Differential Drive Robot with Obstacle Avoidance

- Subtitle -

Project Report

Group Name/Number

Aalborg University
Electronics and IT

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

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: 41

Date of Completion:
December 19, 2016

Abstract:

Here is the abstract

Contents

Todo list

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.

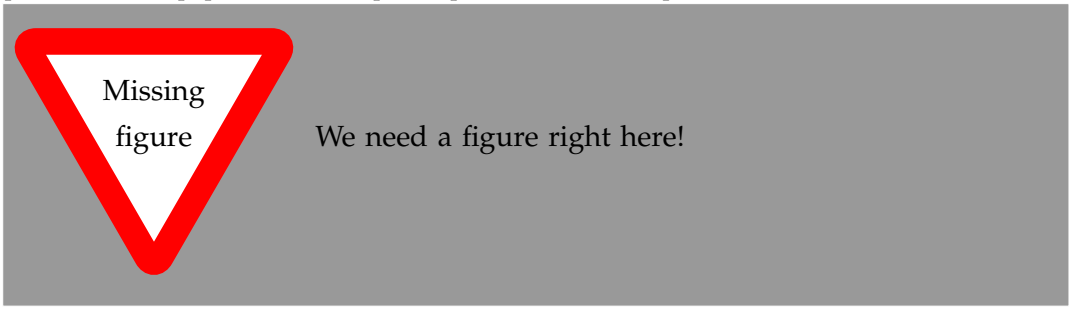
Is it possible to add a subsubparagraph?

Chapter 2

Chapter 2 name

Here is chapter 2. If you want to leearn more about LaTeX 2_ε, have a look at [**Madsen2010**], [**Oetiker2010**] and [**Mittelbach2005**].

I think this word is misspelled



Chapter 3

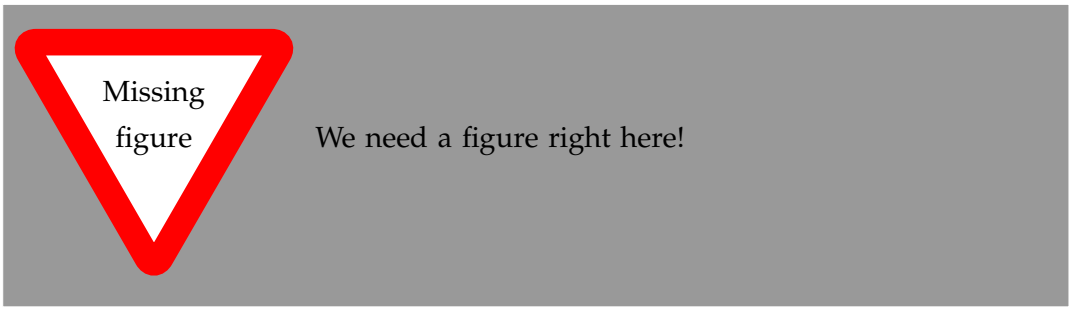
Modeling

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($\boldsymbol{T_e}$) 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($\boldsymbol{K_t}$) as evident in equation 3.1.

$$T_e = IK_t \tag{3.1}$$

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)

$$E_b = \omega K_b \tag{3.2}$$

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.

$$K_t = K_b = K \tag{3.3}$$

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($\boldsymbol{E_b}$). 3.4

$$V = RI + L\frac{dI}{dt} + E_b \tag{3.4}$$

reference to figure

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

$$T_m = T_e - T_b \tag{3.5}$$

Using Newton's second law for rotational motion and substituting from equation3.1, we can rewrite equation 3.5 as:

$$J\dot{\omega} = KI - b\omega \tag{3.6}$$

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:

$$V = RI + L\frac{dI}{dt} + K\omega \tag{3.7}$$

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.

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

$$\Downarrow$$

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

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

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

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)

$$\frac{dI}{dt} = \frac{1}{L}(V - RI - K\omega) \tag{3.11}$$
$$\frac{d\omega}{dt} = \frac{1}{J}(KI - b\omega)$$

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.

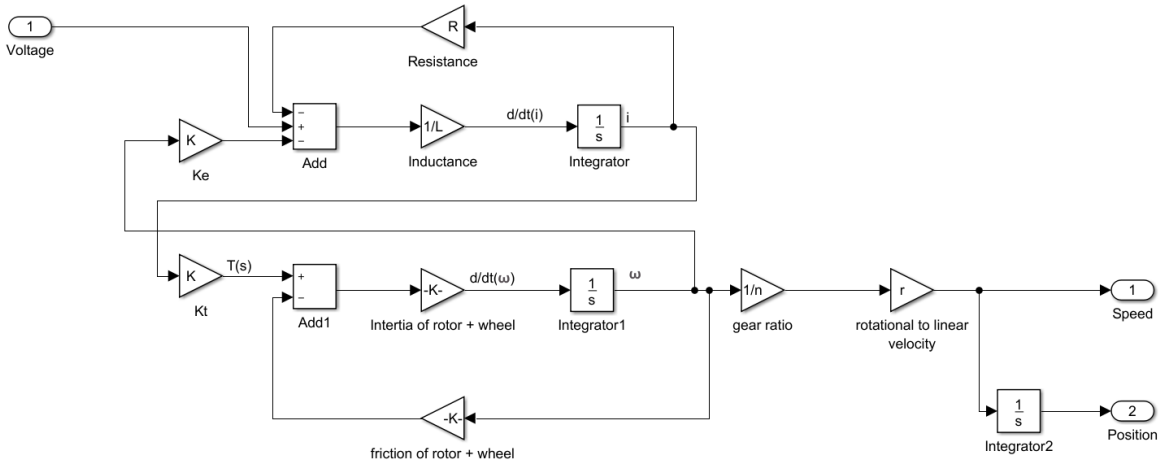


Figure 3.1: DC Motor Block Diagram

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)

$$v = r\omega \tag{3.12}$$

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 .

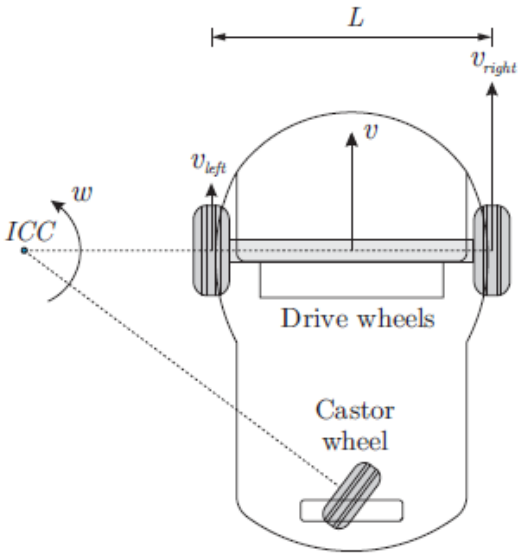


Figure 3.2: Differential drive overview

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.

$$v = \frac{v_r + v_l}{2} \tag{3.13}$$

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)

$$W = \frac{v_r - v_l}{l} \tag{3.14}$$

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

$$v = WD \tag{3.15}$$

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

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

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.

- $v_r = \mathbf{0}$ or $v_l = \mathbf{0}$
  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.

$$q = (x, y, \theta) \tag{3.17}$$

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.

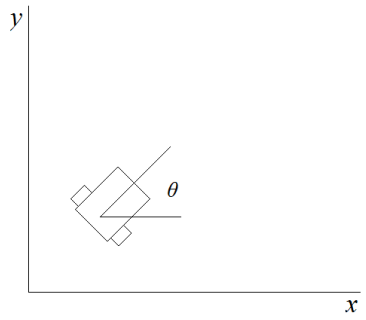


Figure 3.3: Robot representation in Cartesian coordinates

We can picture the constrains 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.

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

Equation 3.18 is a common nonholonomic constraint in mobile robotics. reference to the caltech report

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

$$\begin{aligned} \dot{x} &= vcos(\theta) \\ \dot{y} &= vsin(\theta) \\ \dot{\theta} &= \omega \end{aligned} \tag{3.19}$$

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 results in the kinematics equations for locomotion of a differential drive.

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

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.

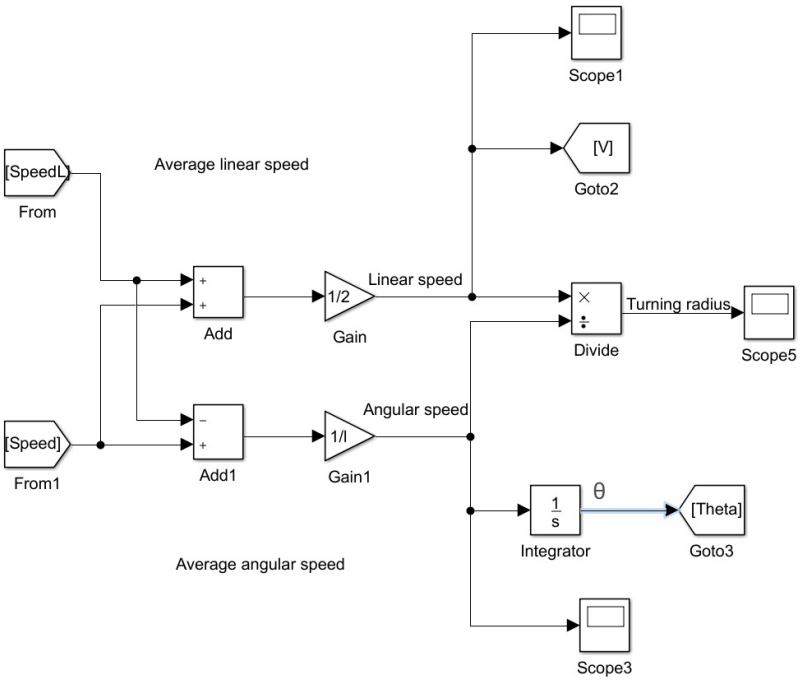


Figure 3.4: Differential drive kinematics

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 it's angle through time, where the angle (θ) is obtain from integrating $\dot{\theta}$, which itself is the angular speed of the robot.

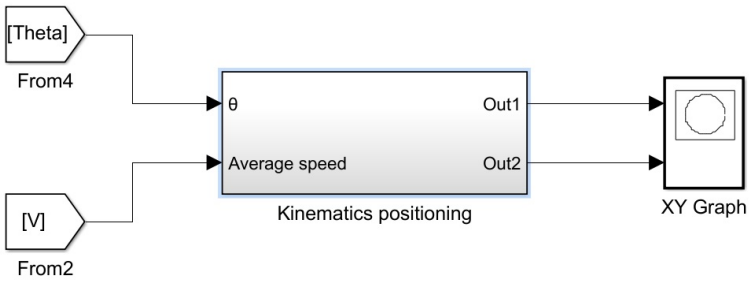


Figure 3.5: Positioning subsystem

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.

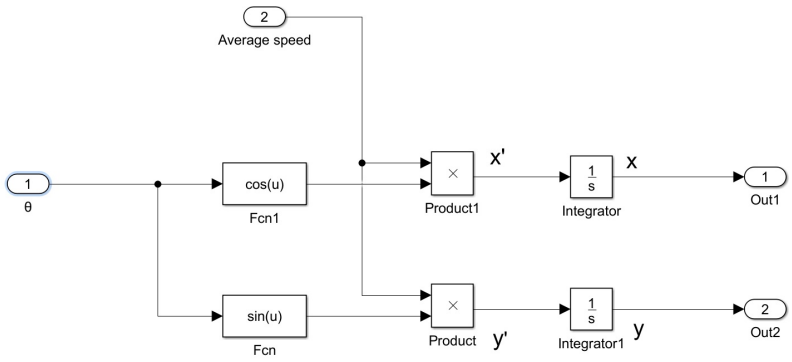


Figure 3.6: Positioning subsystem

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.

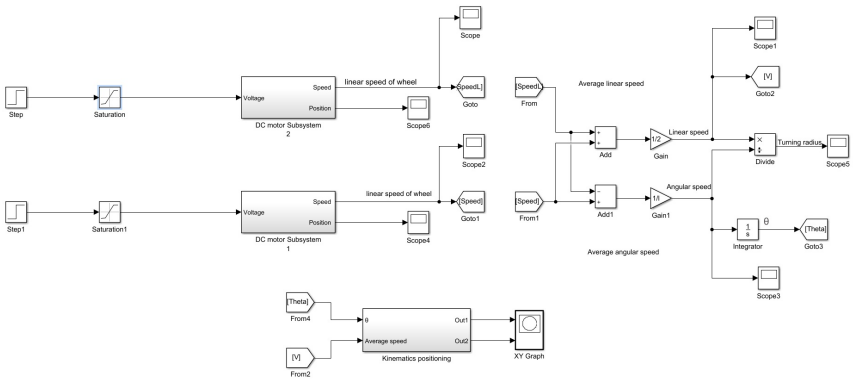


Figure 3.7: Complete system model

In figure 3.7, 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 3.7. 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.

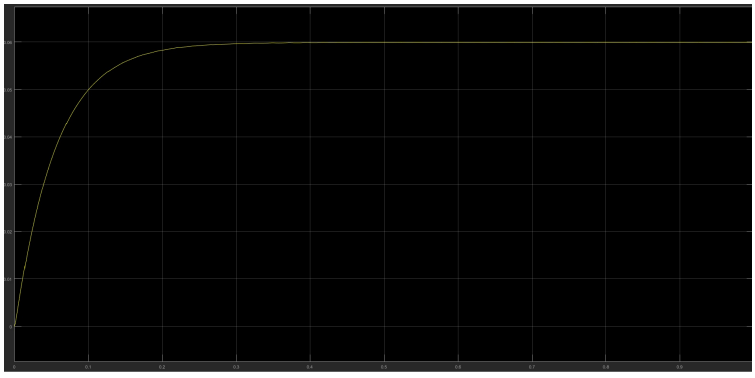


Figure 3.8: Step response of DC motor

Figure 3.8 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.

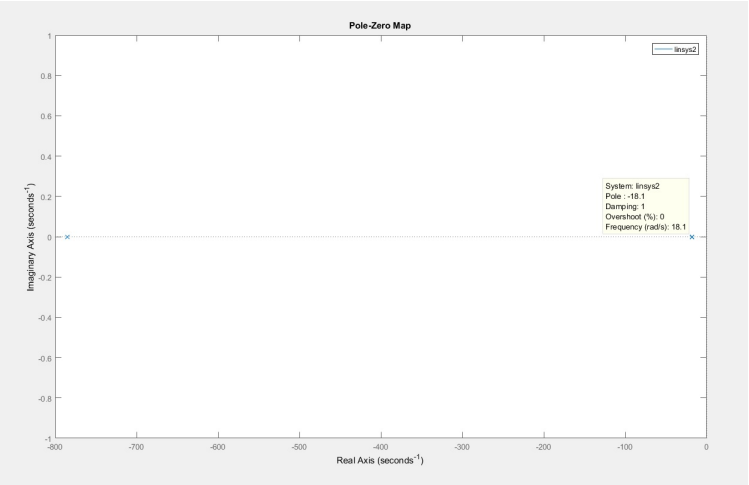


Figure 3.9: Pole-Zero map of DC motor

From figure 3.9, 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 3.2 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.

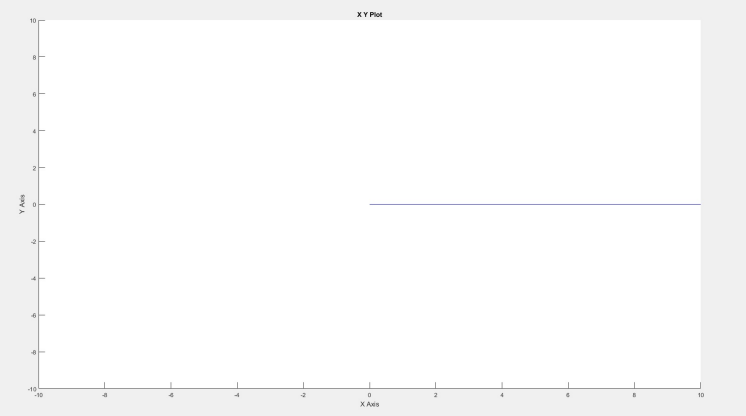


Figure 3.10: XY-graph for equal velocities

In figure 3.10, when both wheel velocities match ($v_r = v_l$), the trajectory the robot partakes is a straight line.

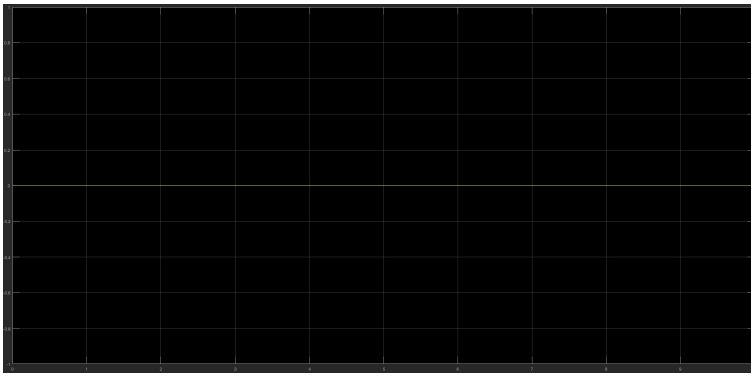


Figure 3.11: Turning speed for equal velocities

As to be expected, the turning speed in figure 3.11 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).

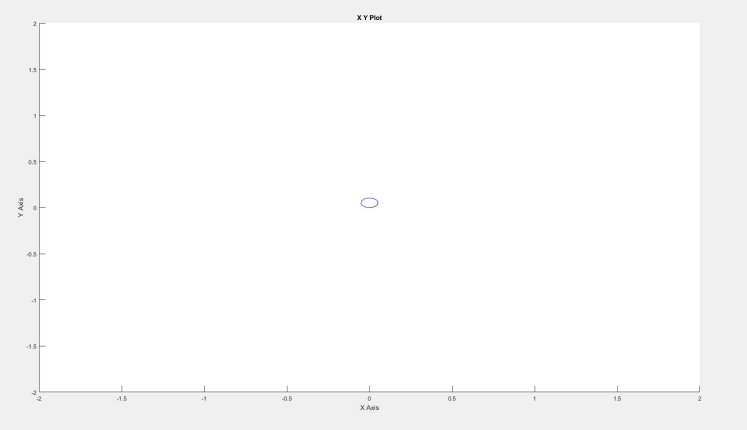


Figure 3.12: XY-graph for zero velocity left wheel

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 $\frac{l}{2}$.

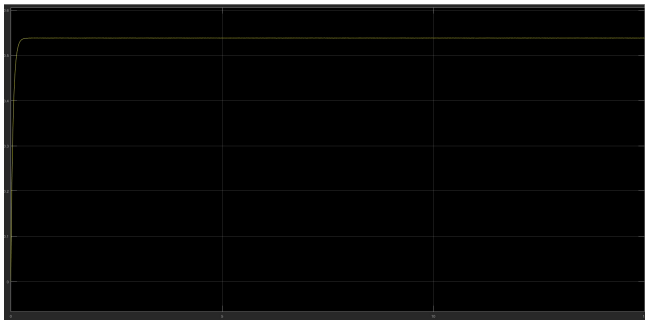


Figure 3.13: Turning speed for zero velocity left wheel

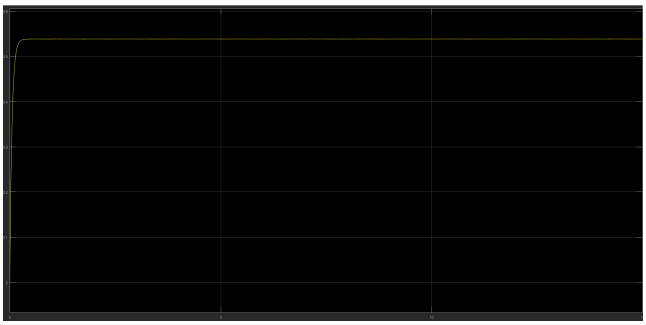


Figure 3.14: Turning radius for zero velocity left wheel

Chapter 4

Hardware

Hardware in the device In the following chapter we will see what hardware we have used to produce this device. First we decided on what platform we will be working on.

4.1 Single board computer

Since we were allready familiar with the Raspberry Pi singleboard computer we decided that it is best to continue with the system we are allready know how to operate. At first we used the Raspberry Pi 2 model B but in concideration of the power supply and performance we swapped it out with a Raspberry Zero.

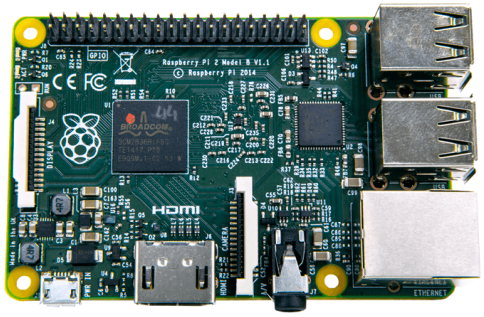


Figure 4.1: Raspberry Pi 2B

We chose the Rasperry Pi Zero over the Raspberry Pi 2B because of the lesser power consumption and the size. Performance of the two computers are similar. Zero has less RAM but in our project it does not have a significant impact.

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

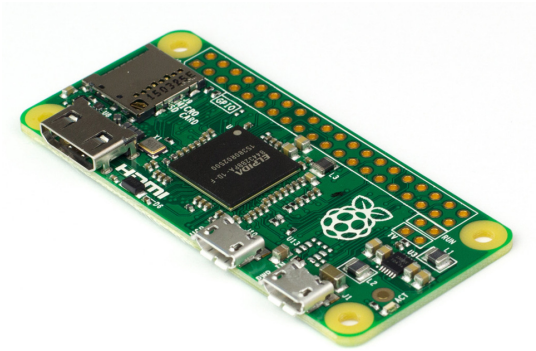


Figure 4.2: Raspberry Pi Zero

Composite video and reset headers

4.2 Distance measuring

Since our device would be avoiding obstacles on the way it is fitted with distance measuring sensors. We have decided to use ultra sonic sensors. Concidering the cost and the performance we are looking for we eventually settled on the HC-SR04 ultra sonic sensor. In our device we have used three of those sensors in orded to cover the front side of the veichle.

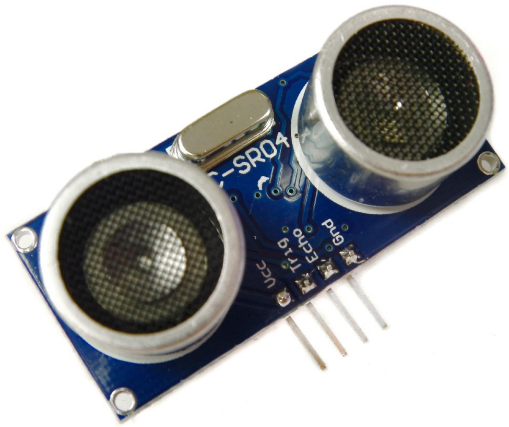


Figure 4.3: Ultra sonic sensor HC-SR04

4.3 DC motors and the chassie

We are using a premade complect of two DC motors and the chassie with the wheels. Each of the motors are connected to the wheels directly.

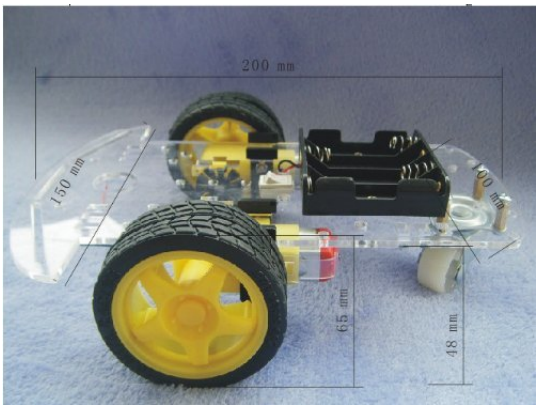


Figure 4.4: Chassie and the two DC motors with a power supply

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

(CITE: https://elektronik-lavpris.dk/p129700/robo0002-smart-robot-car-chassis-kit-with-speed-encoder-and-battery-box/) The two motors are identical and are used to move the device and to steer aswell

4.4 Motor driver

At the beginning of the project we used a L9110S DC Stepper Motor Driver H-Bridge for controlling the movement of the wheels, but as we soon dicovered the suggested driver was not capable of regulating the motor speeds. To controll the speed of the veichle we then swapped it to the L298N driver. 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)

4.5 Speed sensor

The speed of the wheels is measured by the LM393 IR speed sensors.

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

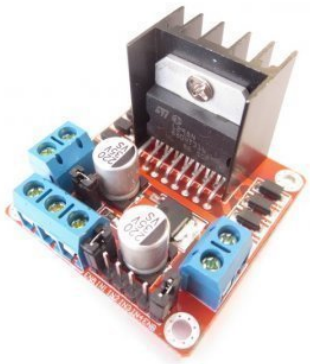


Figure 4.5: The L298N driver

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. (CITE:http://www.banggood.com/LM393-Speed-Sensor-Detection-Speed-Module-For-Arduino-p-970033.html?currency=AUDutm$_source$ = $myshoppingutm_medium = cpcutm_content = saulutm_campaign = xie - AU$)

4.6 Wi-Fi dongle

For the monitoring of the device and connectivity we have used a Edimax EW-7811Un Wi-Fi adapter.

4.7 Assambly

In this section we will be looking more closely what has been connected to what and how. From the figure below we can see the layout of the whole system.

In the middele you can see the Raspberry Pi. Since the software used to make this schematic did not have the Raspberry Pi Zero layout we used model 2 instead. Regarding our project id does not make a difference, since the pins are the same as Zero.

On the right side of the system you can see three ultrasonic sensors. The ultrasonic sensors are connected to the RP Pi. The Vcc and GND pins all go to the Raspberry so that the power for the sensors is taken from the Raspberry itself. The triger pins all go to the same Raspberry pin. That means that when you trigger

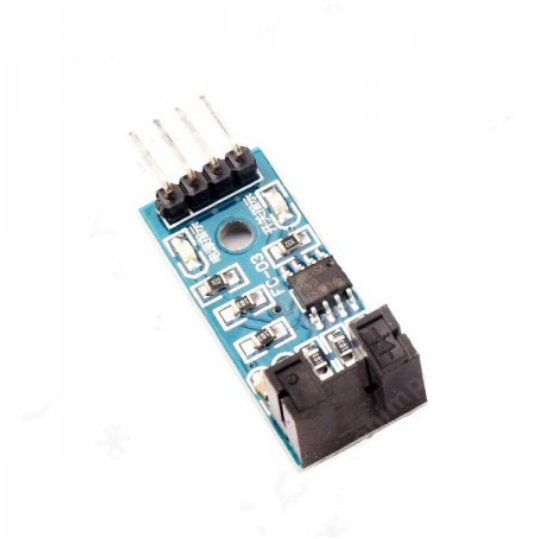


Figure 4.6: The LM393 IR speed sensor

one of the sensors all of them will emit sonic impulse. The trigers are all on the same pin to preserv more pins on the Raspberry, it does not matter if they are on one or seperate pins. The echo pins of each of the sensors go into seperate ports on the Raspberry, each of them has a voltage divider connected to the GND as well.

In the left bottom you can see the driver with the motors and a seperate battery pack. The battery pack is connected to the driver to give power to the motors and the driver itself aswell. Driver is connected to the Raspberry by 6 pins. Four of the pins are for the directions of the both motors. Two of the pins what are the enable pins are used for controlling the speed of the motors by PWM. The motors are connecter directly to the driver by two wires. In the top left you can see the two encoders what are used for monitoring the speed of the wheels.

For the Raspberry we have a external power source what is not pictured on the figure.

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

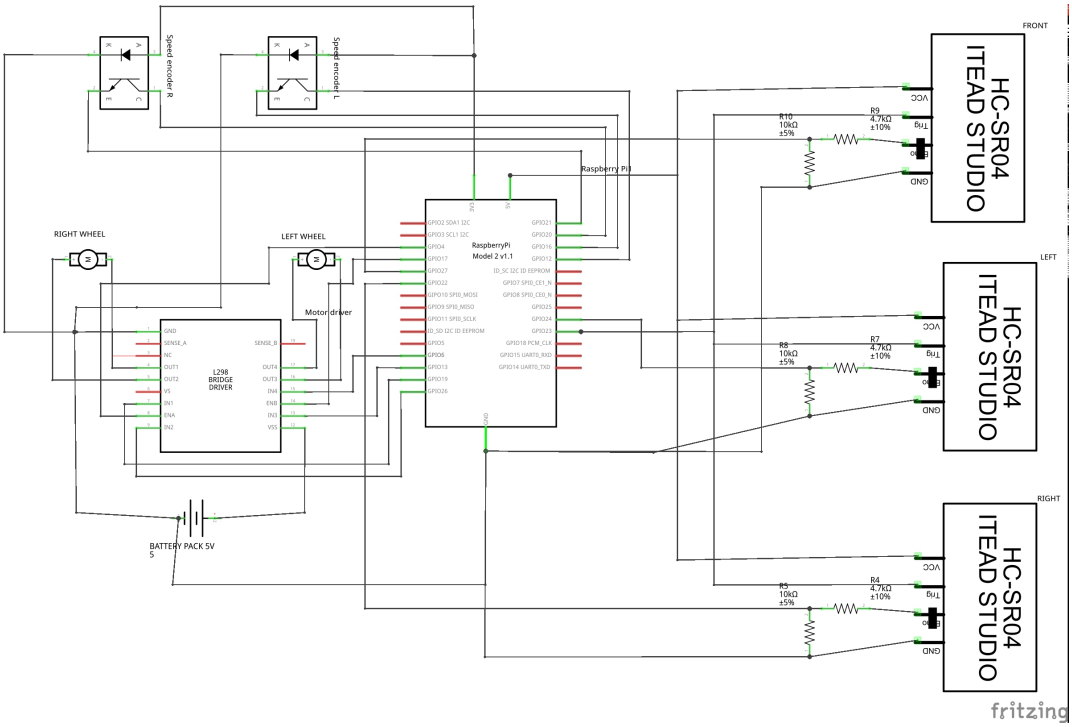


Figure 4.8: Schematics of the device

Chapter 5

Development

Hardware testing

5.1 Ultrasonic sensors

First of the hardware we started testing the ultrasonic sensors. For each of the sensors we built a voltage divider seen on the following figure.

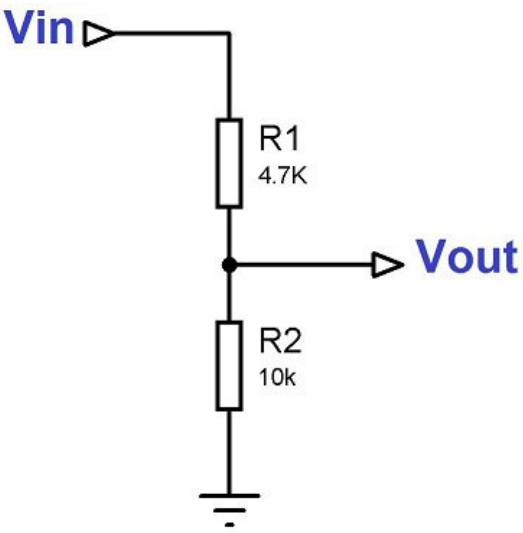


Figure 5.1: Voltage divider

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

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

We tried all the sensors out seperately by connecting them 1 by 1 to the raspberry and ran the test code.

```
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 "W8ing on da 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 correctly while connected seperately so we moved on to try them out all of them at the same time. For that we connected all of the ultrasonic sensors to the raspberry and tried them out. For testing all of them we included some filtering aswell, because while taking every reading we saw that some of the values where off the chart high. High values was most probably due to the noise or just some random jittering. Our filter is made to take three readings at the time and then calculate the average.

```
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)
```

As you can see from the code above we made sure that every reading takes exactly 0.05 seconds. This will help us make every cycle evenly long and we can predict the total time that the program runs the whole cycle. After applying the filter we saw that the readings became alot more percise and consistent. Therefore the time for the sensor reading loop becomes 3*0,05s=0,15s.

5.2 DC motors

For testing the dc motors we drove the motors in forward gear and in backward gear through the driver we are useing. Below you can see the script we used to conduct the testing of the motors.

```
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 "backwarding 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 you can see from the code it runs one of the two motors first forward for 5 seconds and then backwards for 5 seconds. For the second motor we just changed the pin numbers(StepPinForward and StepPinBackward).

Further more we added the speed control via PWM. 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 veichle.

5.3 Raspberry configuration and software

In this section we will look closer what has been done to the raspberry and how it works.

5.3.1 Raspberry setup

First we took the Raspberry Pi Zero and installed the Raspbian operating system. Then we enabled all the GPIO-s,ssh and got the latest Python. Since the Raspberry Pi Zero has only one usb port we decided to operate the device via WiFi. The earlier mentioned WiFi dongle is connected to the USB port and then we connected it to the provided WiFi network. For programming on the Raspberry we used tmux multitab tool and nano text editor via SSH from a Linux machine,

5.3.2 Software

On the veichle itself we are using our self developed software to control the device. It is quite simple code in a sense. Code is written in Python and is using few external librarys.

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

To be exact we are using only 3 external librarys as you can see from the sniplet from above. The sys module:This module provides a number of functions and variables that can be used to manipulate different parts of the Python runtime environment. (CITATION FROM:http://effbot.org/librarybook/sys.htm) The time module: This module provides a number of functions to deal with dates and the time within a day. It's 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. (CITATION FROM:http://effbot.org/librarybook/time.htm) And the RPi.GPIO module is for functions what are connected to the GPIO pins.

Next we have the overall setup of the pins and the variables where you can see all the different values we are using in the code. For further details you can see the setup below.

```
#PIN numbers
LetfPWM=16
RightPWM=20
StepPinForward1=26
StepPinBackward1=19
StepPinForward2=13
StepPinBackward2=6
ECHOF=4
ECHOL=27
ECHOR=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(ECHOF, GPIO.IN)
GPIO.setup(ECHOL, GPIO.IN)
GPIO.setup(ECHOR, GPIO.IN)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(LetfPWM, 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

Since we are using ultrasonic sensors for reading the distances from the veichle to the closest ocstacle, we must ensure that we dont get too big noise from the sensors. For that purpouse we are using a little filter in the part where we read the data from the sensors. Below you can see the function made for the data gathering.

```
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
```

Our data reading from the sensors is quite straight forward. First the TRIG pin sends out a sonic impuls from one of the sensors. At the moment the impulse is triggered the software makes a timestamp. Second timestamp is done when the impulse returns to the sensor.

As you can see from the line: for x in range(0, 2): the reading of the sensor is done 3 times in a row. This ensures that we dont get a random values from the sensor but that we get atleast some filtering on it. Our filter basically takes the average from the 3 readings and then passes it on to the main cycle.

The distance reading is designed so that every loop takes the same amout of time to run it. It is done by taking a timestamp in the beginning of the loop:$read_time_start1 = time.time().Thencalculatingwiththeread_time_start1ontheline : time.sleep(0.05 - (time.time() - read_time_start1)).Thiswillensurethateverycycleisexactly0.05secondslong.Thetimeiscalculatedsothatwewouldsur$
$0.05s = 16.5andsinceitsgoingbackandforththenwedivideitwith2andwegetthemaxrangeforthetimelimitation16.5$
$8.25mThiswillgiveusdoublethemaxrangeprovidedbythehardwareitself.$

As you can see from the lines:

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

        if PIN == ECHOR:
                sensorR_data=distance

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

Every time the data reading is initiated the loop will pass on only one of the readings.

5.3.4 Movements

For different movements we have composed premade functions. There are all together 5 different functions for the movements.

First we have the stop() function what does exactly like the name says it does, it stops any motion of the device. Below you can see the stop function.

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

This just turns all the movement enabling pins to low and the device stops if it had in any motion before.

As a second function we have the forward motion. From below you can see the function itself.

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

As you can see from the code 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. How the speed is determined we will see from further on when we get to the main loop of the veichle.

Third we have the reverse what essentially is the same as the forward only that the pins triggered are the backward pins. As you can see from the code below.

```
def reverse(SPEED):
```

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

These two movements: forward and backward are essential to make the device move.

Next we have the turning functions left() and right(). We decided that the turning functions should be made that the veichle takes the least amount of space to turn. Since we are using a differential drive we can just spin one wheel in one way and the other in the other direction so that the middle point of the wheel axis stais still. Below you can see the two functions needed for turning.

```
def right(turningtime,SPEED):
        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 turning right and left are made with predetermined factor turningtime. This will ensure us that the veichle will not over or under turn. And since we decided to mount only 3 sensors we settled with the turning ratio on 90 degrees. While altering the turningtime we can make the device turn any other amount of degrees. This can be implemented when there are more then 3 sensors and they are located differently.

5.3.5 Main loop

In this section we will look closer at the main loop what will be initiated when the device starts. Before we looked at all the functions seperately. Below you can see the mail loop what is constantly ran in the device.

```
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
```

Here you can see that the loop is initiated with reading of the front sensor:readsensor(ECHOF). And any further action is then determined what the result is. If the distance is over 2m the veichle will start moving forward with full speed. When the distance is in between 200cm and 100 cm the vheicle will start moving with the medium speed. Minimal speed is initiated when the distance from the front sensor is between 100cm and 20cm. When the distance is under 20 cm the veichle will stop and will decide what to do next.

After stoping there are 3 options. First it will decide how far are the left and right boarders. If the sides are closer then 15 cm the device will back up and check again until one of the sides has more room then 15cm. Then the vheicle will turn left or right dependent where there is more room.

If the sides are not close(over 15cm) then the veichle will turn to the side what has more space. It there is not enough space in front after the turn then it will turn again until there is. 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