EE2L21 EPO-4: Kitt - Final Report

June 18, 2015

Group B6

Martijn Berkers	4223438
Thomas Brasser	4139518
Richard de Jong	4238575
Jeroen van Uffelen	4232690

Delft University of Technology

Abstract

In this report an approach to handle the final challenge given in the EPO-4 project is presented. This includes localization of the car with audio communication, collision detection is done using the data obtained from distance sensors, planning a route with the car and driving it. There are 3 parts of the final challenge, driving from point A to point B, driving from point A to point B with waypoint C, and driving from point A to point B with an obstacle in the route.

Introduction

This report describes the design process of an autonomous driving, wireless charging vehicle. The goal of the design is to make the car drive from point A to point B while avoiding obstacles. To achieve this several problems have to be overcome; wireless communication, object sensing, mapping the location and let the car navigate, devising a control system to steer and drive the car, using audio communication for localization of the car and combining these solutions into a system.

To keep structure the report is divided into several sections. First, the specifications of the car will be determined.
The second section explores localization using audio communication, for this training sequences are discussed. The deconvolution of the measured signal and the reference signal and corresponding peak detection to calculate the Time-difference of arrival(TDOA) between microphones. The localization algorithm used to convert these TDOA's to a position, and the verification of this position.
The third section will discuss the route determination of the car, and the corresponding outputs to the wheels of the car.
The fourth section will discuss the system integration of all these parts and the loop where they are used in, and the GUI used for displaying the position of the car and obstacles.
Lastly a conclusion is constructed about the project and discussion about what could have been done different.

Specifications

Contents

Introduction **2**

Specifications **2**

1 Localization using audio communication **4**
1.1 Training sequence . . . 4
1.2 Reference . . . 4
1.3 Deconvolution . . . 6
1.4 Peak detection . . . 9
1.5 TDOA . . . 11
1.6 Localization . . . 11
1.7 Checking calculated position . . . 12
1.8 Resulting Design . . . 13
1.9 Testing . . . 15
1.9.1 Field testing . . . 16
1.10 Conclusion . . . 16

2 Route planning **17**
2.1 Introduction . . . 17
2.2 Design . . . 17
2.2.1 Specifications and Goals . . . 17
2.2.2 Code . . . 18
2.3 Testing . . . 20
2.3.1 Trial and Error . . . 21
2.3.2 Findings . . . 21
2.4 Conclusion . . . 21

3 Control Loop **22**
3.1 Overview . . . 22
3.2 Check if supercaps are charged. . . . 22
3.3 Drive . . . 22
3.4 Sample . . . 22
3.5 Mapping the car . . . 22

4 Conclusion and Discussion **24**

5 Planning and Teamwork **25**
5.1 Planning . . . 25
5.2 Teamwork . . . 25

A Appendix **28**
A.1 Module 1 . . . 28
A.1.1 Open_com.m . . . 28

1 Localization using audio communication

The goal of this module is using audio beacons to accurately determine the position of the car. An audio beacon is placed on the car, the signal is received with five microphones at pre-determined positions. With this information the position of the car can be determined with the TDOA(Time difference of arrival). This time corresponds to the difference in arrival time between microphones. With microphone pairs the position can be determined. To determine when the "sound" is received a training sequence is used and transmitted using the audio beacon. This training sequence is received at all microphones at different times and at intensities, to filter the training sequence from the noise the channel response is used instead of the measured signal. For the estimation of the channel response a known training sequence has to be used. The deconvolution of the inverse of the training sequence and the measured signal gives channel response. There are several implementations for this because the inverse of the training sequence is not easily calculated. The channel response will ideally have one peak at the point where the measured signal is the same as the training sequence. If comparing the position of the peaks of different microphones the difference of arrival in samples can be found, this is converted to time. In practice the measured signal is not clean, noise and reflections are also measured. To chose the right peak the bad ones need to be filtered. To improve the deconvolution properties of the measured signal, instead of the modelled training sequence the impulse response of the sequence is measured, this can be measured when the beacon is close to the microphone.

1.1 Training sequence

The training sequence that will be transmitted by the audio beacon is a binary sequence which uses "on-off keying". Besides a code word with a maximum of 64 bits, the training sequence is restricted to the parameters in table 1. The code word used is a random sequence of 32 bits. The code word is random because a random sequence has good convolution properties. The setup used to measure the microphones had a sample frequency of 48 kHz. That means that the maximum carrier frequency can be 24 kHz. To be on the safe side our Timer0 parameter is 2, which is 15 kHz.
Because our carrier frequency is 15 kHz and we wanted at least 5 waves in 1 bit, our Timer1 parameter is 4, which is 3.0 kHz. Our Timer3 parameter is 7, which is 8 Hz. This is because there has to be a silent period after the code has been send, to let the reflections die out.

Table 1: Timers frequency configuration table

Timer index	0	1	2	3	4	5	6	7	8	9
Carrier Freq (Timer0) [kHz]	5	10	15	20	25	30	0	0	0	0
Code Freq (Timer1) [kHz]	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	0
Repeat Freq (Timer3) [Hz]	1	2	3	4	5	6	7	8	9	0

1.2 Reference

To have the best deconvolution properties an accurate reference signal has to be used. This is because the modelled signal and the transmitted signal are not equal. Because of this the deconvolution properties are not optimal. To get an accurate reference the signal that leaves

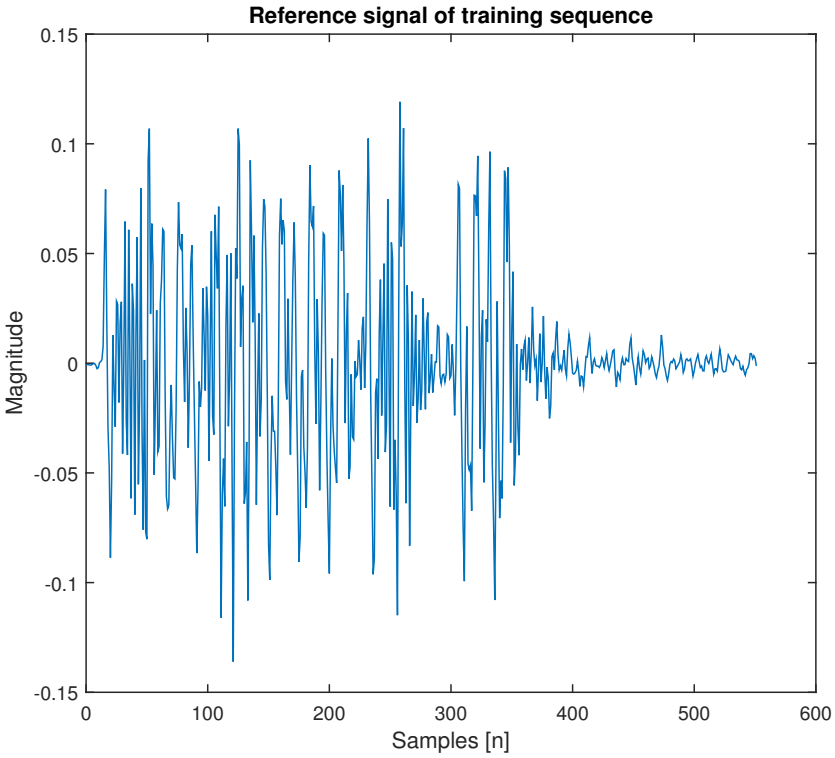

Figure 1: Refsignal

the speaker needs to be measured. At close distances of around 5 cm, the measured signal will be the channel response $y[n] = h[n]$ where $y[n]$ is the measured signal in samples and $h[n]$ the channel response. Each time the training sequence is changed a new reference has to be measured. The reference used was one of the multichannel measurements, because these microphones are the same as the ones used in the final challenge. The signal was manually shortened to only contain the training sequence, the length of this is given by the settings of the training sequence, the settings used give it a length of 512 samples with a sample rate of $48kHz$. Because it is not clear when the signal begins and ends some data points before the sequence and after have been included in the reference. A plot of the reference can be seen in figure 1

1.3 Deconvolution

The deconvolution of the received audio signal by taking the inverse of the reference signal is computationally complicated. There are three alternative methods which can be used for the deconvolution; the first is the calculation of the Toeplitz matrix, the second is using a matched filter and the third is using frequency equalization. The Toeplitz matrix has a size of the length of the measured signal times the length of the impulse response. The impulse response will be length of y minus the reference. Both of these for our usual measurements are around 18000. Taking the inverse of a matrix this size is not a viable option. Because there was still some doubt about which was better, the matched filter or the frequency equalization, it was decided to implement both. As the tests were done the method with the most reliable results will be the one to choose for the final challenge.

The deconvolution function has two inputs, the measured audio signal $y[n]$ and the reference measured training sequence $x[n]$. function will return two estimated channel responses $h[n]$, one for the matched filter and one for the frequency equalization. The matched filter will use the `filter` function in MATLAB, the filter coefficients are determined by the flipped training sequence $x[-n]$. Because this filtered data is not yet yet in the right scale, it is corrected with the magnitude of $x[n]$. The Matched Filter can be written as

$$\hat{h}[n] := y[n] * x[-n] = h[n] * (x[n] * x[-n]) = h[n] * r[n]$$

where $r[n]$ is the autocorrelation of the reference signal. The audio channel response will be smeared, because of the autocorrelation it is expected that there will always be a peak where the signals overlap. Because of this the channel response may be inaccurate but at least the peak will be pronounced and corresponds to the true channel. A plot of the typical impulse response estimated with a matched filter is found in 3. The code of the matched filter can be seen in appendix **??**

An alternative to the Matched Filter is to do frequency equalization. Because the frequency equalization had the best deconvolution properties this is used for the final challenge. $Y(\omega) = H(\omega)X(\omega)$ is derived from $y[n] = h[n] * x[n]$ and hence we can estimate $H(\omega) = Y(\omega)/X(\omega)$. To make this work using the FFT all the sequences need to be of equal length N, where N is the length of y. Since a pointwise division is done in frequency domain the performance will be low for frequencies where $X[k]$ is small. Therefore a threshold ϵ is used to determine which values of $H[k]$ will be set to zero before the Inverse Fast Fourier Transform is applied. The value of epsilon used is calculated in line 10 in Listing 1 after that the indices where X is smaller than ϵ are determined in lines 10-16 and the newly obtained channel estimate is calculated. The entire code can be found in appendix **??**

Listing 1: Calculating Epsilon

```
10 eps = 0.1*max(abs(X)); %Make epsilon depend on X
11 G = X; %make G the same length as X
12 ii = find(abs(X) <= eps);
13 G(ii) = 0;
14 ii = find(abs(X) > eps);
15 G(ii) = 1;
16
```

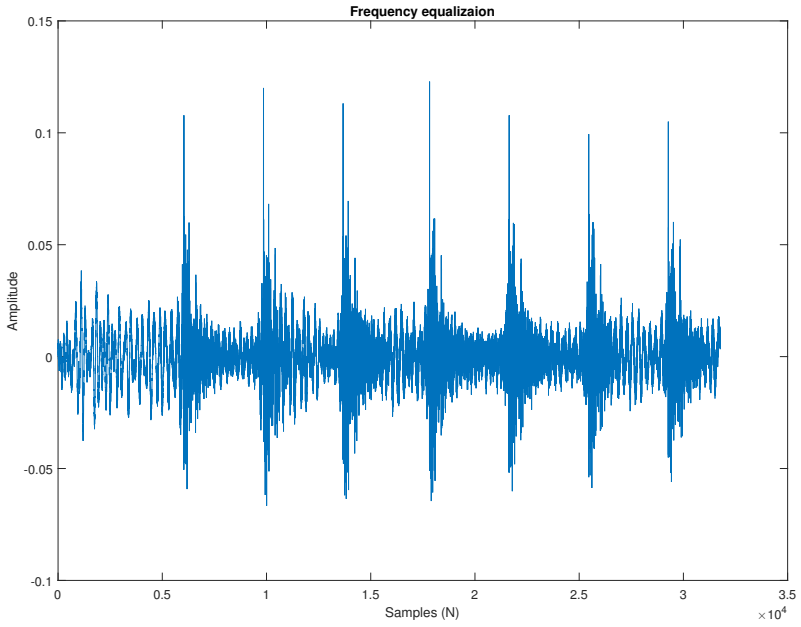

Figure 2: Typical impulse response of frequency equalization

```
17 H = H.*G;
```

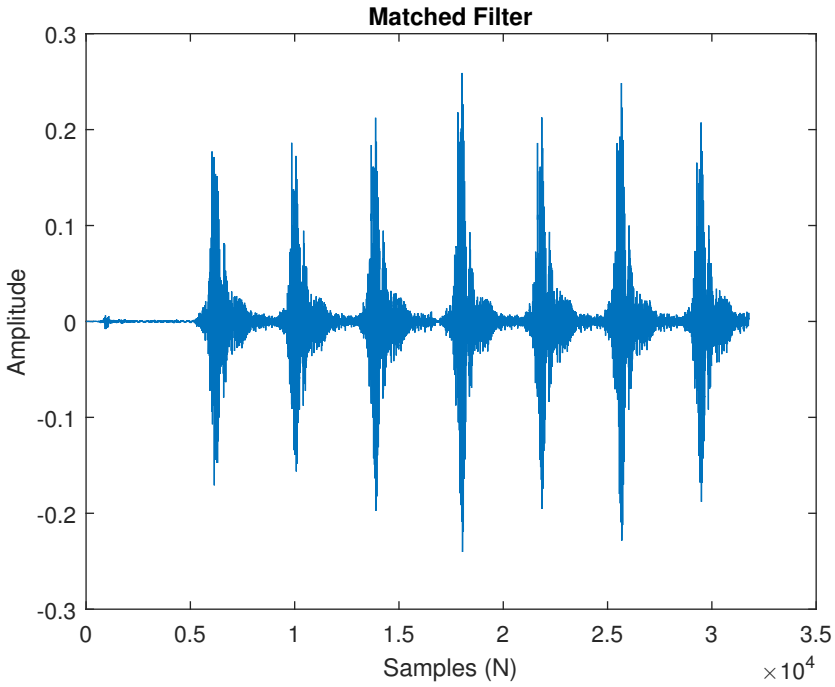

Figure 3: Typical impulse response of a matched filter

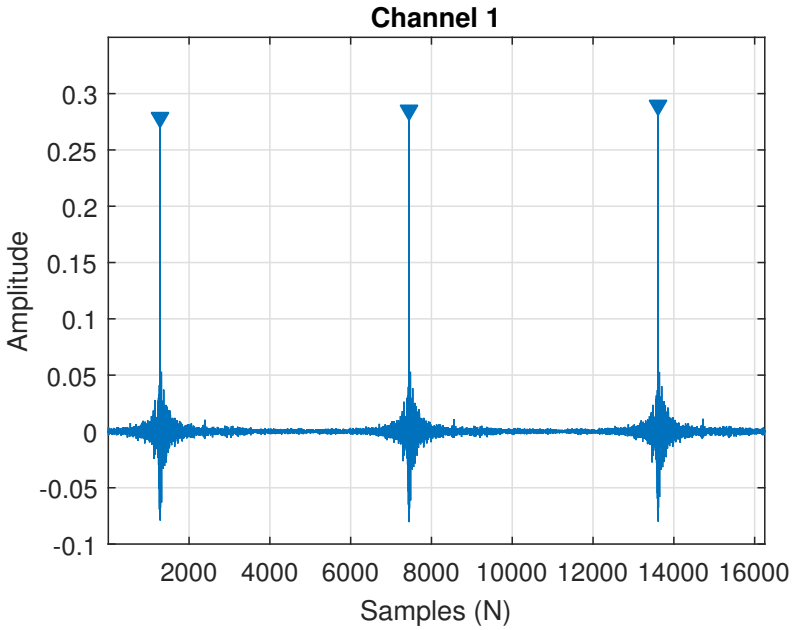

Figure 4: Peak detection channel 1

1.4 Peak detection

When the channels are estimated the position of the peaks of the estimated channels need to be determined. The peaks of the channels are determined using the *findpeaks* command in matlab. This function returns the height of the peaks and the location of the peaks based on the parameters given to the function as can be seen in Listing 2. The code used calculates the first peak of the signal after a 1200 samples silence period which corresponds to the maximum delay that should appear in the signal and a 6000 samples minimum peak distance range. The maximum delay found is across the diagonal of the field and therefore this delay in samples is calculated as $\sqrt{6^2 + 6^2}/340 * 48000 = 1.1979 * 10^3$ samples. Where $340m/s$ is the speed of sound and $48000Hz$ is the sample rate of the system. The minimum peak distance is based on the repetition timer of the beacon settings. An example of the calculated peaks of the first channel are found in figure 4. The first peak found after the silence period of 1200 samples is used as reference for the other peaks. The code/function used to calculate the first peak is found in appendix **??**

Listing 2: Peak position

```
6 Ts = 1200; %Search window
7
8 %Determine the location of the largest peak after a silence period
9 [peaks_f, locations_f] = findpeaks(h1(Ts:end), 'MinPeakDistance',
      6000);
```

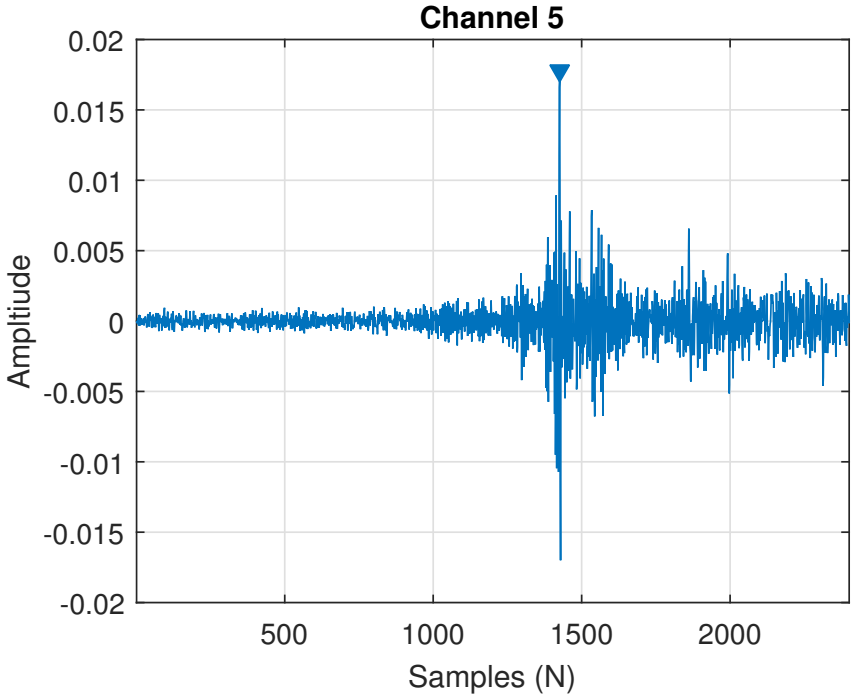

Figure 5: Peak detection channel 5

When the peak of the first channel is found this peak location is used as a reference for the other peak locations as can be seen in Listing 3 and is illustrated in figure 6. The position of peak 1 is the position of $nmax$ in the figure. A search window is applied to the second channel ranging from ± 1200 samples from the position of the peak of the first channel. The location of the highest peak in this interval is returned and corresponds to the delayed or leading impulse from the first channel. The resulting window is shown in figure 5 for channel 5 of the test measurement. The search window has a size of ± 1200 samples and the corresponding peak position is found. This same process is done for the remaining channels and all the peak positions are returned so the TDOA can be calculated.

Listing 3: Peak position channel 2

```
15 loc_peak_1 = loc_peak1;
16
17 [peaks_2_f,locations_2_f] = findpeaks(h2([loc_peak_1-Ts:loc_peak_1+
       Ts]), 'MinpeakDistance', 2*Ts-1);
```

1.5 TDOA

After the peak detection all the matching peak positions are known for each microphone. With these peak positions the differences between microphones can be determined. For use in the localization algorithm they will be returned in a row shape in cm. To convert the samples to cm it is divided by sample rate, the sample rate used is $48kHz$. With this time in seconds, the distance in meters can be recovered with the speed of sound. An estimate of 340 m/s is used for the speed of sound, this distance in meters multiplied will be the difference distance between microphones in cm. For the final challenge five microphones are present, the TDOA matrix will have the size of 5x5. The portion that calculates the TDOA is seen in listing 4. In table 1.5 the measurement used as a example in the peak detection subsection are used to display the tdoa matrix, the distance differences in cm are displayed.

Listing 4: TDOA calculation with microphone positions

```
27 %Calculate all the differences in distances
28 TDOA = zeros(5);
29 for x = 1:5
30     for y = 1:5
31         TDOA(y,x) = position_mic(x) - position_mic(y);
32     end
33 end
34 %Calculate the TDOA in centimers
35 TDOA = TDOA ./ 48000;
36 TDOA = TDOA .* speed_of_sound*100;
37 %Make an array from the data for the function localize
38 TDOA_2 = [TDOA(2,1);TDOA(3,1);TDOA(4,1);TDOA(5,1);TDOA(3,2);TDOA
    (4,2);TDOA(5,2);TDOA(4,3);TDOA(5,3);TDOA(5,4)];
```

distance	mic 1	mic 2	mic 3	mic 4	mic 5
mic 1	0	388.1667	434.2083	187.0000	160.0833
mic 2	-388.1667	0	46.0417	-201.1667	-228.0833
mic 3	-434.2083	-46.0417	0	-247.2083	-274.1250
mic 4	-187.0000	201.1667	247.2083	0	-26.9167
mic 5	-160.0833	228.0833	274.1250	26.9167	0

Table 2: TDOA matrix with data of measurement of peak detection figure 4.

1.6 Localization

In appendix B of the EPO-4 manual is a solution given to localize the car with the TDOA data. This algorithm is partially implemented. The location is found by solving a system of linear equations. First, the matrices have to be filled. The **A** and **b** matrix should look like this:

$$\mathbf{A} = \begin{bmatrix} 2(\mathbf{x}_2 - \mathbf{x}_1)^T & -2r_{12} & & & \\ 2(\mathbf{x}_3 - \mathbf{x}_1)^T & & -2r_{13} & & \\ 2(\mathbf{x}_4 - \mathbf{x}_1)^T & & & -2r_{14} & \\ 2(\mathbf{x}_5 - \mathbf{x}_1)^T & & & & -2r_{15} \\ 2(\mathbf{x}_3 - \mathbf{x}_2)^T & & -2r_{23} & & \\ 2(\mathbf{x}_3 - \mathbf{x}_2)^T & & & -2r_{24} & \\ 2(\mathbf{x}_3 - \mathbf{x}_2)^T & & & & -2r_{25} \\ 2(\mathbf{x}_4 - \mathbf{x}_3)^T & & & -2r_{34} & \\ 2(\mathbf{x}_5 - \mathbf{x}_3)^T & & & & -2r_{35} \\ 2(\mathbf{x}_5 - \mathbf{x}_4)^T & & & & -2r_{45} \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} r_{12}^2 - ||\mathbf{x}_1||^2 + ||\mathbf{x}_2||^2 \\ r_{13}^2 - ||\mathbf{x}_1||^2 + ||\mathbf{x}_3||^2 \\ r_{14}^2 - ||\mathbf{x}_1||^2 + ||\mathbf{x}_4||^2 \\ r_{15}^2 - ||\mathbf{x}_1||^2 + ||\mathbf{x}_5||^2 \\ r_{23}^2 - ||\mathbf{x}_2||^2 + ||\mathbf{x}_3||^2 \\ r_{24}^2 - ||\mathbf{x}_2||^2 + ||\mathbf{x}_4||^2 \\ r_{25}^2 - ||\mathbf{x}_2||^2 + ||\mathbf{x}_5||^2 \\ r_{34}^2 - ||\mathbf{x}_3||^2 + ||\mathbf{x}_4||^2 \\ r_{35}^2 - ||\mathbf{x}_3||^2 + ||\mathbf{x}_5||^2 \\ r_{45}^2 - ||\mathbf{x}_4||^2 + ||\mathbf{x}_5||^2 \end{bmatrix}$$

Where $\mathbf{x}_i$ are the locations of the microphones, r_{ij} is the range difference between microphones i and j.
With these matrices the location of the car can be calculated. This can be done by computing the pseudo-inverse of $\mathbf{A}$, i.e. $\mathbf{y} = \mathbf{A}^\dagger\mathbf{b}$. Vector $\mathbf{y}$ comes out as

$$\mathbf{y} = \begin{bmatrix} \mathbf{x} \\ d_2 \\ d_3 \\ d_4 \\ d_5 \end{bmatrix}$$

In $\mathbf{y}$, $\mathbf{x}$ is the location of the car and d_i is the difference from the car to the microphone.
This method works most of the time, except when the car is near the middle axis of the field. To counter this, the least squares solution is used. Least squares is also not perfect, but instead of a deviation of a meter with the pseudo-inverse, the deviation is maximum 4 cm.

1.7 Checking calculated position

To check if the found location is correct, the location will be checked for validity. This is done by using the previous (known) location. If the car has driven straight for a certain amount of time, the traveled distance cannot be larger than a known value. For instance, if the car has driven for 1 s the travelled distance is about 0.75 m. So if the car has driven for 1 s and the found location is further than 1 m away, the found location will be rejected. Also the cars orientation is kept and checked. If, for example, the car is known to be driving along the positive x axis, and to get to the found location, a sharp turn has to be taken from the known location, the found location will be rejected. Because the location solution is not very accurate in the middle of the playing field, every location found within a meter from the middle will be rejected. If this is the case, the car will drive as long as necessary to get

out of the middle. When a location outside the 1 meter radius from the middle is found, this location is tested for validity. Also a location outside the playing field is rejected. The matlab code for localization and checking of positions can be found in appendix **??**.

1.8 Resulting Design

The combined design for determining the position of the car is implemented in the control loop. First the function `TDOA` in listing 5 is to record audio and pass this on to the `ch5_tdoa_final` function. As seen in this function audio will be recorded from 5 channels with a sample rate of $48kHz$, the time to record will be 0.25 seconds. This time is enough to guarantee that at least 2 samples will be within the recorded sample, one of these samples will be complete enough to use for the deconvolution.

Listing 5: function TDOA

```
function [ TDOA_data ] = TDOA
%TDOA runs the tdoa and returns a column with distances differences
Fs = 48000;
Trec = 0.25;
measured_data = pa_wavrecord(3, 7, Fs*Trec, Fs, 'asio');
[TDOA_matrix, TDOA_data] = ch5_tdoa_final(measured_data);
end
```

Within `ch5_tdoa_final` the data will be split in data of each microphone, this data will be the input of the channel estimation. The channel estimation will return a estimated channel response, this response will be the input of the peak detection each microphone will receive their own channel response. The peak detection will pass the found position to a simple loop that creates the TDOA matrix. The function `ch5_tdoa_final` can be found in appendix **??**. This matrix will be the input of the localization algorithm, this algorithm determines the position of the car in x,y and z coordinates. Only the x and y coordinates are of importance, these will be updated in the global variable `position`. In the localization function the determined position will be checked if it is correct. If it does not "pass" the TDOA will be measured again, after three failed attempts to determine the position the car will first move before attempting again.

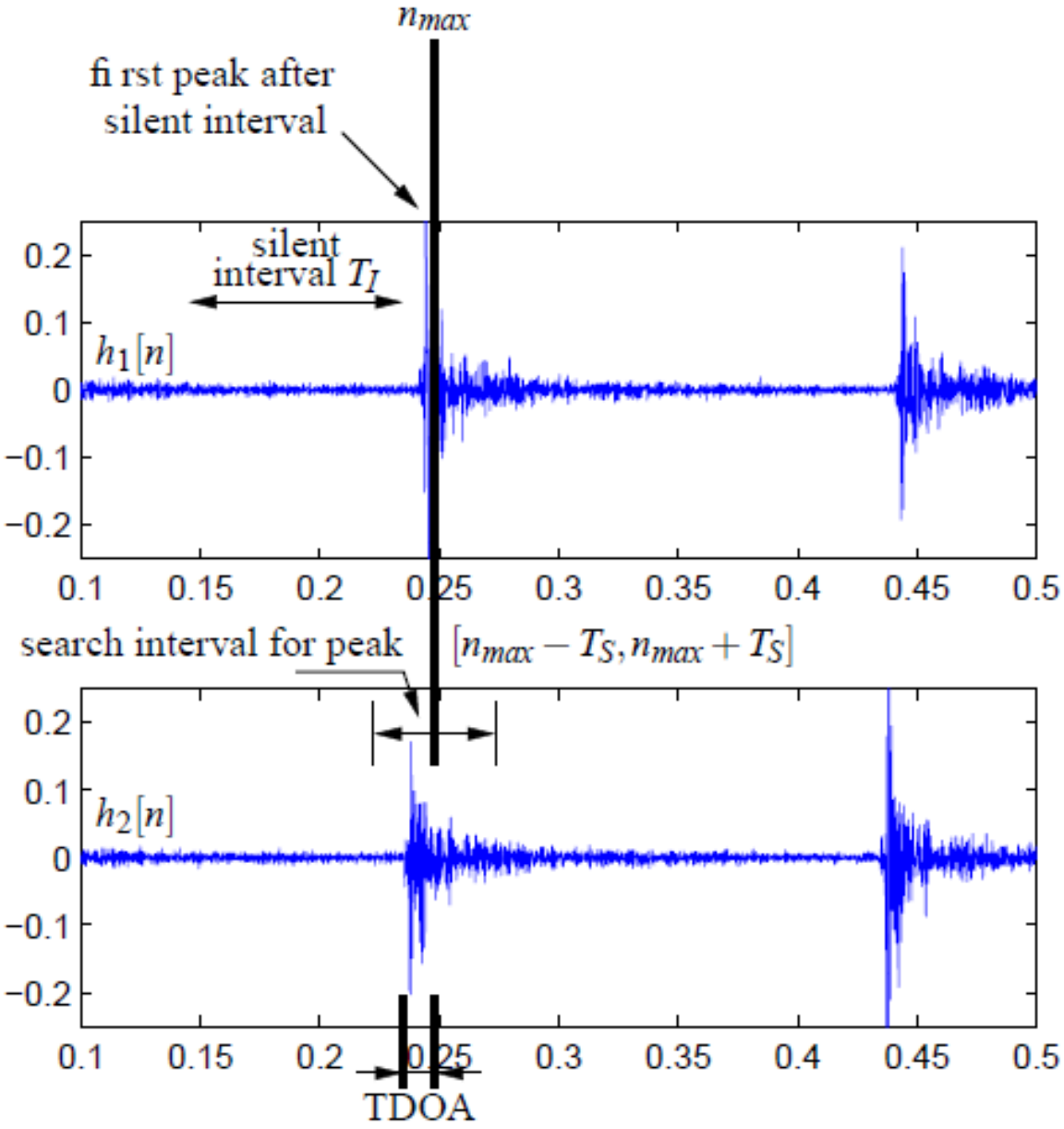

Figure 6: TDOA search window

1.9 Testing

For the testing of the multichannel localization, a test setup was used. In table 3 the bounds of the field are the positions of the microphones one through four. With this field ten measurements are done, the calculated position of these measurements can be seen in table 4. The reference used by the deconvolution is a measurement at one of the microphones. Because of the small distance to the microphone this can be seen as the impulse response of the training sequence through the microphone.

Looking at the results in table 4 the localization is not accurate and does not always work. During the test it was discovered that the first localization algorithm used does not provide an accurate location when the beacon is close to the middle of the field. This is because two of the TDOA values are zero, this happens when microphones are at equal distance from the beacon. For the horizontal middle line the distance from microphone 1,4 and 2,3 to the beacon is the same. This makes it hard to calculate a correct x position, the y position will be calculated correctly. For the vertical middle line the microphone pairs 1,2 and 3,4 are the same, this prevents the localization to correctly calculate the y position.

Setting of the beacon parameters:	Nbits: 32 Timer0: 2 Timer1: 4 Timer3: 7
(x,y,z) locations of the microphones:	Mic 1: 0, 0, 30 Mic 2: 413, 0, 30 Mic 3: 413, 210, 30 Mic 4: 0, 210, 30 Mic 5: 173, 0, 77
Measurement 1:	Beacon at Mic 1
Measurement 2:	Beacon at Mic 2
Measurement 3:	Beacon at Mic 3
Measurement 4:	Beacon at Mic 4
Measurement 5:	Beacon at Mic 5
Measurement 6:	Location(x,y,z): (A) 102, 70, 26
Measurement 7:	Location(x,y,z): (B) 155, 105, 26
Measurement 8:	Location(x,y,z): (C) 216, 88, 26
Measurement 9:	Location(x,y,z): 203, 210, 26
Measurement 10:	Location(x,y,z): 355, 75, 26

Table 3: Measurement position of test setup.

measurement	1	2	3	4	5
x	-0.7041	250.8521	393.7024	3.8097	167.8806
y	-9.9271	394.9473	199.2680	201.5119	1.4720
z	-16.3226	1.1500e+03	19.9370	10.4180	229.1798

measurement	6	7	8	9	10
x	109.1353	216.1429	220.9459	204.5990	136.2869
y	73.1398	105.0000	75.3259	63.1943	176.8470
z	-10.6656	-594.9538	367.3060	-529.2243	-217.4669

Table 4: Localization results for tests.

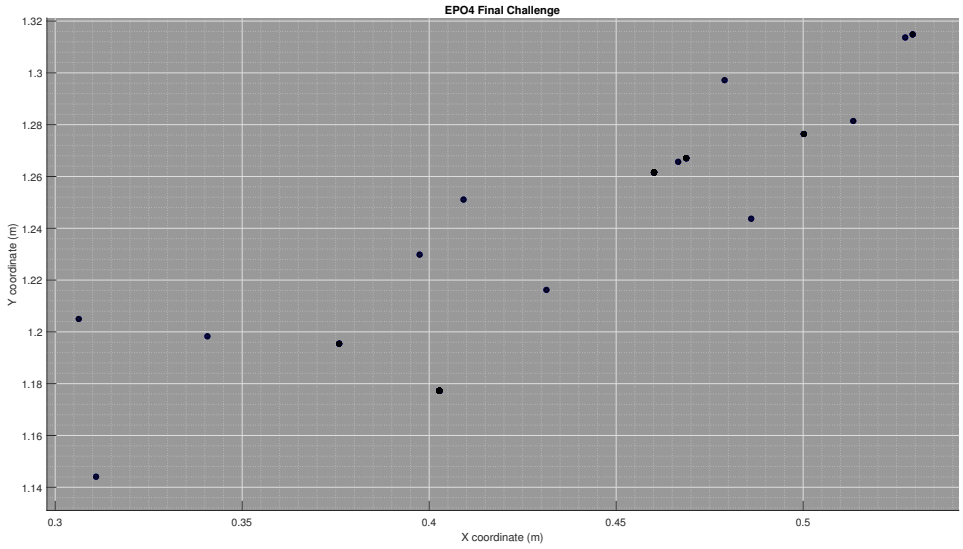

Figure 7: Positions detected by the code on the final test field

1.9.1 Field testing

The TDOA and localization code is also tested on the field that will be used for the final challenge. The car was stopped at a position and the code was ran several times. In figure 7 the results are visualized, the black dots represent the positions detected. The localization of the car is determined with an accuracy of ± 10cm. This should be enough for the car to reach it's destination within 30cm, which is required for the final challenge.

1.10 Conclusion

The localization of the car is done with a reasonable accuracy and it should be high enough for the car to reach it's goals. However with a sample rate of 48kHz the location of the car could be determined more precise. Sound travels around $1/(48kHz) * 340m/s = 0.7$ cm per sample so it should be possible to determine the position of the car with a higher resolution than ±10 cm. The accuracy of the position is reduced when the car approaches the positions where x = 3m or y = 3m. This issue is now solved through filtering the positions and driving a bit further in case the car is near the mid lines (x = 3m or y = 3m) of the field. It could be resolved though by using a different algorithm for the localization of the car.

2 Route planning

2.1 Introduction

The role of the route planner is to decide how the car should move given its current objective. This can be a number of different things, for example, making a turn, driving a straight line or stopping all-together.

2.2 Design

The only variable that is really fixed is the route. This can be either only a destination or first a waypoint and then a destination. All other variables and measurements are approximations and are prone to errors and uncertainties. Because of this nature of our system the design of the planner is made with compensation and iteration in mind from the start.

This is the reason for the main design choice, namely that there is no fixed trajectory which is tracked. Instead of tracking a global trajectory, the Planner is run 'fresh' after each succesfull localization attempt. The Control Loop section3 shows when and why the Planner is executed. This means that the only thing that the planner needs to know is the current position and orientation of the car.

2.2.1 Specifications and Goals

The information available to the planner is summed up in table 2.2.1.

Name	Explanation
`position`	matrix containing all the known positions and orientations of the car, with th
`car.voltage`	contains the current voltage of the car in Volts.
`static_positions.route`	contains the waypoints (if there are any) and the destination of the car in x a
`static_positions.point`	keeps track of which waypoint/destination we are traveling to.
`car.steer_straight`	contains the steer setting for driving straight.

Table 5: Information and variables available to the planner.

The data and information that the planner should provide to the control loop3 is available in table 2.2.1.

Name	Explanation
`speed`	the cars needed speed setting including voltage drop compensation.
`car.d_theta`	difference in angle between the car and the next waypoint/destination.
`car.v_factor`	factor between `0` and `1` depending on the starting voltage and the current voltage of the c
`steer`	steer setting of the car depending on whether to make a turn or not and which way.
`car.status`	status of the car, `1` when at a waypoint, `2` when at the destination and `0` elsewhere.
`car.did_turn`	toggles between `0` and `1` reflecting whether the last driven action of the car was a turn or
`time`	time in seconds the car has to move, this movement can be a turn, driving straight or sta

Table 6: Information and variables returned to the Control Loop.

2.2.2 Code

The code of the Planner?? consists of two parts. In the first part all needed information is generated and the second part consists of a big nested loop which figures out what needs to be done when considering all the available information.

speed setting adjustment
Because of the different behaviour of different cars the values in this code are determined by trial and error, which is generally explained in paragraph 2.3.1.

Listing 6: Speed setting adjustment for different Voltages

```
8  if car.voltage >= 17 % speed compensation for voltage drop
9      speed = 163;
10     disp('Voltage level: normal, just cruizing')
11 elseif car.voltage >= 14
12     speed = 162;
13     disp('Voltage level: meh, lets hit the gas')
14 else
15     speed = 165;
16     disp('Voltage level: drained, pedal to the metal!')
17 end
```

generation of information
First the number of fixed points is extracted from the `static_positions.route` variable. Then the argument and modulus of the to be traveled vector is calculated using basic trigonometry and the Pythagorean theorem. Finally the voltage compensationfactor is determined using the current car voltage.

Listing 7: Generation of needed information

```
19 [~,numberofpoints] = size(static_positions.route);
20 d_x                = static_positions.route(1,static_positions.point)
      -position(1,end);
21 d_y                = static_positions.route(2,static_positions.point)
      -position(2,end);
22 desired_theta     = atan2d(d_y,d_x);
23 car.d_theta       = desired_theta - position(3,end);
24 distance          = sqrt(d_x^2 + d_y^2);
25 test_data.dtheta = [test_data.dtheta, car.d_theta];
26 car.v_factor = 1; %car.voltage / car.voltage; %temporary always 1
```

the big nested if

This part of the code performs the necessary checks to determine what needs to be done with the above generated information. The first check that occurs is whether we have arrived within 30 cm of our current destination, which can either be a waypoint or the final destination. When this is the case the Planner outputs variables to the Control Loop to stand still and informs whether it's a waypoint or destination. When that check returns false we know we have to go somewhere, now the code checks whether the previous movement was a turn or not. This is done because we need to be able to calculate the cars orientation and this can only be done when a straight line has been driven from its previous to its current location. So if the previous movement was a turn, the car is ordered to drive a bit in a straight line so that the next time it performs this check it will not have made a turn. Now we check whether the car needs to make a turn or not, if it has to make a turn it calculates the time it needs to do that using the `turn` function**??** and provides the correct steer setting depending on whether the variable `car.d_theta` is positive or negative. If it doesn't need to make a turn the code differentiates between 2 possible situations. Situation 1: the car is almost at its current destination. If that is the case the function `straight` returns the time it takes to drive to that destination. Situation 2: the car has quite a long way to go before reaching its destination, so we just drive for a bit in a straight line. It does this because due to the aforementioned inaccuracies and uncertainties loads of things can be way of. Therefore this will all resolve itself by running the Planner again after getting a bit closer to our goal destination.

Listing 8: The big nested if

```
27 if distance <= 0.3
28     steer = car.steer_straight;
29     speed = 150; %standing still
30
31     if static_positions.point >= numberofpoints
32         time = 40;
33         disp('WE MADE IT, right? RIGHT?!')
34         car.status = 2;
35     else
36         static_positions.point = static_positions.point + 1; % keep
               track of where we're going
37         disp('ARRIVED AT A WAYPOINT, on to the next one!')
38         time = 3;
39         car.status = 1;
40     end
41 else
42     car.status = 0;
43     if car.did_turn == true
44         time          = 1 / car.v_factor;
45         steer         = car.steer_straight;
46         car.did_turn = false;
47         car.did_last_turn = true;
48         disp('Driving straight a bit to provide data for
               orientation after we made a turn.')
```

```
49      else
50          car.did_last_turn = false;
51          disp('To turn or not to turn, thats the question...')
52          if abs(car.d_theta) <= 5    %straigth
53              disp('No turn!');
54              if distance <= 1.5;
55                  time    = straight(distance);
56                  steer   = car.steer_straight;
57                  disp('Almost there, careful now, we got one shot (
                        nope, just kidding).')
58              else                    %distance is greater than 1.5m
59                  time    = 2 / car.v_factor;    % return a 1s drive
60                  steer   = car.steer_straight;  % straight
61                  disp('Lets drive in a straight line for a bit and
                        check again later.')
62              end
63          else                        % turn
64              disp('Lets turn')
65              %car.d_theta    = roundn(car.d_theta,1.5); % round up
                    or down to nearest 15 degrees.
66              car.did_turn    = true; % make sure that we drive a bit
                    in a straight line next
67              if car.d_theta <= 0
68                  steer = 100;
69                  disp('to the right!')
70              else
71                  steer = 200;
72                  disp('to the left!')
73              end
74              time = turn(car.d_theta);
75          end
76      end
77  end
```

2.3 Testing

To implement the `turn` and `straight` functions mentioned in the previous code section several time tests were done. Because testing with supercaps was impossible we tested with the batterypacks, this meant that no detailed voltagedrop effects on the timings could be found. To simplify things further linear approximations were used, it was decided that when our timings were not causing overshoot our final goal should be reached because of the compensating and self-correcting behaviour of our system.

To determine the linear approximation of our curve timings the following measurement-data was inter- and extrapolated.

To determine the linear approximation of the timings for driving in a straight line we calibrated the speed setting to produce a 1.3 s drive for covering 1 m.

To get an idea of the behaviour of the Planner when included in the Control Loop, small messages were added to the code. These messages showed up in the console while driving with updates on what part of the 'big nested if' was chosen and in conjunction with saved test data, errors or unexpected behaviour was easy to spot.

2.3.1 Trial and Error

Between each run small changes to the variables were made to ensure a viable 'maximum car voltage' performance. These found settings would degrade when the voltage would start to drop but the code is able to compensate for that by rerunning itself untill the car arrives at the destination. This means that in optimal state the car would make only one turn per waypoint and more turns when the voltagedrop makes the turns less precise.

2.3.2 Findings

2.4 Conclusion

To improve on this design the linear approximations and guessed factors could be removed and replaced by measured curves for different voltage levels.

3 Control Loop

The control loop is the main function. All other functions are called from within this function.

3.1 Overview

To implement the control loop, a MATLAB timer is used. The timer object is very useful because it handles a few things which are hard to implement in a normal while loop. A timer has 4 main functions, which are `StartFnc`, `TimerFnc`, `StopFnc` and `ErrorFnc`. The `StartFnc` is used to initialize things just before the car starts driving. The `TimerFnc` is the main loop. In this function the car will drive and all the functions are called. A state machine is used to keep everything clear and structured. The `StopFnc` is used to clean when the function is done driving the car. Last but not least is the `ErrorFnc`. This function can handle every error thrown by one of the other functions. This function mainly stops mayhem from happening.

3.2 Check if supercaps are charged.

For the final challenge the car has to be charged wireless, this is done with a air-core DC/DC converter. The super capacitors need to be charged to 17 V, with the wireless communication the current voltage can be read. The voltage measured fluctuates heavily, because of this an outlier will count as 17 V. To make sure the super caps are charged to 17 V the measurement will count each time a value above 17 V is measured, if the count is five it will assume the car is charged. After the car is charged the next state will be Drive.

3.3 Drive

When all the parameters for the car are determined with the planner, a drive command has to be sent to the car to make it drive. A separate function is made to do this to be sure all the parameters are determined before transmitting it to the car. The function drive_car.m, in appendix **??**, has as inputs a speed setting, steer setting and a time to drive. It executes the drive command for the time needed using a pause command to wait, before transmitting a roll out command which is a speed setting of 150 and a steer setting that corresponds to driving a straight line. After this is done, the next state will be Sample.

3.4 Sample

First the TDOA data will be determined. With this data, a location will be determined with the function localize. If the found location does not pass the tests in localize, a fail code will be given by localize. When the localization fails, new TDOA data will be determined. This will be done until 3 tests fail. When 3 tests fail, the car will drive a little bit forward to hopefully find a better spot for a measurement. When the tests fail 9 times in a row, a different action is taken depending on the failed test.

3.5 Mapping the car

To visualize the location of KITT the matlab class EPO4figure from blackbloard, appendix **??**, is used to map the car location, mic positions, waypoint, destination and obstacle locations.

The class has 5 functions: `setMicLoc`, `setWayPoint`, `setDestination`, `setKITT` and `setObstacle` which all require a location in some form as input. A variable `start` is added to

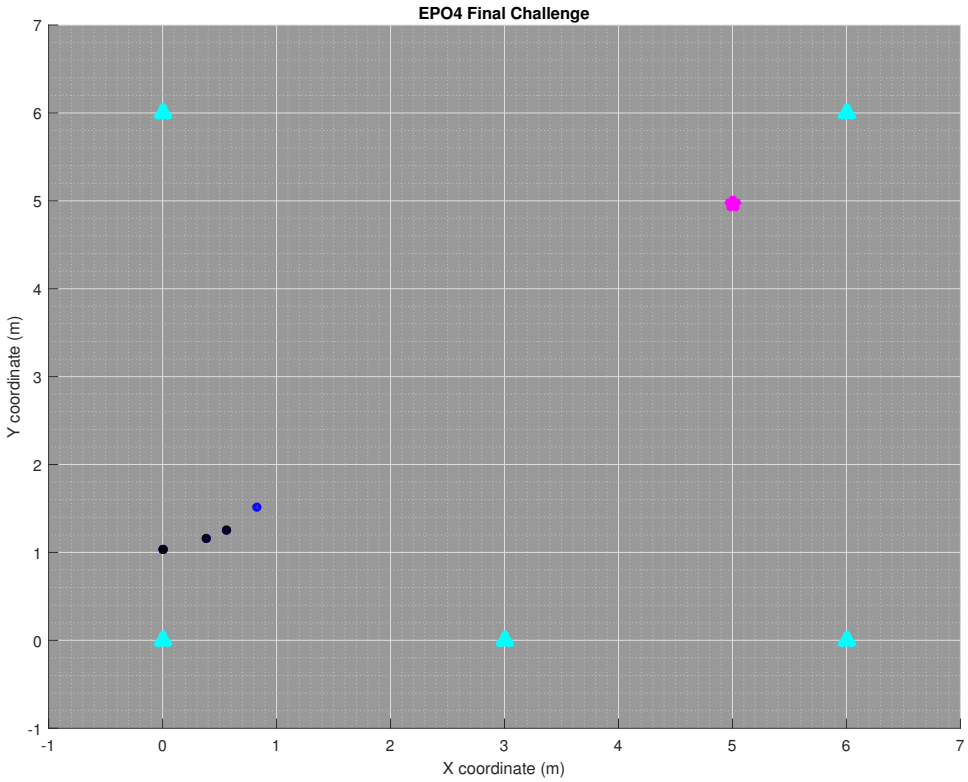

Figure 8: Mapping the locations

the function `setKITT` to prevent the figure to load the last position from a previous test. The last position of the car is shown as a blue dot and the past positions of the car are shown as a black dot, the mic positions are the light blue dots and the violet ones are the destination and waypoint positions as can be seen in figure 8.

4 Conclusion and Discussion

5 Planning and Teamwork

5.1 Planning

The planning of the final challenge is divided in four weeks. The first three lab day sessions are used for localization and multichannel measurements. the three lab day sessions after are meant for system integration, putting all parts together and writing functions to plan the route and control the car. The last few lab days are used for testing with the final setup. The final report deadline is planned for Friday 19 June noon, for this it was decided that two days before this deadline at 18:00 the individual works are finished and afterwards the written pieces will be corrected and if something is missing written. The final report was divided into parts and each individual will write the pieces they worked on, this can be reviewed on the next page.

5.2 Teamwork

Introductie		Martijn
	Abstract	Martijn
	Introduction	Martijn
Specs		Thomas
	Specs	Thomas
Planning/Teamwork		Martijn
	Planning	Martijn
	Teamwork	Martijn
Plaatsbepaling		Martijn
	Introduction	Martijn
	Training sequence	Jeroen
	Reference	Martijn
	Deconvolution	Martijn
	Peak Detection	Richard
	Localization	Jeroen
	Checking calculated position	Jeroen
	Testing	Richard
	Conclusion	Richard
Object detection?		Martijn
	Introduction	Martijn
	Design	Martijn
	Testing	Martijn
	Conclusion	Martijn
Route planning		Thomas
	Introduction	Thomas
	Design	Thomas
	Testing	Thomas
	Conclusion	Thomas
Loop		Jeroen
	Timer Function	Jeroen
	Determining Voltage	Martijn
	Driving	Richard
	Plotting	Richard
	Challenges	Jeroen
	Error	Jeroen
Conclusie		Richard
Discussie		Richard
Appendix		Thomas

References

A Appendix

A.1 Module 1

A.1.1 Open_com.m