

EPO-4

"KITT"

Autonomous Driving Challenge



Group B-09

4589734 - Zep Kleijweg
4458478 - Antony Peetoom
4536762 - Victor Schonk
4599136 - Jurgen Wervers

May 23, 2018



EPO-4


"KIT" Autonomous Driving Challenge

by

Group B-09

For the bachelor project EPO-4
at Delft University of Technology.

Abstract

This report describes the design process to compete in the midterm challenge of project EPO-4, which is part of the Bachelor Electrical Engineering at Delft University of Technology. The goal of this project is to design and implement a control system to let a car stop autonomously, at a given distance in front of a wall using ultrasonic distance sensors. After determining the limitations of the car and the communication system between the car and the computer that perform the calculations, a control system is designed and tested. After initial tests and fine tuning of some system parameters, the car performed quite well during the midterm challenge. 

Contents

Abstract	ii
1 Introduction	1
2 Specifications	2
3 System Design	3
3.1 Strategies	3
4 Measurements	4
4.1 System Delays	4
4.2 Ultrasonic Sensors	4
4.3 Video Analysis	6
5 System Implementation	8
5.0.1 The acceleration state	8
5.0.2 The deceleration state	8
5.0.3 The roll state	8
5.0.4 The stop state	8
5.0.5 The error state	9
5.1 Optimization of the System	9
6 Results	10
7 Conclusion	12
7.1 Discussion	12
Bibliography	13
A Midterm challenge code	14
B Measurements	18
B.1 Video Analysis	18

1

Introduction

The EPO-4 project 'Autonomous driving challenge' is part of the curriculum of the second year of the Electrical Engineering bachelor at Delft University of Technology. This report describes the development of the midterm challenge by group B-09.

During the midterm challenge a start position between 3 meter and 5 meter from a wall is given and a stop distance 30cm to 50cm from the wall is given. The goal is then to get as fast as possible and as accurately as possible from the starting position to the stop position. During the challenge the car has two attempts to perform as good as possible from a short distance, where the starting position is closer to 3 meter. The car also has two attempts to perform as good as possible from a long distance, where the starting position is closer to 5 meter. The car is not allowed to have any overshoot during any attempt meaning it may not move backwards at any time during the challenge.

To accomplish that the car can complete the challenge both fast and accurate a control system was designed using Matlab [1]. The design of the control system used to perform the challenge can be found in chapter 3. The measurement methods and measurement results, to help in determining the control system can be found in chapter 4. And the final performance and results of the midterm challenge can be found in chapter 6



2

Specifications

In order to fulfill the requirements of the midterm challenge as described in chapter 1 as good as possible, a control system needs to be designed and implemented to control the car. The car has a variable speed and can drive both forwards and backwards, with a top speed of around 20km/h . The driving speed of the car depends on the battery voltage, with a lower voltage resulting in a lower speed. The battery voltage will be $17 - 20\text{V}$ during the midterm challenge. Steering will not be necessary for the midterm challenge, so will not be discussed. The computations to control the car will be performed on a computer, which can communicate with the car via a Bluetooth connection. The maximum distance between the computer and the car during the midterm challenge will be 6 meters. The car is equipped with two ultrasonic distance sensors that operate at 40kHz , one on the front left corner and one on the front right corner, both pointing forward. These sensors measure the distance to objects in front of the car, and this information can be sent to the computer over the Bluetooth connection. This can be used as an input for the control system that is running on the computer. Since the computing of the commands for the car happens on the computer and not on the car itself, there is a certain delay between when the car measures a certain distance and when the control system acts accordingly. The key problems to overcome in order to successfully complete the midterm challenge are to accurately measure the acceleration and deceleration of the car for certain speed inputs, in order to know when to start braking. These values will also depend on the battery voltage. The second problem is to accurately determine all the delays in the system in order to be able to account for them in the control system.





3

System Design

To make the car stop at the given distance, two modules on the PC should be designed: a module for communicating with the car, receiving measured distances and transmitting commands for driving, and a controller which decides what the car should do based on the feedback from the distance sensors.

At the end of the car, an off-the-shelf microcontroller handles the reading out of the ultrasonic distance sensors and battery voltage and it controls the servo motors for accelerating and steering. Its design is completely predetermined. For the communication with the car, the Bluetooth link implementation is set up both at the side of the car as on the PC in Matlab [1], giving a function for directly controlling the car.



This function 'EPOCommunications()' can be used with the following arguments [3]:

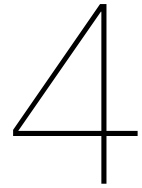
- 'open' to open a specified COM port for Bluetooth communication and 'close' to close it again.
- 'transmit' to transmit a command to the microcontroller, telling either to control the car or to report back.
 - 'M<x>' to drive, where <x> is an argument between 135 and 165, 150 being stationary.
 - 'D<x>' to steer, where <x> is an argument between 100 and 200, 150 being straight forward.
 - 'S' to report back a complete status.
 - 'Sd' to report back the measured distance. Because this is the only value that should be monitored during driving, this argument is almost always used instead of a full status request.

The communication module should consist of functions to easily invoke the function 'EPOCommunications()' and to make sure the micro controller has enough time to finish the command it got before getting a new one. The controller module then uses the communication module to receive the measured distance as input and to tell the car what to do according to the implemented strategy.

3.1. Strategies



To complete the midterm challenge both accurately and fast, a certain strategy has to be determined. Mainly because of the fact that the ultrasonic sensors will prove to provide very inaccurate data when the car is traveling at a high velocity a bit of speed was sacrificed to make the system more accurate. This was done by decelerating the car when it is relatively close to the wall and let it roll out until the car is at the set distance from the wall. Since the car is moving more slowly when it is close to the wall using this strategy, the measured distance from the sensors is more accurate. This strategy is implemented using the states described in Chapter 5. In order to determine when the car has to decelerate the first time, the acceleration and deceleration of the car will have to be measured, as explained in Chapter 4.3.



Measurements

First, a stable Bluetooth connection needed to be established between the computer and the car. After that the proper functioning of the car was checked: steering, moving forward and backward, requesting the status, etc. When everything seemed to work correctly the limitations of the system could be measured. The maximum distance between the car and the computer during the midterm challenge will be around 6 meters, so at this distance the communication should still work fine. The car still received commands sent by the computer correctly at a distance of 10 meters, so this should not pose any problem.



4.1. System Delays

In the Bluetooth communication between the car and computer is a certain delay: the signal is received slightly later than it was sent. There is also a delay in the processing time of both the computer and the car: the micro controller in the car does not immediately get the information when the Bluetooth receiver receives the signal. Then there is the processing delay of both the micro controller and the computer, both have to perform calculations before they can respond to the given input. Since the ultrasonic sensors do not measure continuously but only once every $66ms$, there is a chance that the last distance measurement is for example 60 ms old. This is not really a delay, but does cause an uncertain inaccuracy in the distance measurement.

It is very hard to measure all those delays separately, since it is difficult to make a distinction between where a certain part of the total delay came from. Because of this, certain delays will have to be estimated based on delays that can be measured. What can be measured is the delay between requesting a distance measurement (or full status update) on the computer, and receiving it from the car. This delay contains a processing delay of the request, a communication delay, the processing delay from the micro controller, another communication delay and the processing delay from the computer, in that order. Since this delay varies, it was measured 10.000 times to get an impression of how it is distributed. A histogram of these measurements is displayed in Figure 4.1.

The mean of this measured data set is $58.8ms$, but there are a few outliers with values of $125ms$. This delay can however only be used as an indication of the general magnitude of the delays in the system, since in practice only one communication delay and one processing delay are present before a signal is interpreted. The actual transmission time over the Bluetooth connection is not a significant part of the delay, the propagation time of the signal can be calculated as follows: $t = \frac{\text{maximum distance}}{\text{speed of electromagnetic waves}}$



$\frac{6}{3 \cdot 10^8} = 20ns$. This delay would contribute to the total delay twice in the measurements above, both on the way to the car and on the way back to the computer, but even then this is not significant compared to the mean delay. The propagation time becomes even shorter when there is less distance between the car and computer.

4.2. Ultrasonic Sensors

The ultrasonic sensors are used to measure how far away the wall is from the car. In order to design a functioning control system, the accuracy of those measurements is important and needs to be deter-

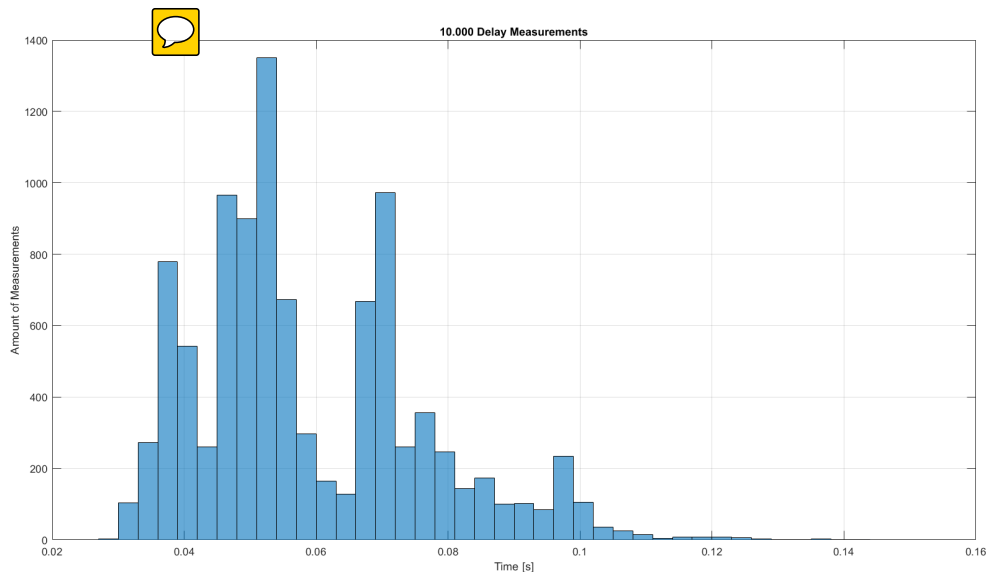


Figure 4.1: Delay Measurements



mined. Both the static and the dynamic accuracy need to be determined, meaning both when the car is stopped and when it is driving. The static accuracy says something about the general accuracy of the sensors, and when this is combined with the dynamic measurements something can be said about the impact of the driving speed on the distance measurements.

The static accuracy was determined by pointing the car towards the wall and comparing the distance indicated by the sensors with the actual distance measured with a measuring tape. This proved to have a deviation below 2cm for distances between 25 – 250cm. If the car is closer than 25cm the measured distance becomes larger than the actual distance, probably because the sensors measure indirect reflections of the transmitted signal because the direct reflection arrives too soon. If the distance is larger than 250cm the measured distance occasionally jumped to a significantly smaller or larger value than the actual distance. This is possibly due to other reflections in the environment being more significant than the reflection from the wall resulting in a lower measurement, or the reflection of the wall being too weak resulting in a larger measurement due to different reflections.

During hard braking the car nose dives, which causes the sensors to point more towards the ground instead of forward. Measurements at 30, 40, 60, 80 and 100cm showed that this does not affect the accuracy of the distance measurement. At a larger distance, however, the measured distance becomes larger than the actual distance. This is probably because the ground then reflects the signal, making the propagated distance longer than the actual distance. This should not pose a problem, since braking hard at those distance is never done. If it does prove to be a problem, it can be avoided by ignoring or not requesting the sensor data during hard braking.

For the dynamic measurements it is difficult to do a precise measurement, because besides the ultrasonic sensors there is no good way to measure and log the current distance between the car and the wall. The solution was to put a measuring tape on the floor and drive next to this, to at least get a general idea. From driving towards the wall at considerable speed it became clear that the sensor data displayed on the display of the car already lagged considerably when compared to the actual distance. As described in the next section, the acceleration of the car also needed to be measured. This acceleration can be used to calculate the speed of the car if the time since the take off is known, which can be measured. With the speed of the car and the delays in the system as measured before, the distance that the sensors lag behind the actual distance can be computed using the following equation: $D = v * t$ with distance D, speed v and time since take off t. This distance that the sensors lag behind can then be accounted for in the design of the control system.



4.3. Video Analysis

To determine the acceleration and deceleration of the car, video analysis was performed. This analysis method was chosen over using others, especially using the ultrasound sensors, **because it is an easy and reliable analysis tool.** Especially the ultrasound sensors would have been very useful for this analysis, but due to the difficulty in coding without delays and the inconsistencies of the sensors this was not an option. **The video tracking was successful even though the video settings were not optimal.** A program called Tracker [2] was used to extract data from the video footage. This was done by tracking certain points on the car throughout the video and comparing this movement to a reference length and position. Which resulted in a usable data set to extract acceleration and deceleration estimates from. This was done by using Matlab [1], in which a linear approximation was made in the parts where the speed was relatively constant, of which the graphs can be seen in figure 4.2 and 4.3. The fitted lines were made with respect to the speed, due to the fact that the speed curve has a smaller error than the acceleration curve, as can be seen in figure 4.2. The acceleration can be easily obtained by differentiating the speed curve. Since the car drove from right to left in the video used for the analysis, the two plots are reversed in time meaning that it should be read from right to left. Code used for these plots and the calculations of values can be found in appendix B.1. This resulted in acceleration values between 1.2m/s^2 and 1.3m/s^2 and deceleration values between 6.1m/s^2 and 6.7m/s^2 . The difference is due to different tracking points behaving differently. These values were tuned later on, this will further be explained in chapter 5.1.

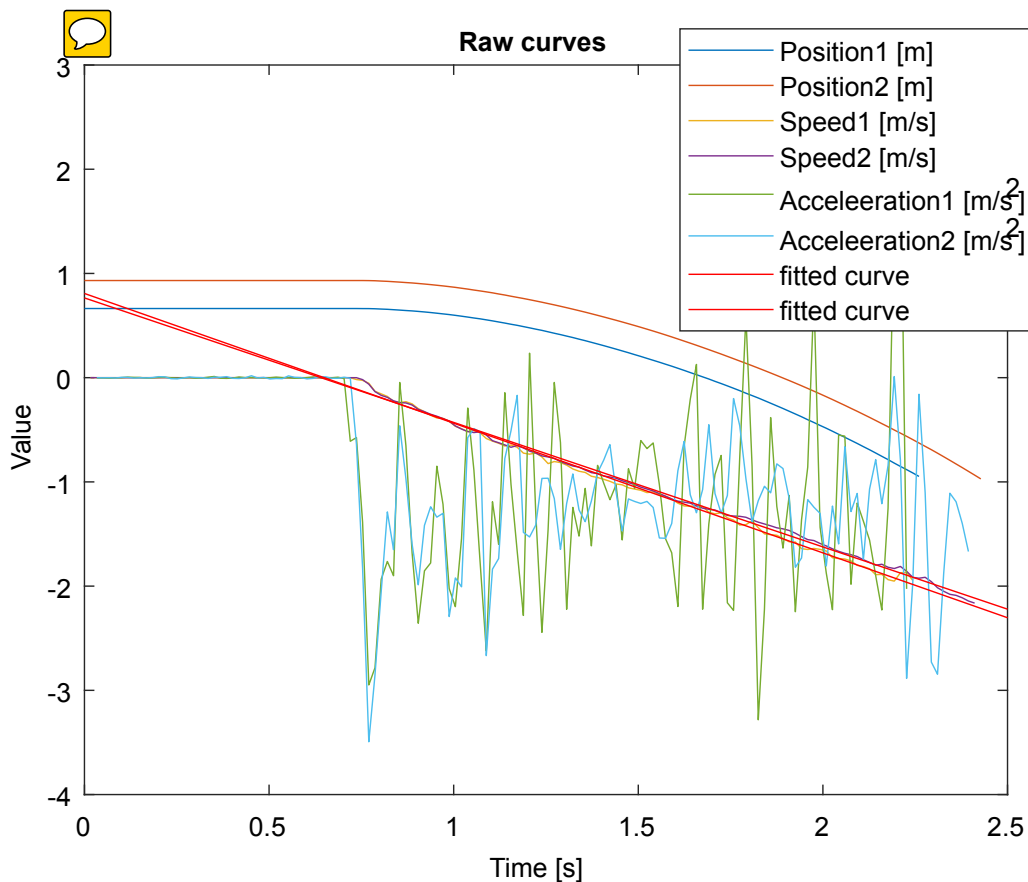


Figure 4.2: Position, speed and acceleration curve during acceleration. The two curves are due to the two different tracking points.

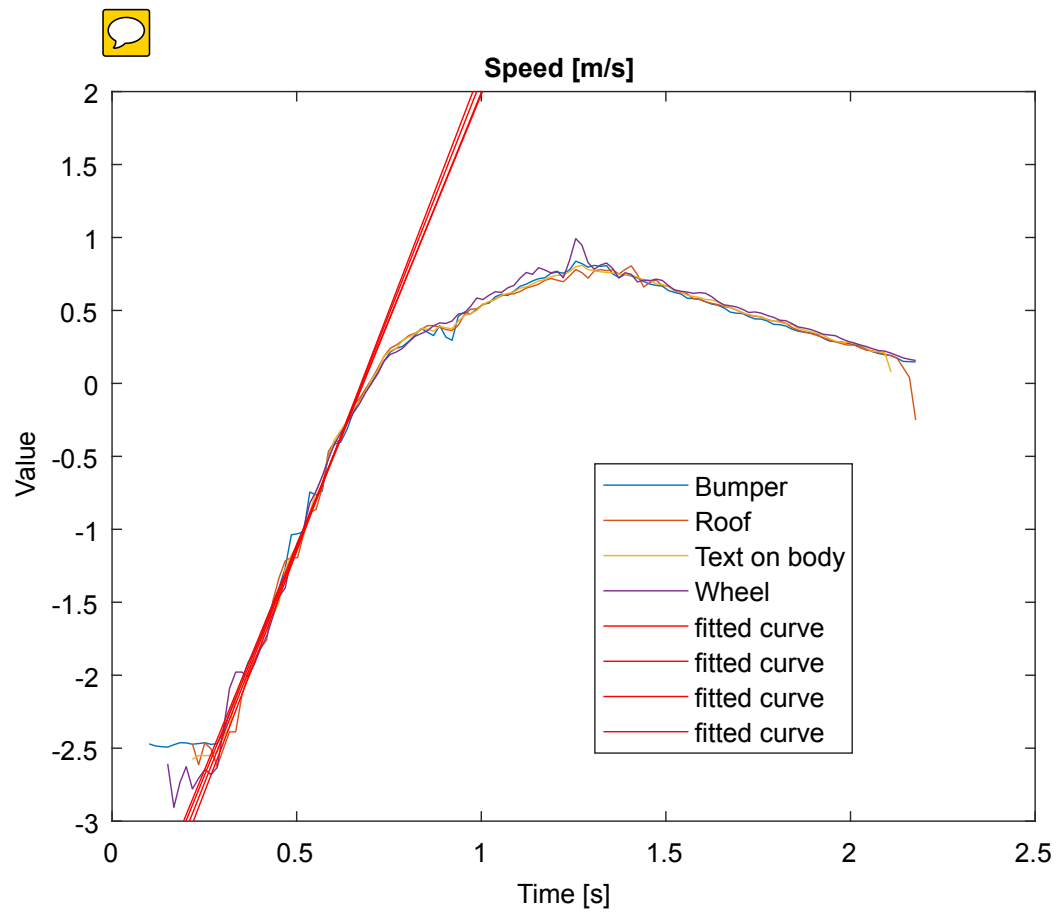


Figure 4.3: Speed curve during deceleration of several tracking points.



System Implementation

The implementation of the discussed design consists of the following parts:

- A setup function, initiating communication by opening a certain COM port.
- A function for resetting the car to initial state and terminating communication. This function is to be used as an emergency brake and at the end of a script to secure a properly closed COM port for Bluetooth.
- Several functions for quickly accessing the given functions for communication, using a pause where necessary to prevent queueing of commands at the microcontroller.
- A script which uses the above functions to drive the car, deciding when to read out the sensors and what the car should do based on its feedback.

This script is the controller of the whole system. It can be found in appendix [A](#). It has several states it goes through before terminating itself with the car at, hopefully, the right distance from the wall:

5.0.1. The acceleration state

A timer is started and the car is told to accelerate as fast as possible. Multiplying the time passed and the approximated linear acceleration gives the speed of the car. Once the car is closer to the wall than *1.5meter* plus the distance it would traverse with the current speed in the approximated delay time discussed in section [4.1](#), the controller exits this state and moves on to the next one.

5.0.2. The deceleration state

The controller uses the cars current speed and its approximated linear deceleration to calculate how long to go in full reverse to slow down to a predefined speed of 0.4 m/s. After this period of time, the car is instructed to continue its way at this speed by giving the associated speed argument of 155. Then, this feedbackless state is succeeded by the next one.

5.0.3. The roll state

Now, the car is able to approach the wall with higher precision thanks to its lower speed. Again, a distance margin is calculated by multiplying the above mentioned approximate delay with the current speed, to ensure the car does not stop too late due to the delay. There is 8 cm between the sensors and the bumper of the car, which has to be added to the distance from the wall to stop at, along with a constant to cancel out a set deviation measured from multiple tests. Once the resulting distance is spanned, the car is instructed to reverse for a very short time as a means of braking, before the last state is accessed.

5.0.4. The stop state

The car is told to stop and the Bluetooth link is securely terminated by closing the COM port.

5.0.5. The error state

Then there is an extra state, the error state, which should not be necessary if everything goes as planned. The system arrives at this state if there is any error while executing the commands from the previous states, to ensure it does not get stuck in the state with the error. In the error state, the car is told to stop and the COM port is closed.

5.1. Optimization of the System

At last some optimization of the system had to be done. This was mainly performed by trial and error. Due to time limitations remeasuring parameters to a more accurate degree was not an option, resulting in simply performing a lot of test runs and changing parameters which were obviously related to certain problems. For example if the initial braking point was too late the delay time should be turned up, because that should happen at a fixed distance with an addition of the distance driven in the delay time. Or the acceleration should be turned up, which results in the code thinking that the speed is higher than before thus resulting in an earlier braking point. Also if the braking at the initial point was too aggressive, either the deceleration or the final rolling speed should be adjusted. This resulted in quite some testing, but eventually turned into a simplification of the system to the degree that only a couple variables were used to describe the system accurately enough. One of those was a fixed distance error, another one was the delay time from sensing a distance to the code being able to use it and the last one was the speed after the initial braking.

6

Results

The results obtained during the midterm challenge can be found in Figure 6.1.

During the first challenge a starting distance of 4.1m and a desired stop distance of 40cm from the wall were given. During the first attempt of the first challenge a deviation of 1cm from the desired stopping distance from the wall was achieved. The time was 5 seconds, which is rather slow, this was caused by the fact that the stopwatch was started at the same time as the Matlab script was executed, after the run command was given to Matlab it took a while until the car actually started moving. During the second try of the first challenge a deviation of 12cm from the desired stop distance from the wall was achieved. This is a rather high deviation which seldom occurred during the testing of the car after the final version of the scripts were written. The time of the second attempt of the first challenge was 3.6 seconds, which is a lot better than the time of the first challenge. This is because of the fact that the stopwatch started from the moment the car started moving instead of from the moment the Matlab script was executed.



During the second challenge a starting distance of 3.4m and a desired stopping distance of 40cm from the wall were given. During the first try of the second challenge a deviation of 1cm from the desired stopping distance from the wall was achieved. The time of the first attempt of the second challenge was 3.2 which is a bit slower than the desired 3 seconds. During the second attempt of the second challenge a deviation of 2cm from the desired stop distance from the wall was achieved. The time of the second attempt of the second challenge was 3.3 seconds which is also a bit slower than the desired 3 seconds.



Figure 6.1: Results of the midterm challenge

Og: 43

EPO4 Midterm Challenge 2018

Groep:

Bg

Niet aanwezige leden: X

Challenge 1: startafstand (ca 5 m) 4.1 m

Gewenste stopafstand: 40 cm

	poging 1	poging 2
afstand	41 cm	28 cm
tijd	5.0 s	3.6 s
overshoot	Ja / Nee	Ja / Nee

Opmerkingen:

Challenge 2: startafstand (ca 3 m) 3.4 m

Gewenste stopafstand: 40 cm

	poging 1	poging 2
afstand	39 cm	42 cm
tijd	3.2 s	3.3 s
overshoot	Ja / Nee	Ja / Nee

Opmerkingen:

Conclusion

In the end the results of the midterm challenge were quite good. During both challenges a deviation of only 1cm was achieved. The car was during all challenges a bit slower than desired. This may have been caused by the fact that the battery voltage of the car was a bit on the lower side and because of the fact that the system was mainly designed to be very accurate while not too much attention went to the measurements and optimization of the speed of the system.

7.1. Discussion

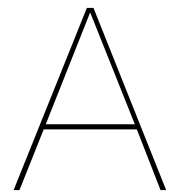
A bit more effort should have been put into making sure the challenge would have been completed within the time limit. A solution would have been to fine tune the initial braking point of the car more, to make sure a shorter distance would have to be travelled at a low speed.

When heavily decelerating, there tends to be nose dive. The effect on the distance sensors was measured to be nihil for short distances. In the final design, the state where the car heavily decelerates is not using feedback from the sensors, making it safe to say the consequences of this effect are negligible.

The final resulting system was not robust. It was optimized for this challenge of stopping at the right distance by adjusting constants using trial and error. For the next step, autonomously navigating, this design is not fit. This is mainly because the distance sensors are too slow for the speed that was achieved. In particular, one constant had to be added which was not accounted for by the model of linear acceleration and deceleration. This constant, which adjusted the distance at which the final state should be activated, was actually a variable depending on begin and end distance and had to be manually inserted prior to each challenge.

Bibliography

- [1] Matlab (scientific analysis tool). URL <https://nl.mathworks.com/products/matlab.html>.
- [2] Video analysis software. URL <https://physlets.org/tracker/>.
- [3] Jorge Martinez Alle-Jan van der Veen. *EE2L21 Project EPO-4: Autonomous driving challenge*, 2018.



Midterm challenge code

```
1 cport = 6; % com port to use
2
3 dbumper = 0.08; % distance between sensors and bumper
4
5 sdist = 0.30; % distance to stop at in meters
6 vroll = 0.4; % speed to approach the final position [m/s] 0.46 @ 154 | 0.56 @ 155
7 vmin = 0.15; % speed to stop decelerating at
8 del = 0.3;%0.2; % delay margin
9 derr = -0.00;%0.6-sdist; % error margin, to avoid overshoot and oscilation
10
11 v = 0; % begin speed
12 a165 = 1.217; % acceleration with motor at full power
13 a135 = -6.57; % acceleration with motor full power backwards while moving forwards
14 % assumed that with motor at 154 with low speeds there is no acceleration,
15 % nor deceleration
16
17 %%% Compensation for unexplained overshoot, linearly proportional to final
18 %%% distance
19 tmp = linspace(0.23,0.23,40); % add 0.05 for short begin dist and substr 0.05 for
    long begin dist
20 dos = tmp(int8((sdist-0.29)*100)); %dbumper
21 clear tmp;
22
23 dm = 3; % 3 for 165 | 2 for 150 | 1 for 135
24 a = [a135 0 a165]; % acceleration for different motor states
25 r = 0;
26 r_val = 15;
27
28 try
29 % openCom(cport);
30
31 tic;
32 drive(165);
33 while 1
34     v = v+toc*a(dm); % current speed of the vehicle
35     tic;
36     dmar = del*v; % distance to compensate the delay
37     r = r+1;
38     if r > r_val
39         if mean(sensors())/100 < 1.5+dmar % currently the minimal sensor value is
            used mean is another option
40             break;
41         end
42     end
43 end
```

```

42     r = 0;
43     end
44 end
45 drive(135);
46 pause(abs((v-vroll)/a(1)));
47 drive(155);
48 % pause(0.04);
49 v = vroll;
50 dmar = del*v;
51 while 1
52     r = r+1;
53     if r>r_val
54         if mean(sensors())/100 < sdist + derr + dmar + dos + dbumper
55             drive(138);
56             break;
57         end
58         r = 0;
59     end
60 end
61 pause(0.2);
62 closeCom();
63 catch
64     pause(0.2);
65     drive(150);
66     pause(0.2);
67     closeCom();
68 end

```

Listing A.1: Code used to perform the midterm challenge. Functions used within this script can be found below.

```

1 function openCom(inputArg1)
2 %OPENCOM Summary of this function goes here
3 % Detailed explanation goes here
4
5     s1 = '\\.\COM0';
6     s1(8) = int2str(inputArg1);
7     EPOCommunications('open',s1);
8     %sprintf('\\.\COM%d',inputArg1)
9
10 end

```

Listing A.2: Code used to open a certain COM port.

```

1 function closeCom()
2 %CLOSECOM This function can be used to close the COM-port.
3 % The closing of the COM-port implemented in a function might be used to
4 % improve robustness.
5
6     EPOCommunications('close');
7
8 end

```

Listing A.3: Code used to close the COM ports.

```

1 function out = sensors()
2 %SENSORS Summary of this function goes here
3 % Detailed explanation goes here
4 % Returns value in cm's
5
6     str = EPOCommunications('transmit','Sd');
7     st = strsplit(str,'\n');
8     clear str;

```

```

9     s1 = cell2mat(st(1));
10    s2 = cell2mat(st(2));
11    outputArg1 = str2num(s1(4:end));
12    outputArg2 = str2num(s2(4:end));
13
14    % s1 = strsplit(string(st(1)),'L');
15    % s2 = strsplit(string(st(2)),'R');
16    % s1 = strsplit(s1(2),'\n');
17    % s2 = strsplit(s2(2),'\n');
18    % outputArg1 = str2double(s1(1));
19    % outputArg2 = str2double(s2(1));
20    out = [outputArg1, outputArg2];
21
22 end

```

Listing A.4: Code used to read the sensor values in an easy way.

```

1 function drive(inputArg1)
2 %drive This function takes in input between 135 and 165, to drive the car
3 %forward or backwards.
4 % An input value of 150 corresponds to no driving action. A value of 135
5 % corresponds to full speed reverse. And a value of 165 corresponds to
6 % full speed forward. This function also limits its inputs to the range
7 % of 135 to 165.
8
9     if inputArg1 > 165
10        dr = 165;
11    elseif inputArg1 < 135
12        dr = 135;
13    else
14        dr = inputArg1;
15    end
16    EPOCommunications('transmit',sprintf('M%d',dr));
17
18 end

```

Listing A.5: Code used to give motor commands.

```

1 function steer(inputArg)
2 %steer This function takes an input from 100 to 200. To steer the car.
3 % An input value of 100 corresponds to the wheels fully turned to the
4 % right. A value of 150 corresponds to the wheels centered. And a value
5 % of 200 corresponds to the wheels turned all the way to the left. This
6 % function also limits its input to the range of 100 to 200.
7
8     if inputArg > 200
9        st = 200;
10    elseif inputArg < 100
11        st = 100;
12    else
13        st = inputArg;
14    end
15    EPOCommunications('transmit',sprintf('D%d',st));
16 end

```

Listing A.6: Code used to steer. This was not used for the midterm challenge

```

1 function [r_val] = stop()
2 %stop This function stops the car from driving and straightens the wheels.
3 %After that it returns the status in the command window.
4 %
5

```

```
6 EPOCommunications( 'transmit' , 'M150' );  
7 pause(0.05);  
8 EPOCommunications( 'transmit' , 'D150' );  
9 pause(0.05);  
10 r_val = EPOCommunications( 'transmit' , 'S' );  
11 closeCom();  
12  
13 end
```

Listing A.7: Code used to stop the car and close the COM ports.

B

Measurements

B.1. Video Analysis

```
1 % error between the two points is 0.05 meter at its maximum.
2 % point(x,1) time [s] 1:end
3 % point(x,2) position [m] 1:end
4 % point(x,3) speed [m/s] 2:end-1
5 % point(x,4) acceleration [m/s^2] 3:end-2
6
7 % data point 45 is the first to change
8
9 load('vid_2.mat');
10 plot(point1(:,1),point1(:,2));
11 hold on;
12 plot(point2(:,1),point2(:,2));
13 plot(point1(:,1),point1(:,3));
14 plot(point2(:,1),point2(:,3));
15 plot(point1(:,1),point1(:,4));
16 plot(point2(:,1),point2(:,4));
17 xlabel('Time[s]');
18 ylabel('Value');
19 title('Raw curves');
20 legend('Position1[m]','Position2[m]','Speed1[m/s]','Speed2[m/s]','
    Acceleration1[m/s^2]','Acceleration2[m/s^2]');
21
22 % figure;
23 % plot(point1(:,1),smooth(point1(:,2)));
24 % hold on;
25 % plot(point2(:,1),smooth(point2(:,2)));
26 % plot(point1(:,1),smooth(point1(:,3)));
27 % plot(point2(:,1),smooth(point2(:,3)));
28 % plot(point1(:,1),smooth(point1(:,4)));
29 % plot(point2(:,1),smooth(point2(:,4)));
30 % xlabel('Time [s]');
31 % ylabel('Value');
32 % title('Smooth curves');
33 % legend('Position1 [m]','Position2 [m]','Speed1 [m/s]','Speed2 [m/s]','
    Acceleration1 [m/s^2]','Acceleration2 [m/s^2]');
34
35 fv1 = fit(point1(45:end-1,1),point1(45:end-1,3),'poly1');
36 fv2 = fit(point2(45:end-1,1),point2(45:end-1,3),'poly1');
37 % figure;
38 plot(fv1);
39 a1 = fv1(2)-fv1(1);
```



```

40 a1r = mean(point1(45:end-2,4));
41 a2 = fv2(2)-fv2(1);
42 a2r = mean(point2(45:end-2,4));
43 a = max(abs([a1,a1r,a2,a2r]));
44 % average of these 3 is rounded up to an acceleration of 1.3 m/s^2
45 plot(fv2);
46
47 xlabel('Time[s]');
48 ylabel('Value');
49
50
51 % polyfit
52 % polyval
53 % fit
54 %%%%%%%%%%

```

Listing B.1: Code used to calculate acceleration values.

```

1 load('vid3_0_50.mat');
2
3 %%% Raw position curve
4 figure;
5 item = 2;
6 plot(bump(:,1),bump(:,item));
7 hold on;
8 plot(dak(:,1),dak(:,item));
9 plot(text_body(:,1),text_body(:,item));
10 plot(wiel(:,1),wiel(:,item));
11 title('Position[m]');
12 xlabel('Time[s]');
13 legend('Bumper','Roof','Text_on_body','Wheel');
14
15 %%% Raw speed curve
16 figure;
17 item = 3;
18 plot(bump(:,1),bump(:,item));
19 hold on;
20 plot(dak(:,1),dak(:,item));
21 plot(text_body(:,1),text_body(:,item));
22 plot(wiel(:,1),wiel(:,item));
23 title('Speed[m/s]');
24 xlabel('Time[s]');
25 legend('Bumper','Roof','Text_on_body','Wheel');
26
27 f1 = fit(bump(13:37,1),bump(13:37,3),'poly1');
28 a1 = f1(1)-f1(0);
29 f2 = fit(dak(7:30,1),dak(7:30,3),'poly1');
30 a2 = f2(1)-f2(0);
31 f3 = fit(text_body(7:30,1),text_body(7:30,3),'poly1');
32 a3 = f3(1)-f3(0);
33 f4 = fit(wiel(10:34,1),wiel(10:34,3),'poly1');
34 a4 = f4(1)-f4(0);
35 amax = max([a1,a2,a3,a4]);
36 amean = mean([a1,a2,a3,a4]);
37 plot(f1);
38 plot(f2);
39 plot(f3);
40 plot(f4);
41
42 title('Speed[m/s]');
43 xlabel('Time[s]');
44

```

```

45 ylim([-3 2]);
46
47 %%% Smooth speed curve
48 %% figure;
49 %% item = 3;
50 %% plot(bump(:,1),smooth(bump(:, item)));
51 %% hold on;
52 %% plot(dak(:,1),smooth(dak(:, item)));
53 %% plot(text_body(:,1),smooth(text_body(:, item)));
54 %% plot(wiel(:,1),smooth(wiel(:, item)));
55 %% title('Smooth speed [m/s]');
56 %% xlabel('Time [s]');
57 %% legend('Bumper','Roof','Text on body','Wheel');
58 %%
59 %% f1 = fit(bump(13:37,1),bump(13:37,3),'poly1');
60 %% a1 = f1(1)-f1(0);
61 %% f2 = fit(dak(7:30,1),dak(7:30,3),'poly1');
62 %% a2 = f2(1)-f2(0);
63 %% f3 = fit(text_body(7:30,1),text_body(7:30,3),'poly1');
64 %% a3 = f3(1)-f3(0);
65 %% f4 = fit(wiel(10:34,1),wiel(10:34,3),'poly1');
66 %% a4 = f4(1)-f4(0);
67 %% plot(f1);
68 %% plot(f2);
69 %% plot(f3);
70 %% plot(f4);
71 %%
72 %% ylim([-3 2]);
73
74 % start braking
75 % bump      13
76 % dak       7
77 % text_body  7
78 % wiel      10
79
80 % 0-crossings      value      next value
81 % bump      37   -0.0589    0.0046
82 % dak       30   -0.0378    0.0299
83 % text_body  30   -0.0600    0.0288
84 % wiel      34   -0.0643    0.0019
85
86 %% figure;
87 %% item = 3;
88 %% plot(bump(:, item));
89 %% hold on;
90 %% plot(dak(:, item));
91 %% plot(text_body(:, item));
92 %% plot(wiel(:, item));
93
94 %%% Raw acceleration curve
95 figure;
96 item = 4;
97 plot(bump(:,1),bump(:, item));
98 hold on;
99 plot(dak(:,1),dak(:, item));
100 plot(text_body(:,1),text_body(:, item));
101 plot(wiel(:,1),wiel(:, item));
102 title('Acceleration [m/s^2]');
103 xlabel('Time [s]');
104 legend('Bumper','Roof','Text on body','Wheel');

```

Listing B.2: Code used to calculate deceleration values.