
Differential Drive Robot with Obstacle Avoidance

- Subtitle -

Project Report
Group Name/Number

Aalborg University
Electronics and IT

Copyright © Aalborg University 2015

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.



Electronics and IT
Aalborg University
<http://www.aau.dk>

AALBORG UNIVERSITY

STUDENT REPORT

Title:

Differential Drive Robot With Obstacle
Avoidance

Abstract:

| |
|----------------------|
| Here is the abstract |
|----------------------|

Theme:

Automation

Project Period:

Fall Semester 2016

Project Group:

ED5-8

Participant(s):

Philip Philev
Mihkel Soolep

Supervisor(s):

Simon Pedersen

Copies: 1

Page Numbers: 19

Date of Completion:

December 19, 2016

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Contents

| | |
|---|-----------|
| Preface | ix |
| 1 Introduction | 1 |
| 1.1 Examples | 2 |
| 1.2 How Does Sections, Subsections, and Subsections Look? | 3 |
| 1.2.1 This is a Subsection | 3 |
| 2 Chapter 2 name | 5 |
| 3 Modeling | 7 |
| 3.1 DC motors dynamics model | 7 |
| 3.1.1 Simulink Model | 9 |
| 3.2 Kinematics Model of Differential Drive | 11 |
| 3.2.1 Kinematics in simulink | 14 |
| 3.3 Complete model | 15 |
| 4 Conclusion | 17 |
| A Appendix A name | 19 |

Todo list

| | |
|--|----|
| <div></div> citation needed | 1 |
| <div></div> citation needed | 1 |
| <div></div> citation needed | 1 |
| <div></div> citation Automated Vehicle Policy,pdf | 1 |
| <div></div> Citation from pdf for the whole list | 1 |
| <div></div> Is it possible to add a subsubparagraph? | 3 |
| <div></div> I think that a summary of this exciting chapter should be added. | 3 |
| <div></div> I think this word is misspelled | 5 |
| Figure: We need a figure right here! | 5 |
| Figure: We need a figure right here! | 7 |
| <div></div> reference to figure | 8 |
| <div></div> reference to figure | 8 |
| <div></div> reference to the caltech report | 13 |

Preface

Here is the preface. You should put your signatures at the end of the preface.

Aalborg University, December 19, 2016

Philip Philev

<pphile14@student.aau.dk>

Mihkel Soolep

<username2@XX.aau.dk>

Chapter 1

Introduction

The Future of Vehicle Automation In recent years, a big emphasis has been put on the development of autonomous or semi-autonomous ground vehicles. It comes as no surprise considering it is no longer a question of *will* this technology be implemented, but rather *when*. The benefits of autonomous vehicle integration can be considered invaluable. Currently 90% of motor vehicle fatalities are estimated to be due to human errors, meaning that vehicle automation could result in substantial decrease of accidents. Furthermore, depending on the percentage of autonomous vehicles on the roads, a research concluded, a drastic reduction in traffic and congestions.

citation needed

citation needed

citation needed

Nonetheless, there is still much work to be done in perfecting the control as well as the sensing capabilities of autonomous ground vehicles, if they are to become the default means of automotive transportation. Some of the issues consist of environmental conditions, which may disturb the sensors accuracy; precise mapping awareness, such as live maps that update when there is ongoing maintenance of infrastructure etc.; improved sensing capabilities (e.g advanced lidars) that can differentiate road damage, liquid spills etc.; ethical choices (as when an accident cannot be avoided), choosing to minimize potential damage and avoid casualties.

Levels of Automation Automated vehicles, as defined by the *National Highway Traffic Safety Administration*(NHTSA - USA), are ones in which at least some aspects of a safety-critical control function occurs without the operator's direct input.(e.g steering, throttle,braking etc.)As such they are classified by the NHTSA in five levels:

citation Automated Vehicle Policy,pdf

- **Level 0 - No Automation**

Logically, this level does not include any direct automation functions, however it may include some warning systems such as blind spot monitoring. The operator has the complete control over the vehicle.

Citation from pdf for the whole list

- **Level 1 - Function Specific Automation**

The system may utilize one or more control functions operating independently from each other, such as cruise control or dynamic brake support. Nevertheless the driver has over control and can limit the functions of the supported aid systems.

- **Level 2 - Combined Function Automation**

The system utilizes at least two primary control functions, intercommunicating with each other in order to allow the operator's disengagement from physical operation of the vehicle. An example of such is a combination between *adaptive cruise control* and *lane centering*. The driver is still responsible for monitoring the environment, even when automated operating mode is enabled.

- **Level 3 - Limited Self-Driving Automation**

The driver accepts to cede full control of all safety-critical functions under certain conditions, and rely completely on the vehicle to monitor the environment if a transition toward manual control is required. Such level of control is observed in automated or self-driving vehicles that conclude when the system is unable to handle an environment, such as road construction site, requiring specific manoeuvres. The driver is not expected to fully pay attention to the road, but is advised to pay attention to sudden changes.

- **Level 4 - Full Self-Driving Automation**

Vehicle is designed to solely operate all safety-critical functions and supervise road conditions. Apart from providing destination input, the driver is not expected to maintain control at any point of the trip.

1.1 Examples

You can also have examples in your document such as in example 1.1.

Example 1.1 (An Example of an Example)

Here is an example with some math

$$0 = \exp(i\pi) + 1 . \tag{1.1}$$

You can adjust the colour and the line width in the `macros.tex` file.

1.2 How Does Sections, Subsections, and Subsections Look?

Well, like this

1.2.1 This is a Subsection

and this

This is a Subsubsection

and this.

A Paragraph You can also use paragraph titles which look like this.

A Subparagraph Moreover, you can also use subparagraph titles which look like this. They have a small indentation as opposed to the paragraph titles.

I think that a summary of this exciting chapter should be added.

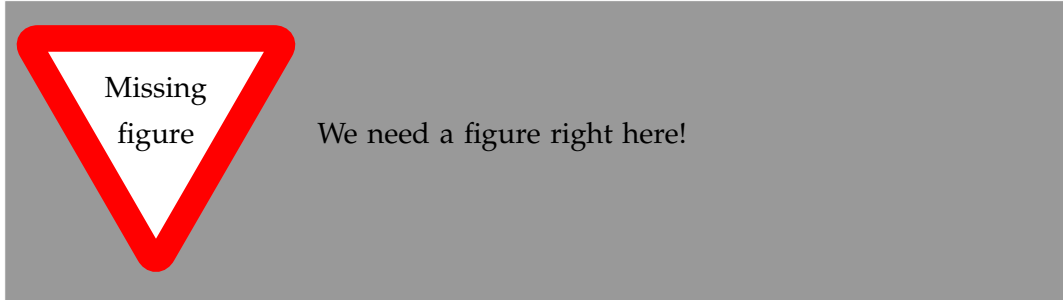
Is it possible to add a subsubparagraph?

Chapter 2

Chapter 2 name

Here is chapter 2. If you want to learn more about $\text{\LaTeX}2_{\epsilon}$, have a look at [Madsen2010], [Oetiker2010] and [Mittelbach2005].

I think this word is misspelled



Chapter 3

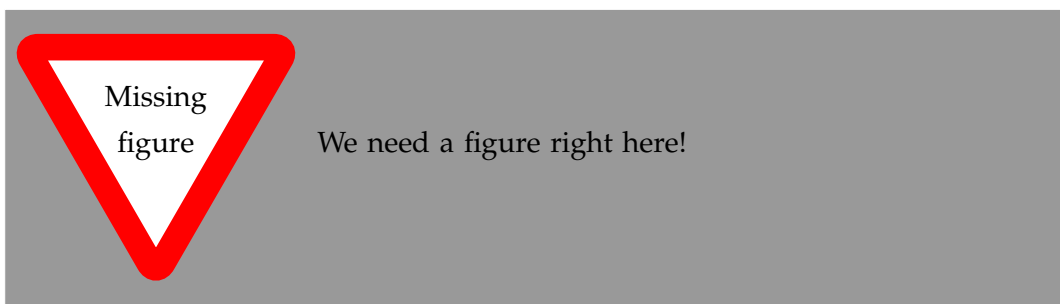
Modeling

In order to understand the behaviour of the system, a mathematical model followed by a simulation had to be done.

3.1 DC motors dynamics model

| Parameter | Description | Nominal Value |
|-----------|---------------------|----------------------------|
| K | Motor constant | 0.1838 V/(rad/s) Nm/amp |
| R | Armature resistance | 11.5 Ω |
| L | Armature inductance | 0.1 H |
| J_r | Rotor inertia | 0 |
| b_r | Rotor damping | 0.0221 |
| J_w | Load inertia | 2.8033e-5 KgM ² |
| n | Gear ratio | 1:48 |

Table 3.1: Motor parameters



This section describes the dynamic mathematical model of the DC motors, including moment of inertia, torque and friction. In a DC motor the produced elec-

tromagnetic torque(T_e) is linearly proportional to the armature current and the magnetic field. If we assume that the magnetic field is constant, the torque is only proportional to the armature current(I) and the torque constant(K_t) as evident in equation 3.1.

$$T_e = IK_t \quad (3.1)$$

The back electromotive force voltage(E_b) is proportional to the angular velocity(ω) of the shaft times the Back emf constant(K_b). (Equation 3.2)

$$E_b = \omega K_b \quad (3.2)$$

Because the two constants K_t and K_b are equal in SI units, in further equations and simulations they will be denoted only as a motor constant K .

$$K_t = K_b = K \quad (3.3)$$

reference to figure

Furthermore, from figure , using Kirchhoff's voltage law, we can derive the equations governing the electrical part of the DC motor, where the applied voltage (V) is proportional to the voltage drop through the armature resistance(R) and inductance(L), and the back electromotive voltage(E_b). 3.4

$$V = RI + L \frac{dI}{dt} + E_b \quad (3.4)$$

reference to figure

The mechanical part of the DC motor (mechanical part of figure) is derived from the equations, where the mechanical torque(T_m) is the difference between the electromagnetic torque(T_e) and the rotational losses (T_b). 3.5

$$T_m = T_e - T_b \quad (3.5)$$

Using Newton's second law for rotational motion and substituting from equation 3.1, we can rewrite equation 3.5 as:

$$J\dot{\omega} = KI - b\omega \quad (3.6)$$

Where J is the load's inertia and b is the viscous friction in the motor's bearings. Further substitution in equation 3.4 with the derived back emf from 3.2 results in:

$$V = RI + L \frac{dI}{dt} + K\omega \quad (3.7)$$

Equations 3.6 and 3.7 are the combined equations of motion for the DC motor.

Applying the Laplace transform to the equations, we can derive the transfer function of the DC motor.

$$\begin{aligned} sJ\Omega(s) + b\Omega(s) &= KI(s) \\ sLI(s) + RI(s) &= V(s) - K\Omega(s) \end{aligned} \quad (3.8)$$

\Downarrow

$$\begin{aligned} \frac{\Omega(s)(sJ + b)}{K} &= I(s) \\ I(s)(sL + R) + K\Omega(s) &= V(s) \end{aligned} \quad (3.9)$$

Substituting with $I(s)$ in the second part of equation 3.9, and setting the angular velocity($\Omega(s)$) as output and the voltage ($V(s)$) as input results in the transfer function for the DC motor.(3.10)

$$\frac{\Omega(s)}{V(s)} = \frac{K}{(Js + b)(sL + R) + K^2} \quad (3.10)$$

3.1.1 Simulink Model

In this subsection, the previously derived equations are represented in a block diagram using Matlab's Simulink environment. There are several possible ways to arrange the blocks governing the DC motor, thus in this paper a familiar approach is considered.

| Parameter | Description | Nominal Value |
|-----------|---------------------|----------------------------|
| K | Motor constant | 0.1838 V/(rad/s) Nm/amp |
| R | Armature resistance | 11.5 Ω |
| L | Armature inductance | 0.1 H |
| J_r | Rotor inertia | 0 |
| b_r | Rotor damping | 0.0221 |
| J_w | Load inertia | 2.8033e-5 KgM ² |
| n | Gear ratio | 1:48 |

Table 3.2: Motor parameters

As evident from equation 3.10, the voltage is the input of the system, while the angular velocity is the output. In order to accurately apply the equations, while attaining the desired result, a modification of equations 3.6 and 3.7 was made.(3.11)

$$\begin{aligned}\frac{dI}{dt} &= \frac{1}{L}(V - RI - K\omega) \\ \frac{d\omega}{dt} &= \frac{1}{J}(KI - b\omega)\end{aligned}\quad (3.11)$$

The block diagram representation in figure 3.1 has the integrals of the rotational acceleration and the rate of change of the armature current considered as outputs based on equations 3.11.

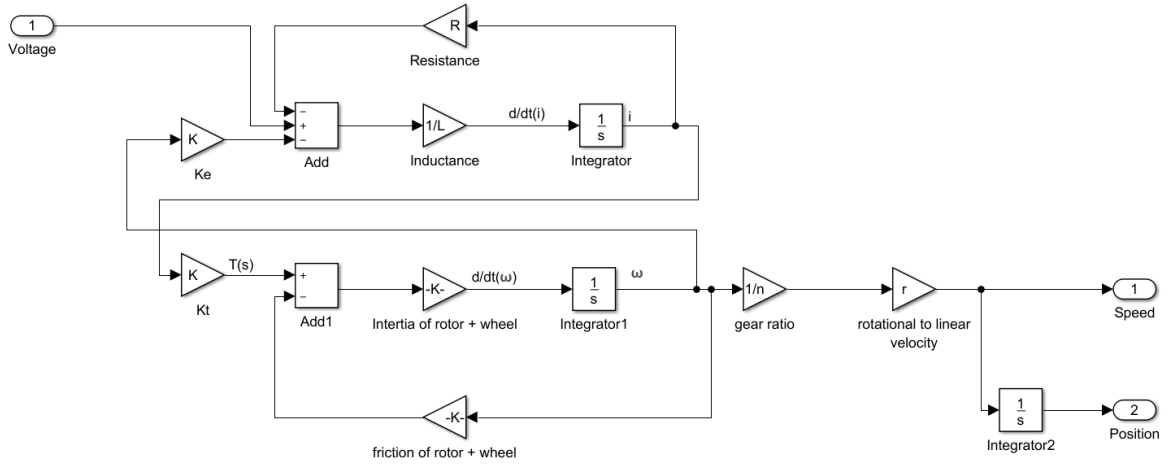


Figure 3.1: DC Motor Block Diagram

The inclusion of the gear ratio (n) and the radius of the wheel (r) products to the angular velocity in the end of the block diagram, results in model scaling for the linear velocity (v) of the wheel. (3.12)

$$v = r\omega \quad (3.12)$$

Performing integration on the derived linear velocity results in obtaining the linear displacement of the wheels, later to be used with the kinematics model.

To summarise, the goal was to relate the voltage to the speed. The input of the block diagram is the voltage of the motor (V) while the outputs are the linear speed caused by wheel rotation and the linear displacement, obtained from integrating the speed. The blocks comprising the upper and lower part of the block diagram, directly correspond to equation 3.11 (upper part correspond to the electrical part of the motor; lower part correspond to the mechanical part of the motor).

Furthermore, as this paper is concerned with the development of a differential drive robot, the block diagram in figure 3.1 is solely a subsystem of the complete

kinematics model. That is, two DC motor subsystems are required in order to describe the complete motor/wheel dynamics.

3.2 Kinematics Model of Differential Drive

Differential drive is a common mechanism in mobile robotics. It consists of two wheels on a common axis, driven by two motor, where each wheel can be independently driven in either forward or backward direction. That is, by varying the velocities of each wheel, different trajectories could be achieved. Importantly, the rotation the robot performs is based on a point common to the right and left wheel axis, denoted as Instantaneous Center of Curvature(ICC). The kinematic representation can be observed in figure 3.2 .

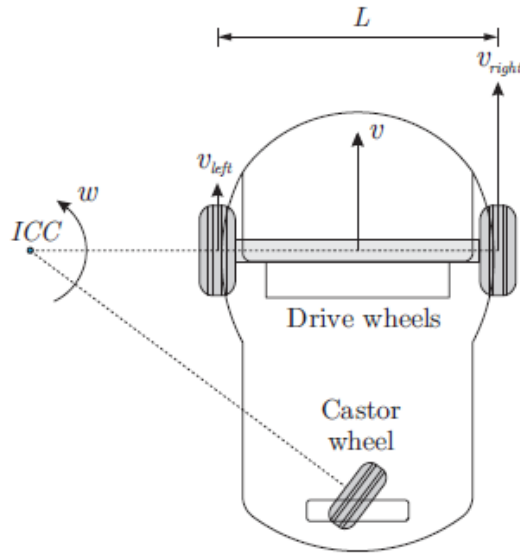


Figure 3.2: Differential drive overview

The linear speed, previously derived from chapter 3.1.1, is represented as v_r and v_l , for each of the wheels. The speed of the robot is taken as the average speed of each wheel.

$$v = \frac{v_r + v_l}{2} \quad (3.13)$$

The angular speed of the robot (or turning speed) is based on the linear speeds of each wheel and the distance between the wheels. It is denoted as W in equation 3.14.(not to be confused with ω , the rotational speed of the motor)

$$W = \frac{v_r - v_l}{l} \quad (3.14)$$

The relation between the linear speed and the angular speed of the robot is similar to equation 3.12.

$$v = WD \quad (3.15)$$

Where D is the turning radius, from the midpoint of the wheels to the ICC. Solving for the turning radius, yields:

$$\begin{aligned} D &= \frac{v}{W} \\ &= \frac{l}{2} \frac{v_r + v_l}{v_r - v_l} \end{aligned} \quad (3.16)$$

Analysis of the above equation leads to the consideration of three cases where certain behaviour is to be expected.

- $v_r = v_l$
In this case scenario, both wheels have the same speed. The robot's speed from equation 3.13 is simply equal to the individual speed of each of wheel. On the other hand, the angular speed from equation 3.14 becomes 0, and the turning radius infinite. The robot is expected to perform straight linear motion.
- $v_r = 0$ or $v_l = 0$
In this case scenario, the turning radius becomes $\frac{l}{2}$. The robot is expected to perform rotation either about the right or the left wheel, with the center of rotation being the zero velocity wheel.
- $v_l = -v_r$
In this case scenario, the turning radius and the linear speed become 0, while the angular speed is doubled. The robot is expected to perform rotation about it's midpoint, or simply put in-place rotation.

It is important to mention that the wheels, present in the system, carry some **nonholonomic** constraints. That is, the robot's local movements are restricted, while no restrictions are present in the global navigation. We can further extend the idea with the use of generalised coordinates.

$$q = (x, y, \theta) \quad (3.17)$$

Equation 3.17 could be seen as a point on a two dimensional Cartesian coordinate system, where x and y are the axis and θ is the angle between the x axis and the point.

Figure 3.3 shows the robot representation based on the generalised coordinates.

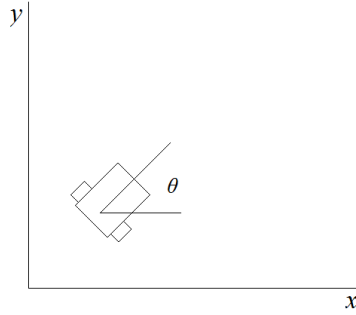


Figure 3.3: Robot representation in Cartesian coordinates

We can picture the constraints for the wheels by setting the sideways velocity to zero. That is, the robot can not perform sideways movements like slipping or sliding, but is not limited from manoeuvring in that position, whatsoever.

$$\dot{x}\sin(\theta) - \dot{y}\cos(\theta) = 0 \quad (3.18)$$

Equation 3.18 is a common nonholonomic constraint in mobile robotics.

Particularly if the robot is viewed as a point, the Kinematics equations in Cartesian space can be derived as:

reference to the caltech report

$$\begin{aligned} \dot{x} &= v\cos(\theta) \\ \dot{y} &= v\sin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \quad (3.19)$$

In the case of a differential drive robot, substitution of the linear and angular velocities, v and ω , with the previously derived in equation 3.13 and 3.14 average robot speed and angular robot speed (with respect to the center of rotation between the wheels), will result in the kinematics equations for locomotion of a differential drive.

$$\begin{aligned} \dot{x} &= \frac{v_r + v_l}{2}\cos(\theta) \\ \dot{y} &= \frac{v_r + v_l}{2}\sin(\theta) \\ \dot{\theta} &= \frac{v_r - v_l}{l} \end{aligned} \quad (3.20)$$

In equation 3.20 we will have a change in the robot's position (x,y,θ) when the velocity of each wheel is controlled.

3.2.1 Kinematics in simulink

| Parameter | Description |
|-----------|-----------------------------|
| l | Distance between wheels |
| v_r | Linear speed of right wheel |
| v_l | Linear speed of left wheel |
| v | Average linear robot speed |
| W | Angular robot speed |
| D | Turning radius |

Table 3.3: Kinematics parameters

Using equations 3.13 and 3.14 a simulink block diagram is constructed in figure 3.4.

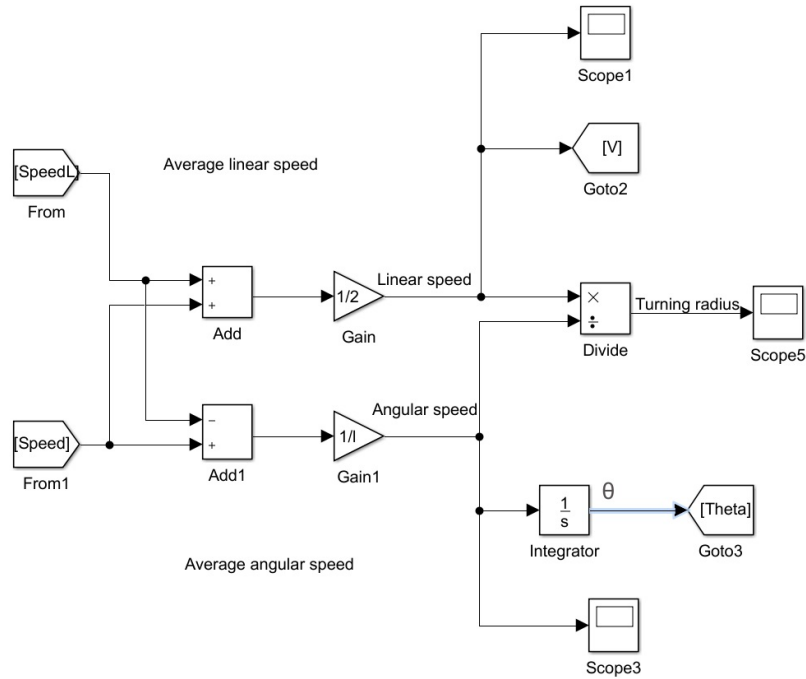


Figure 3.4: Differential drive kinematics

The inputs are the individual wheel velocities, arranged to reflect the previously mentioned equations. The middle output is the turning radius D , which is governed by equation 3.16. The derived average linear speed is to be further used in equation 3.20 to estimate the position of the robot based on its angle through time, where the angle (θ) is obtained from integrating $\dot{\theta}$, which itself is the angular speed of the robot.

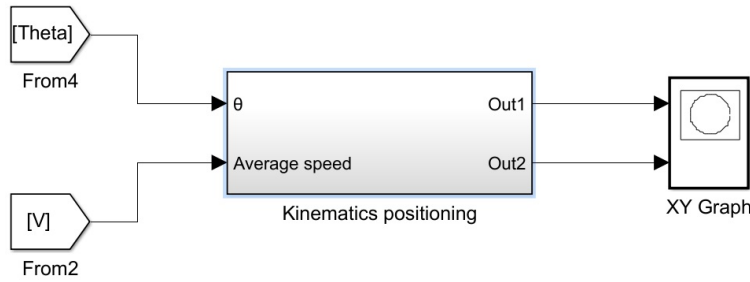


Figure 3.5: Positioning subsystem

In figure 3.5 the inputs are the the average speed and the orientation of the robot, while the outputs X and Y from equation 3.20 are fed in a graph to observe the robot's trajectory through time.

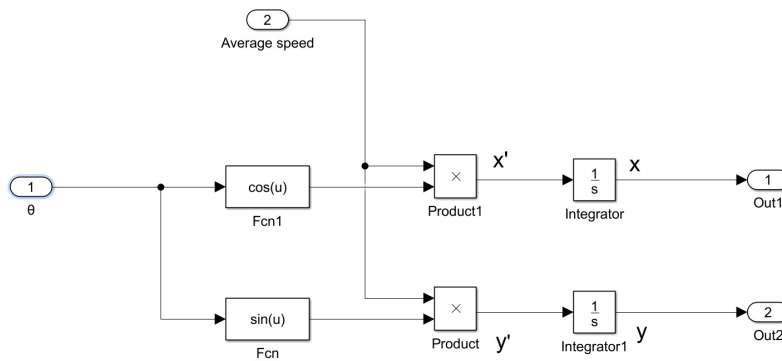


Figure 3.6: Positioning subsystem

The subsystem from figure 3.5 is composed of blocks arranged to reflect equation 3.20. The outputs \dot{x} and \dot{y} are further integrated to graph the trajectory of the robot.

3.3 Complete model

In this section the complete system model is derived. It includes motor dynamics and robot kinematics. That is, the behaviour of the robot is analysed and compared

with the expected performance.

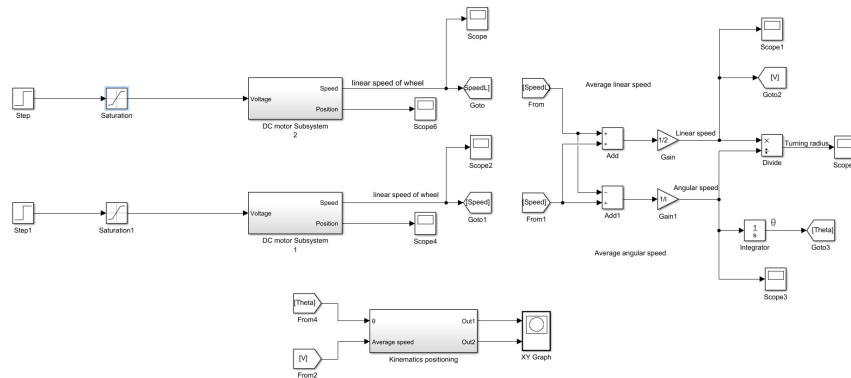


Figure 3.7: Complete system model

In figure ??, the complete system model for a differential drive robot could be observed. As previously mentioned, a differential drive consist of two DC motors, thus to accurately model the relations, two motor subsystem as described in section 3.1.1, are used.

3.3.1 DC motors model analysis

The two DC motors are in the top left corner of the model in figure ?. A step function block is used to simulate the voltage applied to the motor, alongside a saturation block that limits the model from computing with values greater than the maximum allowed voltage in the physical motor.

The motor model proposed in section 3.1.1 is constructed using only linear blocks, thus the open-loop response could be observed by providing a unit step input and observing the response through a scope block.



Figure 3.8: Step response of DC motor

Figure ?? is consistent with the expected step response of a DC motor. When 1

Volts is applied as a step input, the motor achieves maximum speed of 0.06 cm/s (after conversion to linear speed) However to further understand the results, linear analysis on the subsystem has been performed.

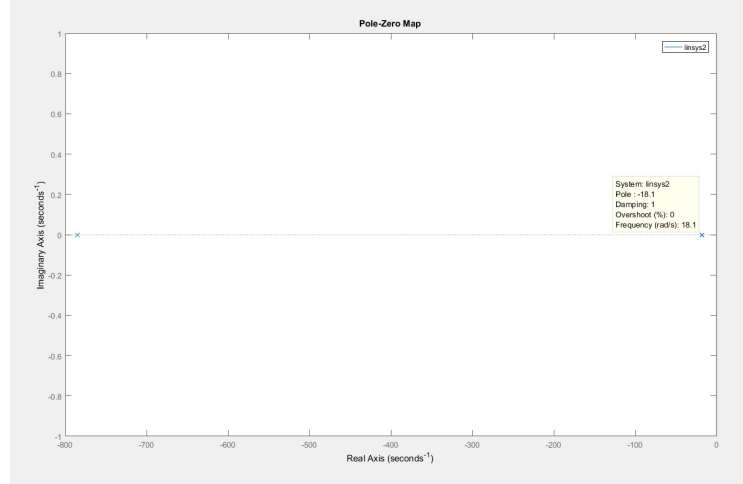


Figure 3.9: Pole-Zero map of DC motor

From figure ??, it is clear that the open-loop subsystem has two real poles in the left hand plane. That is ,no oscillations or overshoot present as it could be seen in the step response. Furthermore, the slower of the two poles will dominate the dynamics of the system, prompting the system to behave as it was first-order.

Obviously, both motors' models behave the same while applying the same step input. Nevertheless, it is important to understand that in real life this may not be the case, as the constants used in the model may not reflect accurately both real-life counterparts.

3.3.2 Kinematics model analysis

We discussed in subsection ?? how small fluctuation in the speed of each motor, will results in a change of the trajectory of the whole robot. It is important to verify that the cases laid in the above mentioned subsection hold true.

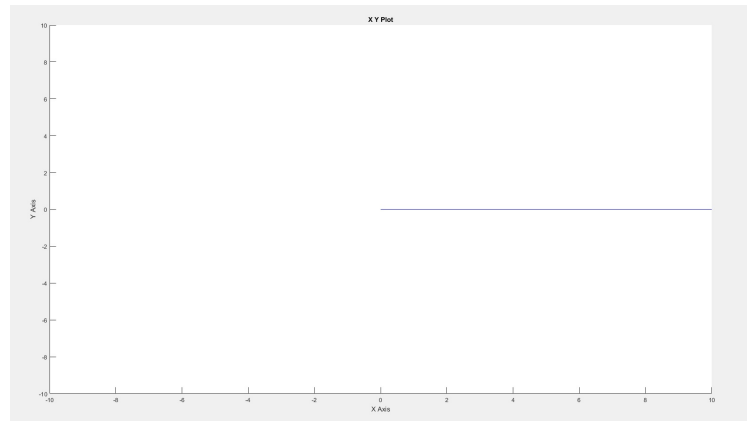


Figure 3.10: XY-graph for equal velocities

In figure ??, when both wheel velocities match ($v_r = v_l$), the trajectory the robot partakes is a straight line.



Figure 3.11: Turning speed for equal velocities

As to be expected, the turning speed in figure ?? remains zero for as long as the velocities of each wheel match.

The other case scenario is when one of the wheels has zero velocity (the other wheel's velocity can not be equal to zero).

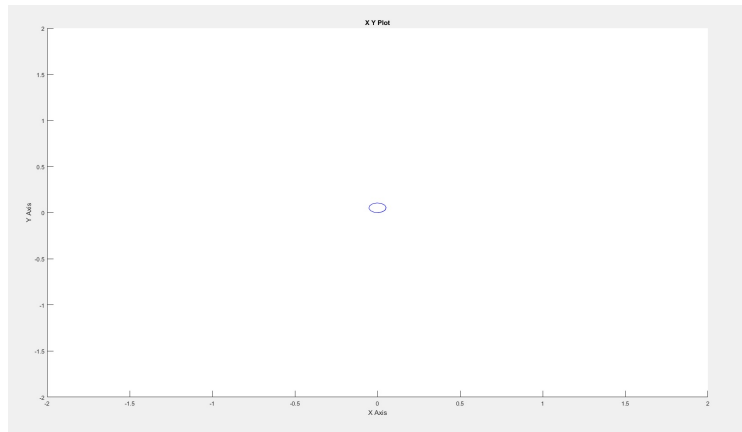


Figure 3.12: XY-graph for zero velocity left wheel

The expected behaviour is rotation where the ICC is positioned at the zero velocity wheel. Furthermore, the turning speed should be constant, while the turning radius equal to $\frac{l}{2}$.

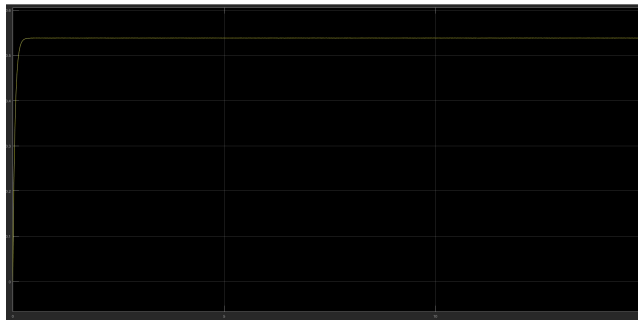


Figure 3.13: Turning speed for zero velocity left wheel

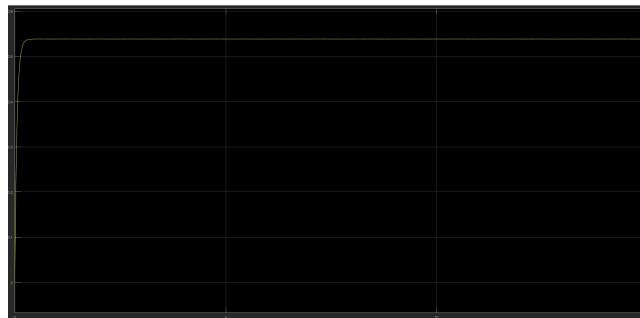


Figure 3.14: Turning radius for zero velocity left wheel

Chapter 4

Hardware

In the following chapter the hardware used to create the robot is discussed. The first decision made was what platform to use, based on the available resources and knowledge of use.

4.1 Single board computer

The Raspberry Pi is the platform of choice used in the development of this project. Several factors influenced the choice, the most prominent of which was the familiarity the single board computer offered. Previous usage of Raspberry Pi 2B, allowed for faster decision making, when exploring for potential resources.

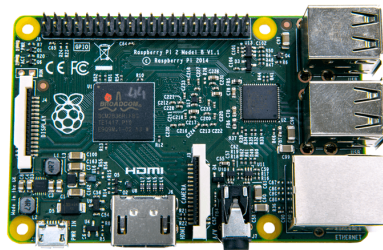


Figure 4.1: Raspberry Pi 2B

The initial choice was a Raspberry Pi 2B, however, after consideration of the size and required power consumption, a choice was made to use Raspberry Pi Zero instead.

Performance of the two computers is almost similar, with the exception that the Zero model has less RAM compared to its 2B counterpart. Nevertheless, no significant impact was registered.

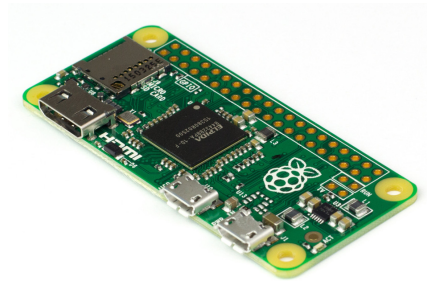


Figure 4.2: Raspberry Pi Zero

Specs of the Raspberry Pi Zero: 1Ghz, Single-core CPU 512MB RAM Mini HDMI and USB On-The-Go ports Micro USB power HAT-compatible 40-pin header Composite video and reset headers

4.2 Distance measuring

Since the main purpose of the device would be to avoid obstacles, distance measuring sensors are required. There are several choices when it comes to distance sensors, vastly ranging in price and accuracy. The optimal choice would be a Lidar, however, at the time of development of this paper, the prices for Lidar were too high for practical implementation in a consumer application. As a result, the choice was made to use a set of ultrasonic sensors, which scaled well according to price and accuracy of measurements. Three HC-SR04 ultrasonic sensors were used for distance measuring purposes in the device. The positioning of the sensors is on the front, and on the sides.

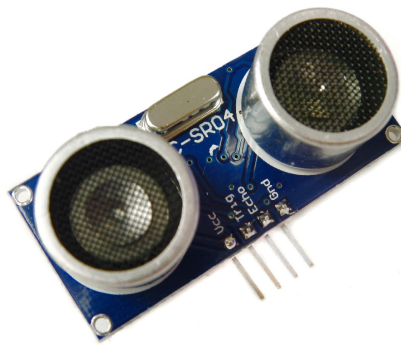


Figure 4.3: Ultra sonic sensor HC-SR04

4.3 DC motors and chassis

Considering the body of the robot, an inexpensive set of DC motors with a chassis was purchased and assembled. It comes with an attachable tachometer wheel for the motor, in order to incorporate a rotary encoder in the design, crucial for regulating the speed of the individual wheels.

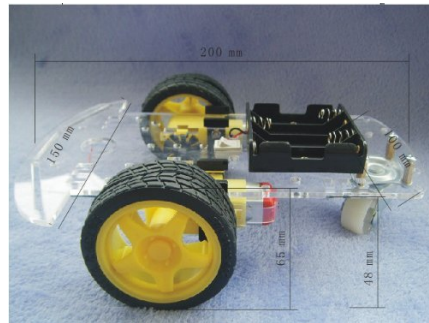


Figure 4.4: Chassie and the two DC motors with a power supply

Motor specs:

Voltage: DC 3V DC 5V DC 6V Current: 100 MA 100MA 120MA Reduction rate:48:1 RPM (With tire):100,190,240 Tire Diameter:66mm Car Speed(M/minute):20,39,48

Motor Weight (g):50

Motor Size:70mm*22mm*18mm

Noise:<65dB

The two motors are identical and are used to move, as well as steer, the device and to

4.4 Motor driver

At the beginning of the project a L9110S DC Stepper Motor Driver H-Bridge was used, capable of controlling the direction of rotation of the wheels, but it was soon discovered the suggested driver was not capable of regulating the motor speeds. To control the speed of the robot, a replacement with a L298N driver had to be performed. The second driver was capable of using the PWM to regulate the speed of the motors.

Driver specs: Working mode: H bridge (double lines) Control chip: L298N (ST) Logical voltage: 5V Driving voltage: 5V-35V Logical current: max 36mA Driving current: 2A (max single bridge) Maximum power: 25W Storage temperature: -20 C +135 C Periphery dimension: 43 x 43 x 27mm(L x W x H)

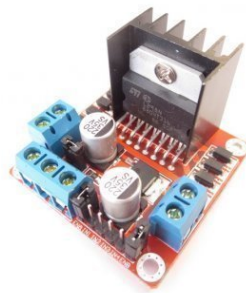


Figure 4.5: The L298N driver

4.5 Speed sensor

The speed of the wheels is measured by the LM393 IR speed sensors attached to the tachometer wheel of the motor.



Figure 4.6: The LM393 IR speed sensor

The speed sensor is used to estimate the error of the wheel speed. Features:

Working voltage: 3.3V 5V Weight: 8g Dimensions: Approx. 3.2 x 1.4 x 0.7cm 5mm Groove width Using wide voltage LM393 comparator Application: Widely used in dynamo speed detecting, pulse counting, etc Output form: Digital switch output (0 and 1) and Analog for Sensitivity.

4.6 Wi-Fi adapter

In order to perform wireless monitoring, as well as extend the connectivity an Edimax EW-7811Un Wi-Fi adapter, was used.



Figure 4.7: Edimax EW-7811Un Wi-Fi adapter

4.7 Assembly

In this section, the schematics of the assembled components has been analysed and given in figure ??.

In the middle section of the schematics, the Raspberry Pi is present. Unfortunately, the software used to make this schematic did not have the Raspberry Pi Zero layout, so the model 2 was used instead. Regarding the project there is not a big difference, as the pins are the same as Zero.

On the right side of the system three ultrasonic sensors could be seen, connected to the Raspberry Pi. The Vcc and GND pins are all connected to the Raspberry, allowing the sensors to draw power from the microcontroller itself. The trigger pins are all connected to the same Raspberry pin, meaning that when one of the sensors is triggered all of them will emit sonic impulse. The triggers are all on the same pin to not limit the number of available pins on the Raspberry. However there is not a significant difference, as the echo pins of each of the sensors go into separate ports on the Raspberry and each of them has a voltage divider connected to the GND.

The driver is present in the bottom left of the schematics, alongside a battery pack connected to the driver and providing power to both motors. It is connected to the Raspberry by six pins, four responsible for the direction of the motors (2 for each motor) and 2 enable pins, used for controlling the speed of the motor by PWM.

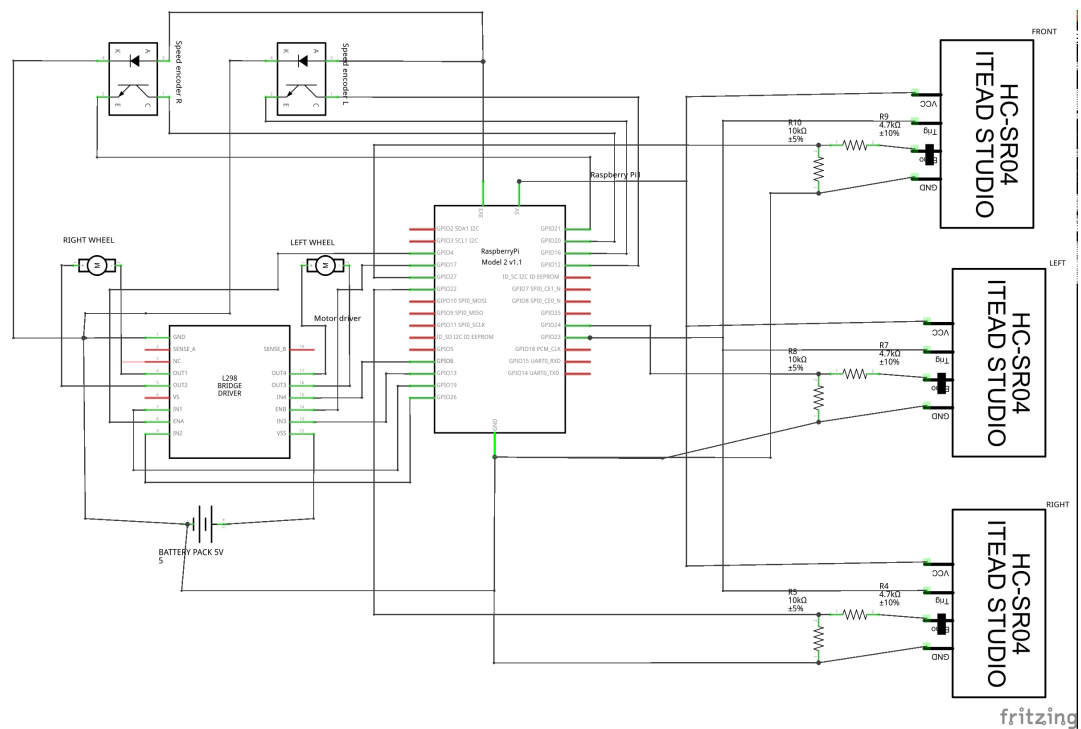


Figure 4.8: Schematics of the device

The motors are connected directly to the driver, by two wires. The two encoders, responsible for monitoring the speed of the wheels, are present in the top left of the schematics. An external power source was used for the Raspberry Pi, which is not present in this schematics.

Chapter 5

Development

Hardware testing

5.1 Ultrasonic sensors

The first part of the hardware outline was testing of the ultrasonic sensors. For each of the sensors a voltage divider was build as seen on figure /refvoltage1.

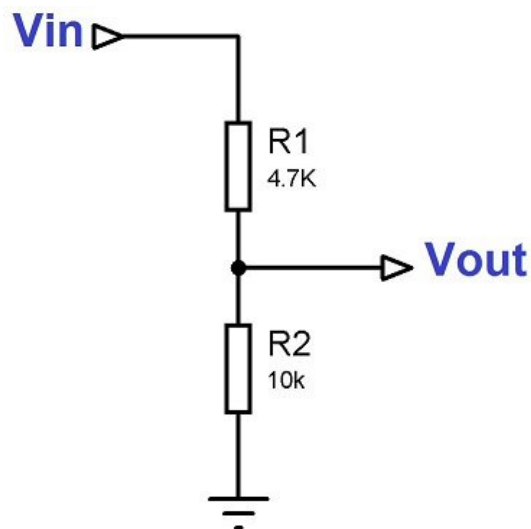


Figure 5.1: Voltage divider

The values of the resistors are calculated by the following equation:

$$V_{out} = V_{in} * R_2 / (R_1 + R_2) \quad (5.1)$$

All sensors were tested out separately by connecting them individually to the Raspberry pi and running the test code below

```

import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)

TRIG = 23
ECHO = 24

print "Measuring distance"

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

while True:
    GPIO.output(TRIG, False)
    print "Waiting for the sensor"
    time.sleep(2)

    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    while GPIO.input(ECHO)==0:
        pulse_start = time.time()

    while GPIO.input(ECHO)==1:
        pulse_end= time.time()

    pulse_duration = pulse_end - pulse_start

    distance = pulse_duration * 17150
    distance = round(distance, 2)

    print "Distance:%d", distance

```

Each of the sensors worked as expected while connected separately, thus the logical progression was to test them out all at the same time. While taking the readings, it was noticeable that some of the values were random or distorted due to noise and could not be used for accurate measurement. Therefore, some filtering had to be included in order to scale the values to an acceptable limit. A moving average filter was used for the previously mentioned task by taking three readings at a time and calculating the average of the values.

```
def readsensor(PIN):
```

```

    for x in range(0, 2):
        read_time_start1 = time.time()
        GPIO.output(TRIG, True)
        time.sleep(pulse)
        GPIO.output(TRIG, False)

        while GPIO.input(PIN)==0:
            pulse_start = time.time()

        while GPIO.input(PIN)==1:
            pulse_end= time.time()

        pulse_duration[x] = pulse_end - pulse_start
        time.sleep(0.05-(time.time()-read_time_start1))

    distance = sum(pulse_duration)/measurment_count* SPEED_OF_SOUND
    distance = round(distance, 2)
    print distance
while True:
    readsensor(ECHOF)
    readsensor(ECHOR)
    readsensor(ECHOL)

```

From the code above, the application of the moving average filter could be seen. Importantly there was a noticeable improvement of the readings, which became much more precise and consistent. Furthermore, it can be seen that every reading takes less than 0.05 seconds. This is useful in making every cycle evenly long and predictable, when it comes to the total time that the program is required to run the full cycle. Therefore the time for the full sensor reading loop becomes $3 \times 0.05s = 0,15s$.

5.2 DC motors

To make sure the purchased DC motors were operational an initial testing was performed by driving each of them in forward and in backward gear through the used driver. This can be seen in script below.

```

GPIO.setmode(GPIO.BCM)
GPIO.setup(StepPinForward, GPIO.OUT)
GPIO.setup(StepPinBackward, GPIO.OUT)

def forward(x):

```

```

GPIO.output(StepPinForward , GPIO.HIGH)
print "forwarding running motor "
time.sleep(x)
GPIO.output(StepPinForward , GPIO.LOW)

def reverse(x):
    GPIO.output(StepPinBackward , GPIO.HIGH)
    print "backward running motor"
    time.sleep(x)
    GPIO.output(StepPinBackward , GPIO.LOW)

print "forward motor "
forward(5)
print "reverse motor"
reverse(5)

print "Stopping motor"
GPIO.cleanup()

```

As it can be seen from the code, one of the motors was initially driven forward for 5 seconds and subsequently backwards for 5 seconds. To apply the test for the second motor simply the pin numbers(StepPinForward and StepPinBackward) were changed.

Furthermore, Pulse Width Modulation (PWM) was utilized in order to regulate the separate speeds of the motors, as it was an initial condition to implement cruise control. in the final software what is ran on the device we have changed the forward() and reverse() so that we can change the speed of the motors at our desire.

```

def forward(forwardtime ,SPEED):
    print "REVERSE"
    GPIO.output(StepPinBackward1 , GPIO.HIGH)
    GPIO.output(StepPinBackward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    time.sleep(forwardtime)
    GPIO.output(StepPinBackward1 , GPIO.LOW)
    GPIO.output(StepPinBackward2 , GPIO.LOW)

```

As you can see from the code above we use the drivers PWM input to change the speed of the vehicle.

5.3 Raspberry Pi configuration and software

In this section, a description of the steps taken to configure the Raspberry Pi, was made.

5.3.1 Raspberry setup

Initially, after obtaining the Raspberry Pi Zero, an installation of the Raspbian OS was performed. Subsequently, all GPIO pins were enabled, followed by configuration of the Secure Shell (SSH) protocol in order to perform remote logins to the microcontroller. Furthermore, as mentioned in the Hardware chapter, due to its size, the Raspberry Zero has only one USB port, resulting in a space limitation. Thus, the logical choice was to connect the Wi-Fi adapter to that USB port, and configure the microcontroller remotely, through the SSH. Additionally, extra tools such as Tmux Multitab and Nano Text Editor were used to clarify the programming sessions.

5.3.2 Software

The software constituting the robot's behaviour is self-made with the addition of several external libraries

```
import sys
import time
import RPi.GPIO as GPIO
```

To be precise, only three external libraries were used at the time of the development of this paper. Nevertheless, as the system is still in active development the list may expand after the submission of the documentation.

- **Sys module**

This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment.

citationsys

- **Time module**

This module provides a number of functions to deal with dates and the time within a day. It is a thin layer on top of the C runtime library. A given date and time can either be represented as a floating point value (the number of seconds since a reference date, usually January 1st, 1970), or as a time tuple.

citationtime

- **RPi.GPIO module**

This module is for functions that are connected to the GPIO pins.

The overall setup of the pins and the variables, with the different values used in the code, is present in the snippet below.

```
#PIN numbers
LeftPWM=16
RightPWM=20
StepPinForward1=26
StepPinBackward1=19
StepPinForward2=13
StepPinBackward2=6
ECHO_F=4
ECHO_L=27
ECHO_R=22
TRIG=17

#Values for reading the sesnsors
SPEED_OF_SOUND = 17150
measurment_count = 3
pulse = 0.00001
pulse_duration = [0,0,0]
sensorF_data=0
sensorR_data=0
sensorL_data=0

#navigation variables
reversetime=0
turningtime = 1
MAXSPEED = 1
MEDSPEED = 0.6
MINSPEED = 0.1

#GPIO setup for each pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(StepPinForward1, GPIO.OUT)
GPIO.setup(StepPinBackward1, GPIO.OUT)
GPIO.setup(StepPinForward2, GPIO.OUT)
GPIO.setup(StepPinBackward2, GPIO.OUT)
GPIO.setup(ECHO_F, GPIO.IN)
GPIO.setup(ECHO_L, GPIO.IN)
GPIO.setup(ECHO_R, GPIO.IN)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(LeftPWM, GPIO.OUT)
GPIO.setup(RightPWM, GPIO.OUT)
```

```
#PWM channels and frequency
PWML=GPIO.PWM(16, 0.5)
PWMR=GPIO.PWM(20, 0.5)
```

5.3.3 Ultrasonic sensor reading

As previously mentioned, the ultrasonic sensors are the proposed way of implementing collision detection. The desired function is for them to measure the distance from the robot to the closest present obstacle, and return the data for further usage in the code. In subsection ??, it was mentioned of the potential problems with noise and the proposed filtering method, using the average of every three reading. Below is a section of the code, with the function for data acquisition after filtering the readings.

```
def readsensor(PIN):
    for x in range(0, 2):
        read_time_start1 = time.time()
        GPIO.output(TRIG, True)
        time.sleep(pulse)
        GPIO.output(TRIG, False)

        while GPIO.input(PIN)==0:
            pulse_start = time.time()

            while GPIO.input(PIN)==1:
                pulse_end= time.time()

            pulse_duration[x] = pulse_end - pulse_start
            time.sleep(0.05-(time.time()-read_time_start1))

    distance = sum(pulse_duration)/measurment_count* SPEED_OF_SOUND
    distance = round(distance, 2)
    print distance
    if PIN == ECHOF:
        sensorF_data=distance

    if PIN == ECHOR:
        sensorR_data=distance

    if PIN==ECHOL:
        sensorL_data=distance
```

The data reading is straightforward. The **TRIG** pin emits an ultrasonic impulse from each of the sensors in addition to a digital timestamp that records the exact time of the event. When the impulse reaches back the sensor, a second timestamp is recorded. The logic behind it is that every loop should take an equal amount of time for execution. The first timestamp in the beginning of the loop is initialized with $read_time_start1 = time.time()$, and calculated afterwards in $time.sleep(0.05 - (time.time() - read_time_start1))$. The expected cycle length is 0.05 second and is done to ensure that readings are acquired in the sensing range 0.02m to 4m. (For further information consult with the data sheet in Appendix . The set time of 0.05 second is used, as it would give the range, limited by time.

ref to appendix

$$330(m/s) * 0.05s = 16.5 \quad (5.2)$$

And since the impulse is going to the object in range and returning to the sensor, a division by 2 will give the max range for the time limitation

$$16.5/2 = 8.25m \quad (5.3)$$

The result is almost double the sensing range supported by the hardware.

Furthermore, from the line: for x in range(0, 2): the reading of the sensor is done 3 times in a row, thus the implementation of the moving average filter.

The last lines of code ensure that when data reading is initiated, it would pass the value to only one of the readings, avoiding data collision.

```
if PIN == ECHOF:
    sensorF_data=distance

if PIN == ECHOR:
    sensorR_data=distance

if PIN==ECHOL:
    sensorL_data=distance
```

5.3.4 Movements

The expected movements of a differential drive robot was covered in section ???. In this subsection, a description is conducted on the pre made functions governing the behaviour of the physical system. It is important to mention

cont and explan PID implement

At the completion of this chapter, five different functions concerned with the movements, have been used.

The most basic function is the **stop()**, which as its name suggests, halts the motion of the device

```
def stop():
    GPIO.output(StepPinForward1, GPIO.LOW)
    GPIO.output(StepPinForward2, GPIO.LOW)
    GPIO.output(StepPinBackward1, GPIO.LOW)
    GPIO.output(StepPinBackward2, GPIO.LOW)
```

All the movement enabling pins will be set to low, which prompts the device to halt any ongoing movements.

The second function is the forward motion.

```
def forward(SPEED):
    print "FORWARD"
    GPIO.output(StepPinForward1, GPIO.HIGH)
    GPIO.output(StepPinForward2, GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
```

From the code snippet, it can be seen that when the forward function is executed, both of the motors start moving in the same direction and the speed is determined by the variable SPEED. It is important to mention that small fluctuations in the speed of each wheel will cause curved motion instead of the desired forward linear motion, as discussed in section ?? . Thus appropriate control needs to be applied in order to maintain equal speed in both wheels. How the speed is determined will be further discussed in subsection ??: main loop.

ref to model

PUT ENCODERS - PID
ETC HERE

The third function is the reverse, or what is essentially the same as the forward function only that the triggered pins are the backward ones. As in the forward movements, appropriate control is needed to maintain backward linear motion.

```
def reverse(SPEED):
    print "REVERSE"
    GPIO.output(StepPinBackward1, GPIO.HIGH)
    GPIO.output(StepPinBackward2, GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
```

The two movements: forward and backward are the essential functions for the robot's linear motion.

Next we have the turning functions left() and right(). It was decided that the turning functions should be made that the vehicle takes the least amount of space to turn. As it was explained in section ?? ,in a differential drive if the velocities of each wheel are the same but in a different direction, the robot will rotate around its middle point of the wheel axis, which essentially becomes the turning radius.

Below are the two functions needed for turning.

```
def right(turningtime, SPEED):
```

Model

```

    print "RIGHT"
    GPIO.output(StepPinBackward1 , GPIO.HIGH)
    GPIO.output(StepPinForward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    sleep(turningtime)
    GPIO.output(StepPinBackward1 , GPIO.LOW)
    GPIO.output(StepPinForward2 , GPIO.LOW)

def left(turningtime ,SPEED):
    print "LEFT"
    GPIO.output(StepPinForward1 , GPIO.HIGH)
    GPIO.output(StepPinBackward2 , GPIO.HIGH)
    PWML.start(SPEED)
    PWMR.start(SPEED)
    sleep(turningtime)
    GPIO.output(StepPinForward1 , GPIO.LOW)
    GPIO.output(StepPinBackward2 , GPIO.LOW)

```

As we can see both turns, right and left, are made with predetermined factor **turningtime**. This will ensure that the vehicle will not over or under turn. And since a decision was made to mount 3 ultrasonic sensors, the turning ratio on 90 degrees was found adequate. Alteration of the turningtime will allow the device to turn to any desired amount of degrees, however a requirement of more than three ultrasonic sensors and specific positioning might be necessary.

5.3.5 Main loop

In this subsection the main loop, initiated when the device starts, is analysed.

```

while True:
    readsensor(ECHOF)

    if sensorF_data > 200:
        forward(MAXSPEED)

    if 200 > sensorF_data > 100:
        forward(MEDSPEED)

    if 100 > sensorF_data > 20:
        forward(MINSPEED)

    if 20 > sensorF_data:

```

```

stop ()
readsensor (ECHOR)
readsensor (ECHOL)

while sensorR_data<15 and sensorL_data<15:
    reverse (MINSPEED)
    readsensor (ECHOR)
    readsensor (ECHOL)

if sensorR_data>sensorL_data== True:
    right (1,MINSPEED)
    break

if sensorL_data>sensorR_data==True:
    left (1,MINSPEED)
    break

```

The loop is initiated with reading of the front sensor **readsensor(ECHOF)**. Any further action is determined, based on the reading taken from it. If the distance to the closest object is over 2m the vehicle will start moving in a forward direction with full speed. When the distance drops down between 200 cm and 100 cm, the vehicle speed will be reduced to medium speed in order to prepare for a potential turning. Minimal speed is initiated when the distance from the front sensor drops down between 100cm and 20cm, allowing the robot to perform an instant stop. When the distance to the closet object falls under 20 cm,the vehicle will stop and will decide how to proceed based on 3 possible cases. The ultrasonic sensors on the left and right will estimate the distance to the closest object on both sides.

- Case 1
If both sides fall under 15 cm of the closest object, the robot will perform reverse motion and constantly recheck if space becomes available on either side. Once that is true, the robot will perform appropriate turn.
- Case 2
If one or both of the side sensors register distance to the closest object more than 15 cm, it will turn in the direction of the larger available space.
- Case 3
If the robot performs a turn with a minimal acceptable distance reading, it will re-evaluate the possible turning options.

After the turning phase the loop will go back to the beginning and start all over again.

The loop is made as simple as possible to avoid any complications while running the code.

Chapter 6

Conclusion

In case you have questions, comments, suggestions or have found a bug, please do not hesitate to contact me. You can find my contact details below.

Jesper Kjær Nielsen
jkn@es.aau.dk
<http://kom.aau.dk/~jkn>
Fredrik Bajers Vej 7
9220 Aalborg Ø

Appendix A

Appendix A name

Here is the first appendix