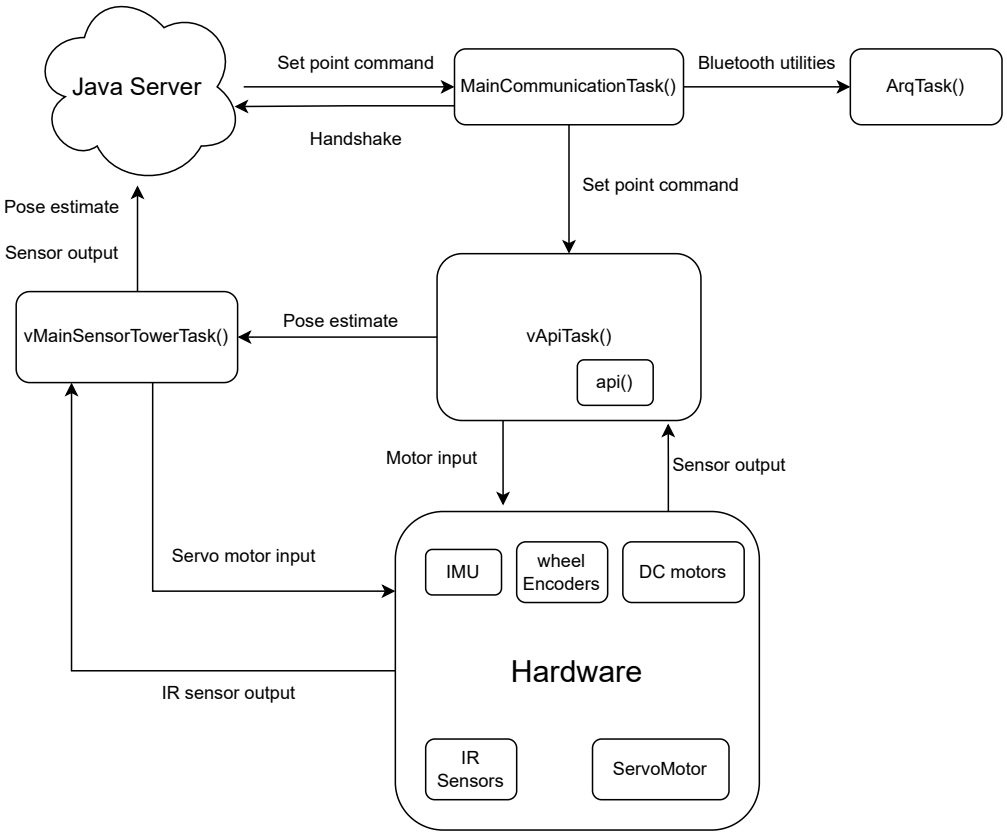


Figure 8: Dependencies in the Robot system. Illustration of how hardware is connected to tasks and server.

Considerations

Sequence of action

There are different ways to design a controller for end position. One could regulate angle towards the target and move forward simultaneously or one could decouple rotary movement and forward movement, and do the movements separately. The latter was chosen in this project as it was deemed easier and potentially faster in certain configurations. This is also the method that was been previously implemented in C code.

How to interface vApiTask

Since the api function is written in MATLAB, it needed to be considered how to interface the vApiTask which was written explicitly in C. Furthermore, the less variables that needs to be sent as reference (and in general) to the api function, the better as it can become quite big.

Real time system

The system is limited by how fast it can read the sensors and the delays. This is because it needs to yield or delay tasks to maintain bluetooth connection with the server. Therefore this must be considered when designing the control system.

MATLAB coder

MATLAB coder generated code will not always work directly. Sometimes you would need to customize basic functions in order to generate code from MATLAB.

Measurements to use for estimation

Measurement to be used for estimation will be important. Initial testing revealed that the encoders are quite consistent while moving linearly, but showed some irregularities while rotating. IMU are also quite consistent, but may cause drift while integrating. In this report the encoders were used as the measurement for developing the system.

5.2 Control Modules Overview

An overview of the code can be seen in Figure 9.

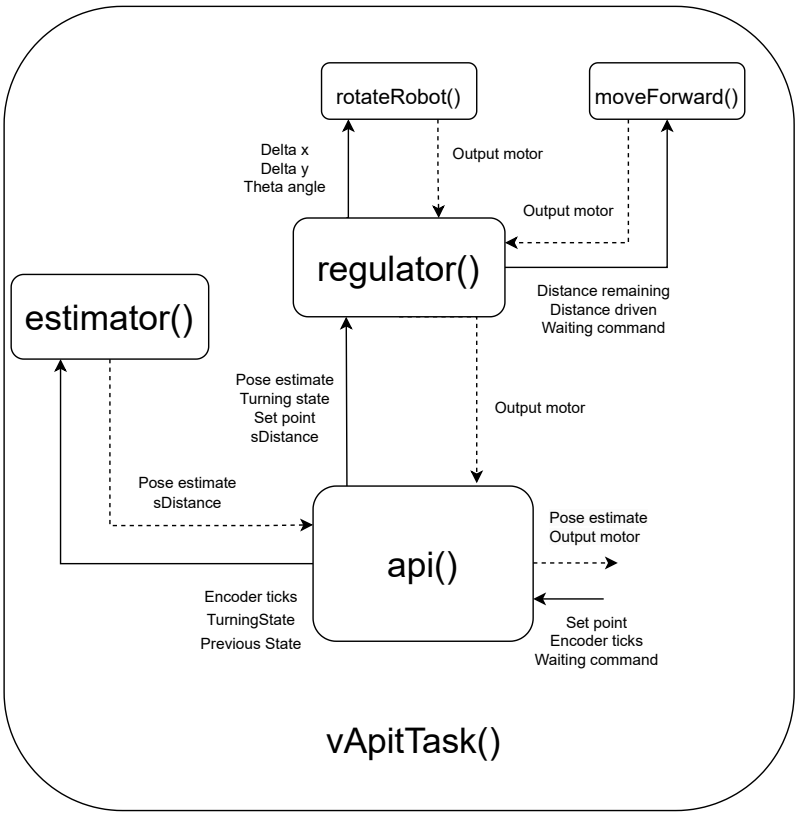


Figure 9: Overview of connections between MATLAB functions and vApitask().

5.3 vApiTask()

This function was altered based on the test that was being done. Different tests required different variations of vApiTask().

Some general changes that has been done to vApiTask() are:

- Delays between reading sensor output and setting motor input have been removed and replaced with yields.
- Debug functionality has also been added to this function.

5.4 Api()

The main part of this project is based around the implemented api code. It is implemented in MATLAB and then generated into C code by using MATLAB coder as shown in Figure 2. The api function is called in vApiTask() where it is used to calculate pose and input to both DC motors. It contains two main modules; the estimator and the regulator. It has some inspiration from previous code, but most variables has been changed. The ManTask function has been removed as it was deemed redundant.

The function looks like this:

```
function [gX_hat,gY_hat,gTheta_hat,distanceDriven,...
    leftU,rightU,turning,waitingCommand] = ...
    api(setpointX,setpointY,newCommand,waitingCommand,...
    ticksLeft,ticksRight,distanceDriven,turning,xprev,yprev,thetaprev)
%changes global states and returns input to the motors

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

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

[uL, uR,distanceDriven,turning,waitingCommand] = regulator(xHat,setpoint,turning,...
    sDistance,distanceDriven,waitingCommand);

gX_hat      = xHat(1);
gY_hat      = xHat(2);
gTheta_hat  = xHat(3);
leftU       = uL;
rightU      = uR;
end
```

The function takes in the set point command, the previous position and the encoder output. Moreover, it calls the estimator for a position estimate then the regulator for motor output.

Estimator

The estimator as of now only uses encoders to estimate position. This limits the accuracy of how the position estimate, but can give a satisfactory estimate . Position is based on its previous position and the calculated θ angle between the target position and the current position. The function can be seen below:

```
function [xHat, sDistance] = estimator(Encoder,prev,turning)
% Calculation of current position and orientation [x_hat, y_hat, theta_hat],
% and distance moved during this sample

%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)*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)+sTheta;
    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
```

The estimator is divided into two parts based on the movement of the robot; forward movement and rotation.

In the forward movement estimator, the position estimate and θ_{robot} estimate are updated. Position is updated by adding distance increment to the position estimate. The calculation is shown in (1) and (2).

$$x_{k+1} = x_k + sDistance \cdot cos(\theta) \tag{1}$$
$$y_{k+1} = y_k + sDistance \cdot sin(\theta) \tag{2}$$

where sDistance is the distance of one sample calculated based on average ticks from both encoders while driving forward.

In the rotary movement estimator, only the θ_{robot} angle is updated. An assumption has been made that the robot position is $\approx$ stationary during rotation. θ is updated by adding a sTheta-increment to the previous θ_{robot}.

$$\theta_{k+1} = \theta_k + sTheta \tag{3}$$

where sTheta is based on the difference between the ticks from the right encoder and the left encoder divided by two and multiplied by sign of the difference (4). ssa is also used on θ_{k+1}. This calculation needs an initial θ_0, meaning that you would need to the robot to be placed with a $\theta_0 = 0$ for this calculation to work.

$$sTheta = 0.5 \cdot sign(sTicksRight)abs(sTicksRight - sTicksLeft) \tag{4}$$

where sTicks are the encoder ticks read during this cycle, and sign() takes the sign of sTicksRight because counter clockwise rotation has been defined as position direction of rotation.

An assumption made for the θ calculation is that by rotating the robot the amount of distance displaced is only around the circle between the wheels. In other words the difference between the ticks of the motors are the distance it travels around the circle intersecting the wheels. During the initial tests it was observed that the robot would not change its position while rotating like this. How it rotates is depicted in Figure 10.

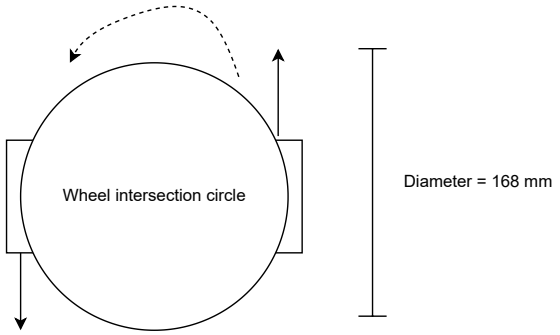


Figure 10: Rotation around the intersection circle.

Regulator

The regulator is divided into two parts; forward movement and rotary movement. It is a sequential regulator where it will check for a new command, rotate towards the target location and then move straight towards the location. The regulator code can be seen below:

```
function [uL,uR,distanceDriven,turning,waitingCommand] = regulator(xHat,setpoint,turning,...
    sDistance,distanceDriven,waitingCommand)
% Returns the inputs to the Left and Right motors
% xHat = [x_hat, y_hat, theta_hat] ~ [mm,mm,rad]
% setpoint = [x_d, y_d] ~ [mm,mm]
% turning ~ is the robot in rotating mode?
% sDistance ~ distance moved this sample
% waitingCommand ~ the robot is in an idle state waiting for a new command

%xHAt
x           = xHat(1);
y           = xHat(2);
theta       = xHat(3);
theta_0     = theta;

%setpoint
x_d         = setpoint(1);
y_d         = setpoint(2);

%Errors
delta_x     = x_d - x;
delta_y     = y_d - y;

%Thresholds
angleThreshold  = deg2rad(5);
drThreshold     = 50;
ddThreshold     = 1200;

distanceRemaining    = sqrt(delta_x^2+delta_y^2);
distanceDriven       = distanceDriven+sDistance;

if turning
    u = 12;
    [uL, uR,turning] = rotateRobot(delta_x,delta_y,theta_0,angleThreshold,u);
    distanceDriven = 0;
else
    u = 10;
    [uL, uR] = moveForward(distanceRemaining,distanceDriven,...
        drThreshold,ddThreshold,u,waitingCommand);
    if (uL == 0) && (uR == 0)
        waitingCommand = 1;
    end
end
```

Rotary movement is the first action that happens when the robot receives a new command. A new command is received and the turning variable becomes active. rotateRobot() is called and the robot rotates. The code for rotateRobot() can be seen below:

```
function [uL,uR,turning] = rotateRobot(delta_x,delta_y,theta_0, angleThreshold, u)
%Rotates robot based on its current position and threshold for rotating
% delta_x   = x_d - x
% delta_y   = y_d - y
% theta_0   = theta_hat

% angle towards setpoint
theta_target        = atan2(delta_y,delta_x);
testThetaDegrees    = rad2deg(theta_target)

thetaError          = modulus(theta_target-theta_0+pi,2*pi)-pi; %%smallest signed angle

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
```

RotateRobot() compares the angle of the robot θ_{robot} with the angle towards the target θ_{target} and then decides what direction to rotate robot. θ_{target} is depicted in Figure 11 and the calculation can be seen in (5).

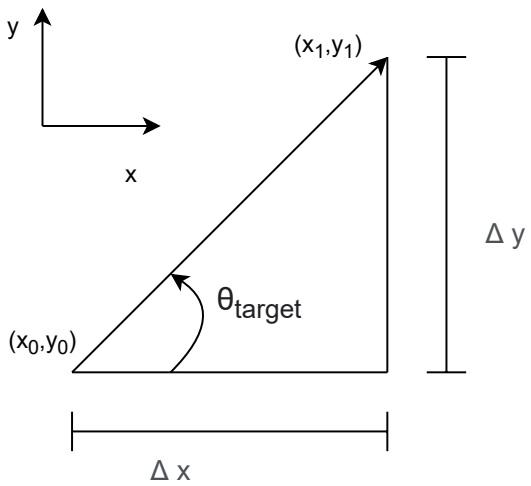


Figure 11: $\theta, \Delta x, \Delta y$ Initial position (x_0, y_0) and target position (x_1, y_1) with $\theta_{robot} = 0$.

$$\theta_{target} = ssa(atan2(\frac{\Delta y}{\Delta x})) \tag{5}$$

where atan2 is a 4-quadrant tangens function and ssa is a *smallest signed angle* function which maps the angle to the interval $[-\pi, \pi]$ and always chooses the smallest angle. This function was been inspired by Thor I. Fossen's implementation in his Marine Systems Simulator toolbox in MATLAB. A custom modulus function had to be implemented because of floating point differences between MATLAB and C.

After the robot has rotated towards the correct target angle, moveForward() is called and the input to the motors are changed. Two calculations that are tracked during control are distanceRemaining (6) and distanceDriven (7). DistanceRemaining calculates the distance towards the target location, and distanceDriven tracks how long the robot has driven since he received a new command. These are used in the moveForward function to decide when to stop moving.

$$distanceRemaining = \sqrt{\Delta x^2 + \Delta y^2} \tag{6}$$

$$distanceDriven = distanceDriven + sDistance \tag{7}$$

where $\Delta x = x_{target} - x_{robot}$, $\Delta y = y_{target} - y_{robot}$ and sDistance is the calculated distance for a sample based on encoder ticks.

moveForward() contains necessary logic for starting and stopping forward movement: The left DC motor was a bit stronger than the right. Adding +1 to the right motor gave a better result. The code for moveForward() is shown below:

```
function [uL,uR] = moveForward(distanceRemaining,distanceDriven,...
    thresholdDR,thresholdDD,u,waitingCommand)
%Moves the robot Forward if thresholds are met
%  distanceRemaining [mm]
%  distanceDriven [mm]
%  thresholdDR [mm]
%  thresholdDD [mm]

if distanceRemaining>thresholdDR && distanceDriven<thresholdDD && not(waitingCommand)
    uR = u+1;
    uL = u;
else
    uR = 0;
    uL = 0;
end
```

6 Integration Testing

After implementing the software into the robot system, different types of integration tests were performed; manual maneuvering, rotation test, straight line test and square test.

6.1 Manual Maneuvering

The manual maneuvering tests consist of 3 tests; 20 cm line test, 90° rotation and 180° rotation. Since these are manual tests there may be some discrepancies in the manuvering. This is especially the case while rotating since the wheels has to rotate by the same amount of ticks.

The 20 cm (200 mm) line test was done by manually moving the robot in a straight line 20 cm. The motivation behind the test was to see if the implemented estimator was able to give good estimates while moving straight. Figure 12a shows the estimated position and Figure 12b shows the θ angle of the robot. The test shows that it tracks the position very well for 20 cm. In this test it ended on (20cm, 0.5 cm) a perfect x estimate, and almost perfect y estimate. The θ estimate went from 0 to 3.5°.

The 90° and 180° rotation tests were done by manually rotating the wheels to rotate the robot to the corresponding ground truth value. Figure 12c and Figure 12d shows the θ angle of the robot in the 90 and 180 tests, respectively. The 90° test shows that the tracking of the angle is quite accurate. It went from 0 to 86 in 10 samples, only 4° off the target. The 180° test was more inaccurate where it ended on 167°, 13° off the target. This test took 35 samples to make and was tougher to maneuver. That is also why there are several bumps in the graph.

6.2 Rotation Test

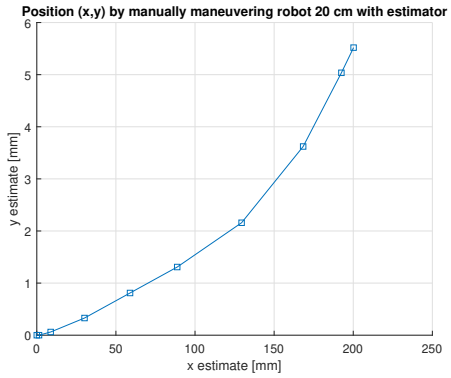
The rotation test was a test to see if the robot would be able to rotate 90 degrees and then move in a straight line. The robot was started in position $(0, 0)$ and commanded to move to (0,300). The initial θ_{robot} angle was set to 0, meaning it would have to rotate 90 degrees first in order to reach its target. Figure 12e and Figure 12f shows the position estimate and the *theta* angle of the robot, respectively. The position estimate shows that it ends in position $\approx$ (23,295). The plot also shows that the drift in x is quite linear which is an interesting takeaway from this test. It is also worth mentioning that the rotation part of the sequence is not part of the position estimation plot. This is because an assumption that the robot position is stationary during rotation. The θ estimate plot on the other hand has captured the whole sequence. It starts off by rotating peaks at 100° then adjusts itself by changing angular rotation and settles at 85 while slowly increasing. It was also observed to be a satisfactory end point with respect to ground truth indicating that the estimate was indeed good.

6.3 Straight Line Test

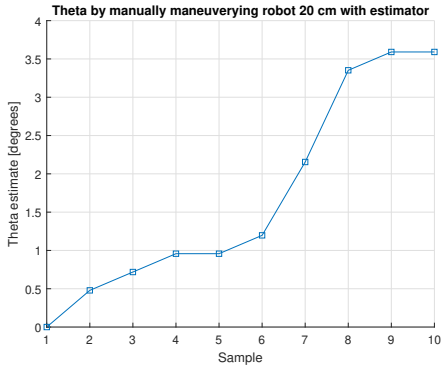
The straight line test was a test to see if the robot would be able to follow a straight line command. The robot was started in position $(0, 0)$ and commanded to move to $(1000, 0)$. The *theta* angle was set to 0, meaning it would only need to move forward to reach its target. The straight line test position estimate and θ estimate are depicted in Figure 13a and Figure 13b, respectively. The position estimate plot shows that the y moves within the [-50mm,50mm] range. In other words it changed directions during the test. This can also be seen in the θ estimate as it goes down to $-7°$ and then up to 27 towards the end. The ground truth observation was that it diverged slightly from a straight path.

6.4 Square Test

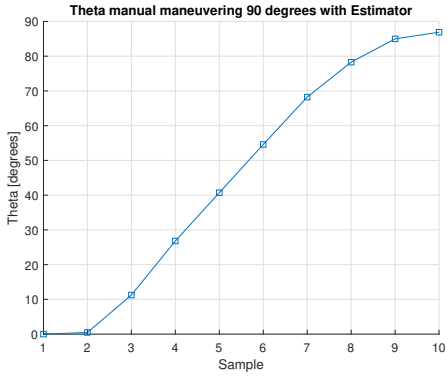
The square test was a test to see if the robot was able to maintain a somewhat consistent route and receiving multiple commands in sequence. The test waypoints were (300,0),(300,300),(0,300),(0,0) and initial position was (0,0). The square test 1 and 2 is shown in Figure 13c and Figure 13d, respectively. Square test 1 shows a path that resembles a square. It seems to always keep its position inside the square and the ending location is a bit off from the last way point. Square test 2 has a more satisfactory path it gets closer to the way points between commands. The main difference between square test 1 and 2, is that the commands were given at a slower rate in square test 2. This may be the reason for the last way point being a bit further away from its intended endpoint. The ground truth observation of the square tests were that the robot in both tests moved in a square like shape of what looks to be around 300 mm.



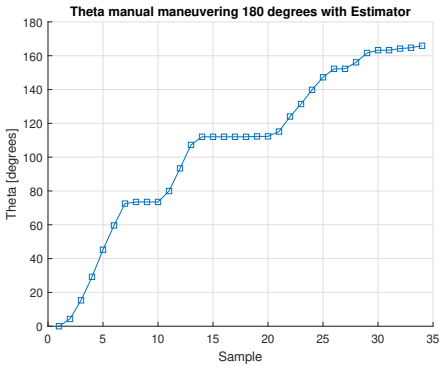
(a) Manually maneuvering 20 cm while estimating postion.



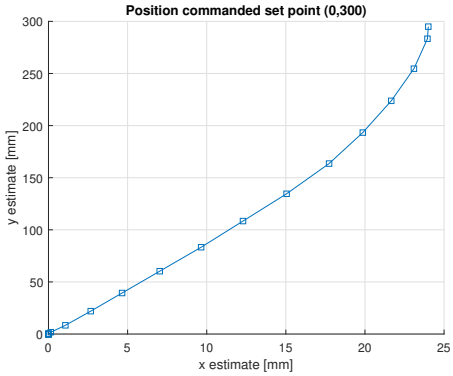
(b) Manually maneuvering 20 cm while estimating θ.



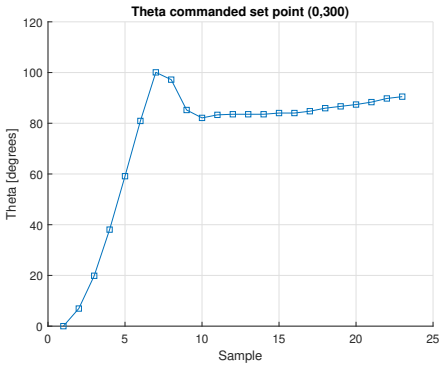
(c) Manually rotating the robot 90° while estimating θ.



(d) Manually rotating the robot 180° while estimating θ.

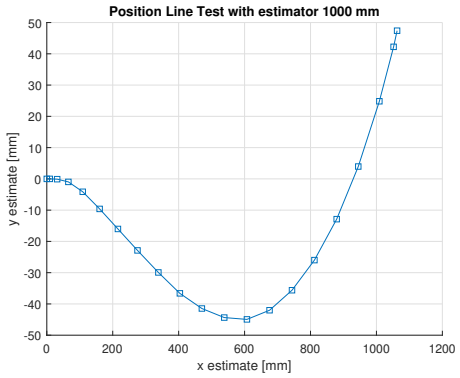


(e) Position estimate while commanded to position (0,300)mm.

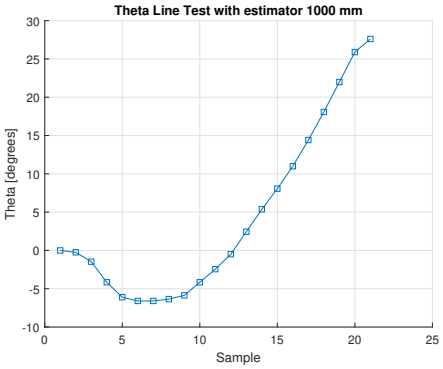


(f) θ estimate while while commanded to position (0,300)mm.

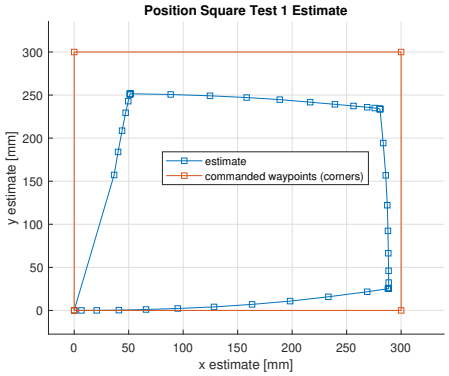
Figure 12: Manual maneuvering tests and commanded rotation test.



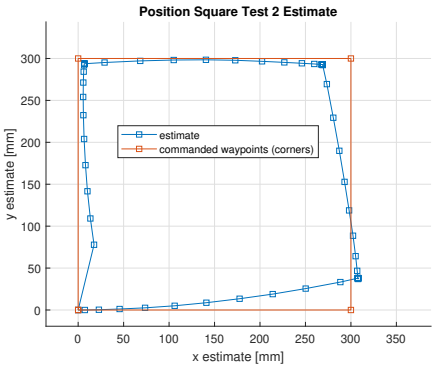
(a) Position estimate while commanded to position (1000,0)mm.



(b) θ estimate while commanded to position (1000,0)mm.



(c) Moving robot in a square. Test 1.



(d) Moving robot in a square. Test 2.

Figure 13: Line Test and Square Test.

7 Discussion

Setting up the Robot- and Server System

A lot of time of this project was spent trying to get the robot up and running. The robot itself came with some hardware flaws and there were problems with software and packages to be installed. Examples were the CMake path dependencies, the jdk versions for Java Server, the NRF51 Dongle and .jar files in Netbeans.

Initial Testing

Initial testing revealed some problems with the hardware. Some time was spent here trying to figure out how well all parts of the system worked. The tests revealed some weaknesses with the system. This was an important step in order to understand how the system could be built and also repair damaged components.

During initial testing, the DC motor input required to move the robot was found to be 9 duty while standing still. It is uncertain if anything lower would exert any force if the robot was already moving, most likely yes. The standstill measurement was important to find an appropriate starting input for the regulator. Additionally, it was found that the robot would not move straight with the same duty input given to both of the motors. Moreover, the duty input was integers which limited the amount of tuning on this variable to make the robot move evenly. The left motor was a bit stronger than the right motor and therefore was given one higher of an increment to the input. It was considered to change the drivers to be able to take floats as input to make this tuning more accurate, but it looking at the driver function it looked like the duty variable had to be an integer. There may be other workarounds to this tuning problem, but it was decided not to pursue this issue further as it would require to refactor the motor driver functions quite a bit.

Both encoders had some wheel play which will affect the accuracy of the position and angle estimation. Tightening the wheels may reduce this problem enough to make it not noticeable. As it is now, it causes a small drift in the encoder estimates, but is not enough to destroy the system behaviour if only a few commands are done. However, it does become a problem if the robot is starting and stopping in quick succession. The wheel play also made it cumbersome to accurately measure the amount of total ticks for one revolution of the wheels. In other words it is possible one revolution is not exactly 300 ticks.

The IR sensors had some damages on the cables and was fixed. The only part of the controller code that uses the IR-sensors is the collision avoidance part. The IR sensors were not implemented into the api() function during this project, but naturally it should be there as it is used for control input.

MATLAB and Development Software

Debug functionality was added to vApiTask(). This was important for development of the controller. In this project VS code was used as code environment with CMake to flash code to the nRF52840 DK via USB. The functionality relied on NRF connect for VS and NRF terminal extensions. These were very useful tools for debugging during testing.

MATLAB coder makes it easy to make functions for C code. It allows the developer to not think about notation as much while developing functions. You can utilize functions already made in MATLAB and generate them to C without having to worry about variable types, memory loss, references and so on. This makes it easy to implement new functions.

MATLAB coder has some restrictions on what can be generated. These restrictions may be that certain basic functions differ on a fundamental level and therefore cannot be translated. An example of this in this project was the modulus function in this project that needed to be custom

made in order to be generated. It was therefore extra important to generate the code regularly and look at the MATLAB documentation before using any functions to see if the functions used were compatible.

The MATLAB generated C code was not very readable. Even if the code written in MATLAB is very readable, it does not translate to the C code which is a disadvantage when trying to debug after generating.

Another problem that occurred while using MATLAB coder to make the api function was the size of the function. Even though the api function on a basic level only used the set point, encoder ticks and waiting command state to return the motor output and pose estimate, it was not able store states from previous cycles. This required the function to have states sent in as references which increased the amount the input arguments by a lot. If the variables would stay persistent between function calls, the argument list would be much smaller and the function call would be more readable. It also required initializing these variables which required changing vApiTask() while testing, which increased the added time to test the code. Attempts on using persistent variables functionality from MATLAB resulted in cumbersome debugging in vApiTask(). The code would be generated into C, but would catch errors for undefined references when compiling the code.

Integration Tests

The integration tests was important to test the system modules working together. It gave a clear indicator whenever something was wrong and ensured that all modules could work together, not only individually.

Previously, the robot was not able to receive multiple commands. The main reason for that was that it never finished its current command. Initially, it was considered if the robot was not sending idle messages to the server, but this was not the case. Adding proper stop conditions to the regulator solved the problem.

As mentioned previously implemented code the robot was not able idle after reaching its end destination. It would instead start rotating around. There were a few problems that caused this. First off, one of the encoders were not properly connected to its pin. This was identified during the initial testing of the system. Secondly, the θ measurement was a bit wrong and at last vTaskDelay() was used wrong. To prevent the robot from rotating when it reached its final destination, a stop condition was added. The stop condition is just a constant distance of maximum travel and a stop radius around the end point. The maximum distance condition would be better if it was a condition that checked if the distance travelled had exceeded the distance between the start position and the end position. This condition is a bit tricky to implement as the robot works in cycles and keeping track of the first start position is not as easy. To solve this problem, you could find out how to save the initial position when starting a new command. This may be when the new set point command is read from the queue or some other logic for knowing when a new set point has been received.

The robot still diverges from a straight line while driving forward. The drift has been reduced a substantial amount by tuning the PWM duty input to the motorswitch function, but it has not removed the drift completely. This is especially the case if it is driving long distances. The cause of the drift is mostly the thrust/voltage differences on the DC motors, but could also be caused by a slight misalignment of the two wheels. Another reason could be the terrain the robot is driving on. The ratio issue of the DC motors could not be tuned properly because of a limitation of using integers as input to the motorswitch function in the driver software. This made fine tuning diffult without refactoring motor drivers. Another possible solution to the line diverging problem, would be to make small micro adjustment to the motors while driving forward. This would require the implementation of a PD or PID controller logic in the moveForward() function.

One of the changes that was made to vApiTask() was to remove delay between reading sensor output and setting motor input. This helped the system become quicker and more responsive. However, the vTaskdelay() that was put at the start of the while loop in vApiTask() could not be removed as it was essential to maintain bluetooth connection with the server. Furthermore, it

could not be lower than 50ms because anything lower would cause the system to become unstable and crash. However, only one vTaskDelay() was needed during operation to keep the system up and running. Places where vTaskDelay() was thought to be necessary, such as before and after function calls to api() was either removed or replaced with taskYIELD() meaning the delays were set to a minimum. The delays between api() and setting input PWM duty to the motorswitch function were removed as it was causing instabilities in the control. The regulator would react to an input that happened 200ms. The necessary delay at the start of the vApitask() loop would also sometimes damage the system. This became mostly a problem while rotating the robot where it would go past the angle termination threshold. By adding bidirectional rotation regulators with smallest signed angle this problem was solved. Additionally, having a lower angular velocity during rotation helped tracking angles better.

The system seem to have some sort of built in fault tolerance. The system would reset when the system caught a out of memory (SEGFAULT) error. The reset is not always easy to detect while doing testing, and is worth to keep in mind.

The assumption made in calculation of θ where the difference in distance of the wheels equal the the arc of rotation is a potential error in the estimate. This may be a potential reason for drift after rotation.

The assumption of not moving while rotating is system flaw of the current system. Even though the robot does not move significantly while rotating, it is still something that affects the accuracy of the estimate.

Errors occurred while testing the rotateRobot() function. The robot would sporadically rotate back and forth. The sign function was added to the sTheta calculation to fix this bug.

8 Further Work

The robot still need some work in order to work properly in a square test and line test.

Adding a Variable Distance Driven Threshold

To make the system more robust to disturbances, a variable distance driven threshold should be implemented.

Adding IR Sensor into Controller

api() does not use IR sensor for control. This part of the code is still in vApiTask(). This should be moved into api() for consistency.

Adding IMU Measurements

Since the encoders seem to not work as well when rotating the robot, adding IMU measurements during rotation could be a way to fix the problem. As of now there are only IMU readings in the code, it is not being used to calculate anything.

PID Regulator

The code does not contain any PID regulator as of now. It has modes for how to move in certain scenarios, but no regulation depending on distance. The controller could benefit from a damping term.

Feed forward or Reference model

One of the problems with the robot system is that it is reactive to the environment, and not proactive. This means that it would almost never hit the target location exactly. A adding a feed forward signal to the system might be enough to fix this issue. Another way to fix it would be to add a reference model and add a prediction.

Extended Kalman Filter

The robot system is a non-linear system which contains multiple sensors. Ideally, you would like to have a prediction estimate while controlling the robot. Making an extended kalman filter would be a way to do this.

Bibliography

[1] Amazon. *uxcell DC 12V 220RPM Encoder Gear Motor with Mounting Bracket 65mm Wheel 1 Set for Smart Robot DIY*. URL: https://www.amazon.com/gp/product/B078HYX7YH/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1 (visited on 19th Dec. 2020).

[2] Maria Gilje. 'Implementing Bluetooth LE on a MATLAB controlled nRF52840 robot'. June 2022.

[3] Eivind H. Jølsgård. *Embedded nRF52 robot*. 17th Dec. 2020.

[4] Robotpark. *L298N DUAL H BRIDGE MOTOR DRIVER*. URL: https://www.robotpark.com/L298N-Dual-H-Bridge-Motor-Driver (visited on 19th Dec. 2020).

[5] Robotpark. *nRF52840 Development Kit PCA10056 v1.0.0 user guide v1.2*. URL: https://infocenter.nordicsemi.com/pdf/nRF52840_DK_User_Guide_v1.2.pdf (visited on 19th Dec. 2020).

[6] Nordic Semiconductor. *freeRTOS Real-time operating system for microcontrollers*. URL: https://www.freertos.org/ (visited on 19th Dec. 2020).

[7] Nordic Semiconductor. *L298N DUAL H BRIDGE MOTOR DRIVER*. URL: https://www.nordicsemi.com/Products/Development-hardware/nrf52840-dk (visited on 19th Dec. 2020).

[8] Nordic Semiconductor. *nRF5 SDK Software development kit for the nRF52 Series and nRF51 Series SoCs*. URL: https://infocenter.nordicsemi.com/pdf/nRF52840_DK_User_Guide_v1.2.pdf (visited on 19th Dec. 2020).

[9] Sparkfun. *Accelerometer, Gyro and IMU Buying Guide*. URL: https://www.sparkfun.com/pages/accel_gyro_guide (visited on 19th Dec. 2020).

[10] Sparkfun. *General Purpose Type Distance GP2Y0A21YK/ Measuring Sensors*. URL: https://www.sparkfun.com/datasheets/Components/GP2Y0A21YK.pdf (visited on 19th Dec. 2020).

[11] Sparkfun. *Servo - Generic Metal Gear (Micro Size)*. URL: https://www.sparkfun.com/products/14760 (visited on 19th Dec. 2020).