ELAD - Easy Learning for Autonomous Driving

Mechatronics Advanced course MF2059 Project report

Agnes Jernström
Fangjin Kuang
Nikhil Bohra Nirmal Kumar
Gabriel Andersson Santiago
Henrik Ljunggren
Ramin Seyed Farshchi
Ylva Steffner

Abstract

ELAD - Easy Learning for Autonomous Driving, is a project that aim at developing a learning platform for autonomous driving for the consultancy company ÅF in order to facilitate learning about autonomous drive for their consultants. An autonomous car in size 1 to 10 of a real car was built using a RC car, a camera, a LIDAR, an IMU and motor encoders. The setup consists of a motherboard, Nitrogen 6 MAX, a PCB STM-32 for connection of sensors and a host computer. The motherboard runs on Lubuntu and communicates over Wi-Fi with the host computer and over CAN with the STM-32 card. ROS is used as a framework for the software. Besides setting up the system, different functionality was required to be implemented as well. The main requirements regarding vehicle functionality was; lane following for straight and curved roads, obstacle detection and avoidance, stop line and intersection detection and intersection handling. These functionalities were successfully implemented and the robot could drive in the constructed track autonomously. In order for the vehicle to function as a learning platform the following measures were taken; a simulation environment was setup in Matlab and Simulink, a GUI was created that enabled monitoring and control of the vehicle, the standard tool for documenting the code, Doxygen, was used and the project was documented on a local web page that enabled easy overview and navigation.

Acknowledgements

We would like to thank our supervisor at KTH, Naveen and our supervisors at ÅF, Carl-Johan, Xavier, Jan, Peter and David for feedback and support throughout the project.

Gabriel Andersson Santiago, Nikhil Bohra Nirmal Kumar,
Agnes Jernström, Fangjin Kuang, Ylva Steffner,
Henrik Ljunggren and Ramin Seyed Farshchi

Stockholm, December 2016

Nomenclature

Symbols

θ	Orientation
ω	Angular velocity
v	Translational speed

Abbreviations

GUI	Graphical user interface
ROS	Robot operating system
CAN	Controller area network
IMU	Inertial measurement unit
RC	Radio controlled

Contents

1 Introduction

1.1 Background

Autonomous driving is developed, tested and implemented at a rapid pace in order to increase road safety, save energy and free up commuting time etc. A self-driving vehicle independent of human control consists of four fundamental technologies as shown in Figure 1 and listed below:

1. Observation
2. Orientation
3. Path planning and decision-making
4. Actuation

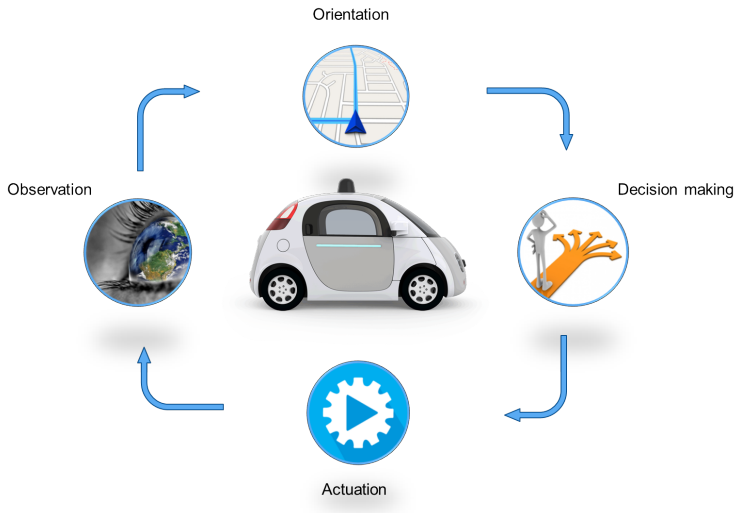


Figure 1: Fundamental areas for self-driving vehicles.

The observation module provides information about the surrounding environment by sensing the environment structures in a multi-sensor way, using for example camera, LIDAR and encoders. Orientation, the second module, interprets the sensor information in order to estimate the locations of geometric features such as objects and lanes. The purpose of the decision-making module is to ensure that the vehicle follows the rules regarding safety, vehicle dynamics and environment contexts. The final module, actuation, executes the commands necessary to steer the vehicle according to the decisions taken in the previous module [1].

1.2 Purpose

In order to research autonomous driving, miniature vehicles in the scale 1:10 of a real car can be used. This is what the ELAD - Easy Learning for Autonomous Driving, project aims at. The ELAD project is a student project for the "Advanced Course in Mechatronics" (MF2058 and MF2059) at KTH. The project is issued by the consulting company ÅF. The aim of the project is to improve ÅF's teaching platform about autonomous vehicles so that consultants at ÅF quickly can learn, test and research autonomous drive in order to be prepared for customer projects in that area.

1.3 Scope

In this report "self-driving" denotes a vehicle that is able to:

- Follow a lane
- Avoid obstacles

without the help of any human input. The aim of the ELAD prototype has thus been to run within a lane consisting of one solid and one dashed line and to stop in front of objects appearing in the lane. The prototype does not need to know its destination or where it is located in relation to a real world map. The test track consists of white lines on a black carpet to maximize the contrast. It can be seen in figure 2.

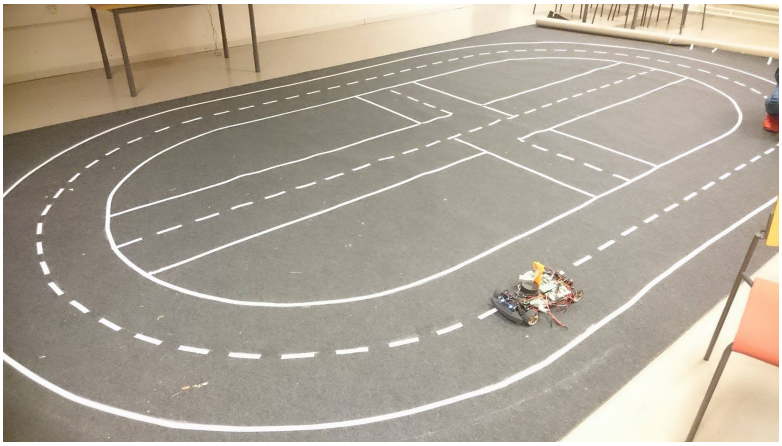


Figure 2: The track used for testing

The road on the test track consists of two lanes and each lane is 30 cm wide. The vehicle is only built to handle completely flat surfaces. The prototype does not have a set minimum speed. For a full set of requirements, see Appendix B. The aim has been to fulfill requirements up to priority 2.

1.4 Method

The development has been done iteratively using Scrum with two week sprints. New functions have been tested continuously. The tasks have been created based on the prototype requirements which in turn have been prioritized. Tasks based on high priority requirements have been focused on first.

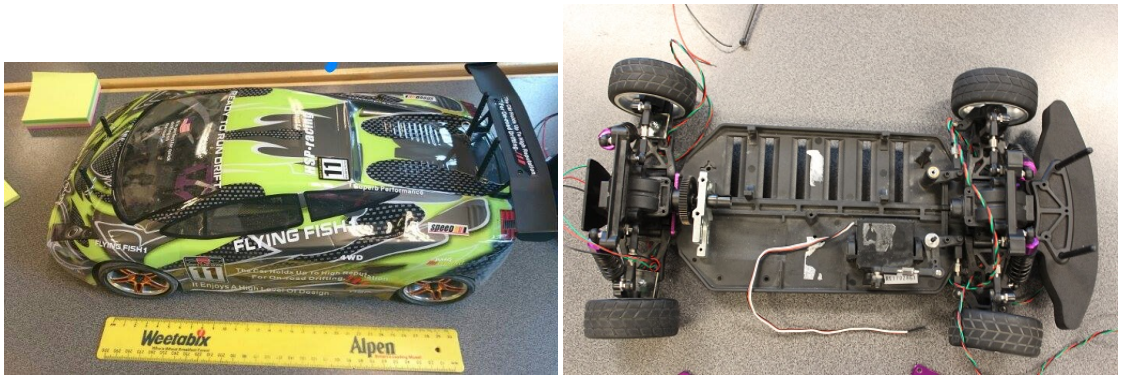
2 System Architecture and Hardware setup

This chapter goes through the setup of the prototype. It describes how everything is connected, what software and hardware is used and how the system is configured.

The ELAD project is connected to two other similar projects at ÅF about autonomous drive. All necessary hardware for the project was suggested by one of these teams and supplied by ÅF. This included the choice of motherboard, microcontroller, lidar, camera and encoders. A general layout of the car was also suggested. The components and the layout were evaluated in the spring and were deemed good enough for the project.

2.1 Assembly of prototype

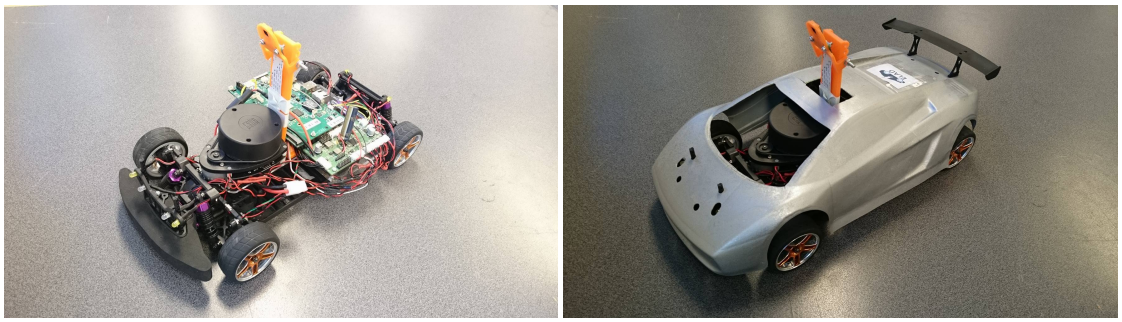
A RC car, HSP Flying Fish 1/10, have been used as a platform for the autonomous prototype. Figure 3a shows the RC car and Figure 3b shows the RC car after it has been stripped of all components, except the servo for steering, in order to prepare it for a transformation into an autonomous vehicle.



(a) RC car (b) RC car disassembled

Figure 3: The original car and the car after everything had been disassembled.

A lidar, a camera (Nit6X-5MP), encoders, a mother board (Nitrogen6-MAX) and a PCB (STM32, made by ÅF for this project) were mounted on the stripped RC car, figure 4a.



(a) Sensor for autonomous driving mounted. (b) Transformed autonomous car with chassis.

Figure 4: The transformed autonomous car with and without the chassis.

To give the camera a larger field of view it was mounted in a 3D printed stand, seen in orange in figure 4a. The stand was constructed to allow the camera angle and height to be changed if deemed necessary. Headlights, taillights and blinkers were also added to the prototype to allow it to follow standard traffic rules in further implementations. These are connected to the STM-32

card.
Since one of the requirements for the ELAD project was that it should be easy for someone to replicate the prototype or improve on it, a full instruction on how to assemble the prototype was created. This can be found in appendix A.

2.2 Hardware specifications

This chapter describes hardware used in the project in detail.

STM-32

STM32f4 was the microcontroller used in the project. It was mounted on a custom made PCB made by ÅF which had connectors for all the necessary sensors. The code for the microcontroller was written by ÅF and was already flashed to the microcontroller when the PCB was recieved. The PCB utilizes FreeRTOS as the real-time operating system for scheduling the sensor reading and actuating which is done on the PCB. No modification was made by us in the code on the STM-32 card because it fulfilled all the requirements needed for the project.

LIDAR - RPLIDAR 360

The LIDAR used is the RPLIDAR 360° which is a 360 degree 2D omnidirectional laser scan, meaning it sends out one laser which measures the distance to the closest object and spins to get a full view of its environment. The scanning frequency can be configured between 1 Hz and 10 Hz, with a standard frequency of 5.5 Hz, via user configurable PWM signal. The standard frequency of 5.5 Hz have been used in this project. The motor for the LIDAR needs a power supply of 3.6-6V. The digital system power supply is 3.6-5V. The communication interface for the RPLIDAR is UART/USB. The RPLIDAR can detect objects between 0.2 to 6 meters with an accuracy of 99% of the measured distance [2]. As seen in figure 4a, part of the lidar's range will be obstructed by the camera stand. This was deemed acceptable since the lidar is only used to find objects which are in front of the vehicle. The lidar has been placed 10 cm above the ground in order to keep the front of the vehicle from obstructing the view of the lidar. For an object to be detected it needs to reach the same height as the lidar.

Camera

The camera chosen for this project is the $Nit6X\_5MP$. The driver of the camera module operates from a single 2.3V to 5.5V supply. Some core data is presented in table 1 [3].

Resolution	2592 x 1944
Frame-rate	15 fps, 60 fps, 120 fps
View angle	65°
Focusing Range	10 cm - inf

Table 1: Core camera data

Motor

The motor used is the FingerTech "Gold Spark" 16mm Gearmotor. It operates on 3V - 18.5V, but operating voltage higher than 10 V decreases the motor life. There are nine different gear ratios; 20, 35, 50, 63, 86, 115, 150, 250, and 360:1. The one used in the project had a gear ratio of 1:20. For more information about the motor, see datasheet in reference [4].

Encoders

The encoders used in this project uses hall effect sensors mounted along the wheel axis. There are ten magnets mounted inside each wheel and every time a magnet passes by the hall effect sensor a pulse is sent to the STM32 board. Only the front wheel encoders was used in this project for calculating the rotational speed of the wheels and could with that information calculate the speed of the vehicle.

Inertial Measurement unit - MPU 9250

The IMU used in the project is a 9-axis MPU 9250 that consists of

- a 3-axis gyroscope for angular velocity
- a 3-axis accelerometer for linear acceleration
- a 3-axis magnetometer for direction of local magnetic field
- a Digital Motion Processor

The channels the IMU uses on the STM board are AX, AY and AZ for the values from the accelerometer, GX, GY and GZ for the values from the gyroscope and MX, MY and MZ from the magnetometers. It needs three frames to send data over the CAN bus. [5]

2.3 Architecture

The lidar, IMU, encoders, motor and servo are connected to the STM32 PCB that is made by ÅF specially for this project. The camera is connected to the motherboard, a Nitrogen6_MAX. The motherboard and the STM32 communicates via CAN and the motherboard can communicate with a host computer via Wi-Fi. This is illustrated in figure 5.

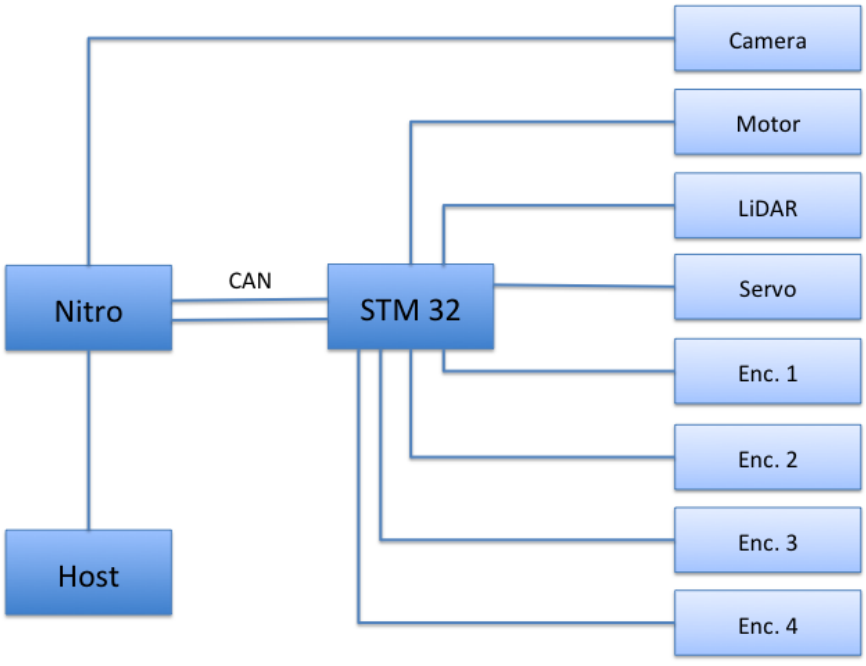


Figure 5: System architecture. All components except the camera are connected to STM32. The camera is mounted on the Nitrogen motherboard. STM32 and Nitrogen communicates over CAN and Nitrogen communicates with a host computer via Wi-Fi.

2.4 Configuration

Lubuntu

The motherboard runs Lubuntu, an open source, light weight operating system based on Linux [6]. All code on the motherboard and the host computer is written in C++ and handled by ROS.

ROS

ROS stands for Robot Operating System and is an open source framework for writing software for robots. It contains libraries and tools which simplify the process of developing a robot [7]. The file structure in ROS is built upon nodes. The nodes are independent of each other and each node can contain several functions. For ROS to work you need to setup a master node that enables all the other ROS nodes to find and communicate with each other. ROS nodes communicate with each other by publishing and subscribing to topics [8]. An example of how ros is working can be found in figure 6.

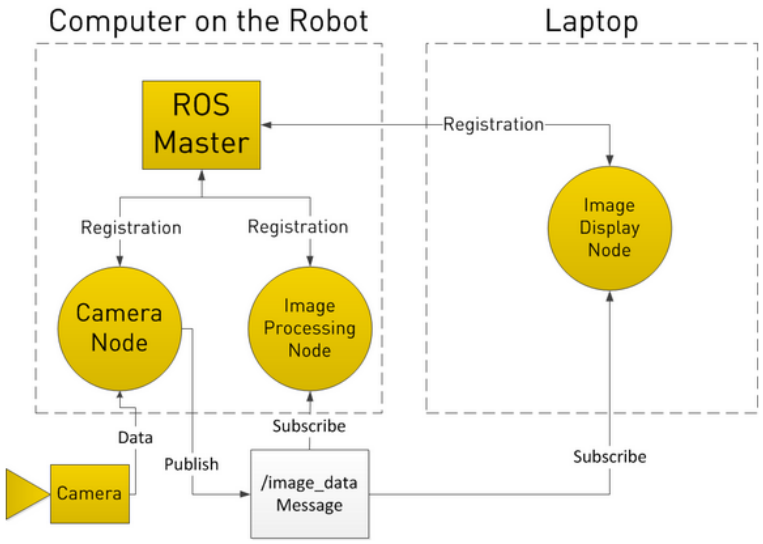


Figure 6: Example of how ROS is working, with a setup of a camera node, connected to a camera, an image processing node and an image display node. The camera node states when registering with the master node that it will publish a topic called image_data. The imaging processing node and the image display node states that they will subscribe to the topic image_data. This enables the camera node to directly send data to the other nodes as soon as it gets data from the camera. [8]

All nodes, except the GUI node and the lane detection node, are located on the motherboard on the vehicle. The GUI node and lane detection node are located on a host computer. All nodes and their connections can be viewed in appendix D.

CAN

The motor, the servo and all the other sensors except the camera are connected to the STM-32 card. In order for the sensors to send information to the nodes on the motherboard a communication network over CAN was set up. The communication between the different nodes and the CAN-bus is handled by the CAN node. This node is divided into two parts, one that receives CAN messages from the STM-32 board and one that transmits messages to it.

The receiving end periodically checks for new messages on the CAN-bus at a rate of 100 Hz. This rate is low enough as to not interrupt any callbacks but still fast enough for the applications in need of the data. Each message has a unique ID. Depending on this, the current message is sent to a callback function which interprets the data and forwards it to a ROS topic.The data received from the CAN-bus is divided in accordance to table 1.

Frame	Data type	Content
1, MPU9250 Sensor (ID 200)	*int*16_*t*	Accelerometer X
	*int*16_*t*	Accelerometer Y
	*int*16_*t*	Accelerometer Z
	*int*16_*t*	Gyroscope X
2, MPU9250 Sensor (ID 201)	*int*16_*t*	Gyroscope Y
	*int*16_*t*	Gyroscope Z
	*int*16_*t*	Magnetometer X
	*int*16_*t*	Magnetometer Y
3, MPU9250 Sensor (ID 202)	*int*16_*t*	Magnetometer Z
	*uint*16_*t*	MPU time stamp
	*uint*8_*t*	Car speed (cm/s)
	*uint*16_*t*	Time stamp of car speed
	*uint*8_*t*	Battery level
Lidar data, (ID 100)	*uint*16_*t*	Distance to object
	*uint*16_*t*	Angle to object
	*uint*8_*t*	Quality
	*uint*8_*t*	Start bit
	*uint*16_*t*	Lidar timestamp

Table 2: Sensor data from CAN-bus

Each byte of data contains 8 bits. This means two bytes need to be shifted together for the 16 bit variables. This is done in each callback function.

The transmitting part subscribes to all topics, which contains data needed by the STM-32 board in order to control the vehicle. When a new message is posted on one of these topics the data is inserted into an array. This array is part of a struct that also contains an ID that is common to all messages sent out by the ROS node and the length of the message (8 bytes). The array is divided as shown in table 2.

Position:	0	1	2	3	4	5	6	7
Data to:	Motor	Servo	Lidar	Lights/buzzer	Nothing	Nothing	Nothing	Settings

Table 3: Message array

3 Observation and Orientation

The vehicle uses several different sensors to perceive its environment and orient itself within it. This chapter describes how this is done.

3.1 Odometry

Odometry describes how the pose of the car changes over time. The pose includes both the position and orientation. There are different ways to get the pose depending on what kind of architecture the robot has. Odometry in this project was done by using the data from the IMU module MPU 9250 and magnetic encoders in the wheels.

In order to interpret the pose of the car, two Cartesian coordinate frames were set up. One is called map frame which is stationary relative to the real world and the other one is called the base link frame which has its origin in the middle of the front axis of the car. The x-axis for the base link frame is always aligned with the direction of the car. The pose of the car e.g. the pose of the base link frame is relative to the map frame and the x, y and θ coordinates describe this link between frames.

To calculate the pose (x, y, θ), the translational speed and angular velocity are necessary. In order to get translational speed, two hall effect sensors working as encoders were assembled to the two front wheels. The velocity could then be calculated according to the formula below:

$$v = \frac{(P_1 + P_2) \times \pi \times a}{2 \times P_W \times t \times M_W} \tag{1}$$

where t is the sampling interval, P_1 and P_2 are pulses from EN1 and EN2 respectively, a is the diameter of the wheels, M_W is the number of magnets per wheel and P_W is the number of pulses detected by the encoder per magnet.

As for the angular velocity, it was derived from the IMU data. There are 9 16-bits messages sent from the IMU, however only the message from channel Gz of the gyroscope is needed since only angular velocity in the xy plane is of interest in this case, this angle is also called the yaw angle. As the range of the gyroscope is from -250°/s to + 250°/s [5], and the message was saved in a parameter which is signed 16-bits integrate, so the resolution can be calculated as:

$$resolution = \frac{range}{2^n} \tag{2}$$

Where n is the number of bits used to denote Gz.
Then the angular velocity ω can be calculated as:

$$\omega = GZ \times resolution = \frac{Gz \times (250 - (-250))}{2^{16}} \tag{3}$$

Where Gz is the decimal value of the message from channel GZ of gyroscope, .

After collecting the information of the translational speed and the angular velocity, the pose of car (x, y, θ) could be updated through the following formulas. The process is also shown in figure 7.

$$x_{k+1} = x_k + v \times cos(\theta_k) \times \Delta t \tag{4}$$

$$y_{k+1} = y_k + v \times sin(\theta_k) \times \Delta t \tag{5}$$

$$\theta_{k+1} = \theta_k + \omega \times \Delta t \tag{6}$$

The old pose of the car is (x_k, y_k, θ_k) while the new pose of the car becomes (x_{k+1}, y_{k+1}, θ_{k+1}). The variable v is the translational speed, ω is the angular velocity and Δt is time difference

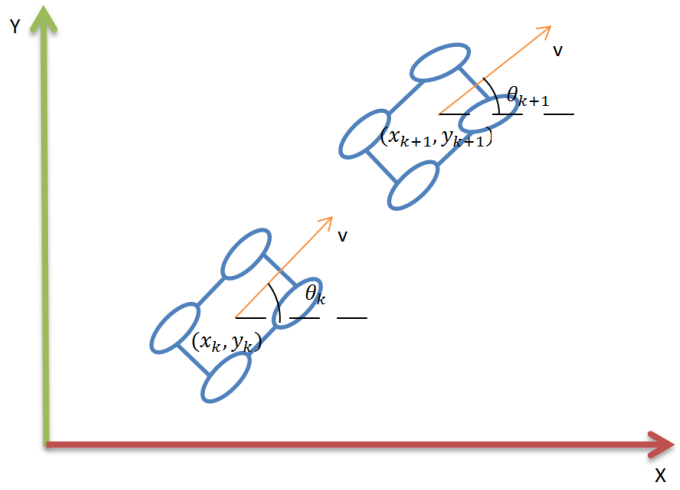


Figure 7: Image showing how to update pose information

between those two states.

To test if the odometry was right, a ROS package called Rviz was used in this project. It can show two coordinate frames in interface and also the path of how the car drove. This is shown in figure 8 and 9.

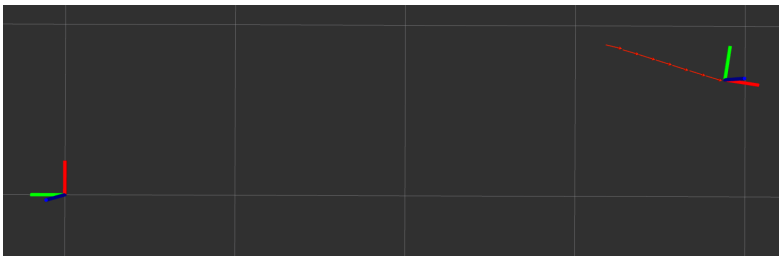


Figure 8: Image showing the map frame and the base link frame in rviz

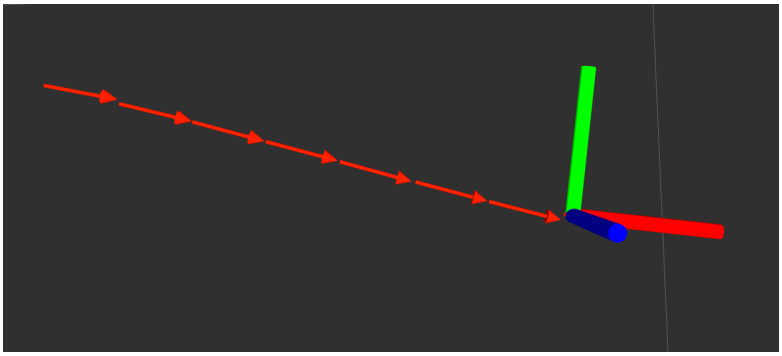


Figure 9: Image showing the path where the car passed through in rviz

3.2 Lane detection

In order for the car to follow a lane it must see it first. Seeing the lane is divided into two major steps, lane detection and lane estimation. Lane detection is about gathering evidence of features representing a lane, such as line segments, and lane estimation is stitching all of these features together to form the lanes.

As tools in the development for the lane detection and lane estimation a few libraries has been used. These libraries are OpenCV, GScam, PCL (point cloud library) and Eigen. OpenCV is a general image processing library which is used to a large extent in this project. GScam is used to handle the video feed from the camera, PCL is used to perform segmentation algorithms and Eigen is used as a general algebra tool.

Detecting lane features in the lane detection algorithm was done using the camera. The camera is capturing individual frames at a certain specified frequency and for each frame a ROS message is sent. A subscriber in the lane detection algorithm with a buffer of only one message will cause a callback to be executed. This means that the lane detection will run at the same frequency as the camera but will start to skip frames if the process is lagging behind. The very first step that was done in the lane detection when receiving the frame was converting the given image which is in ROS message format to OpenCV format which lets us use all of OpenCV's useful functions.

First generation lane detection

During this project two different lane detection and estimation algorithms were created. In the first one the image was converted to a gray scale image before a filter was applied that made the image binary followed by a Canny edge filter that found all the edges. A Houghlines function was then invoked on the edge image to detect all lines. It uses the hough transform to calculate the probability that the edges forms a line. Depending on the probability threshold, the minimum line length and the maximum distance between lines different lines will be found. Lines that was found but which were close to horizontal were removed.

The inclination of each line was calculated and then used to extend the line to the baseline. The point of each line at the baseline was then put in a vector and the two points which had the closest distance to the middle of the picture, where the car was assumed to be, were selected. If only one line was found then a default value was chosen as the distance to the other line. The difference in distance between the two points and subsequently the two lines was then sent to the PID control node to be able to compute a output value to the servo.

To be able to see the decision of the lane algorithm an arrow was drawn in the picture which indicated how much difference it was between the lines and in which direction so that the user could get a fairly good idea about how the car wanted to steer.The closest lines on both side of the vehicle were also drawn in the picture. Beside the steering value the number of lanes that the algorithm founds is outputted to the state machine along with the image that is published continuously to the GUI.

Second generation lane detection

For the more advanced lane detection and estimation algorithm a filter was applied to the image in order to enhance the line features symbolizing the lanes. This filter called a progressive line filter was used in one of the cars competing in the Darpa challenge for autonomous driving in the suburbs[9]. The filter blurs horizontal lines but keeps vertical lines which satisfy the line width criteria. Using the fact that the line width decreases in the camera perspective the

further the line is from the car we can make the filter change topology based on the row index of the image. At the top of the image the line width will be thin but towards the bottom of the image the line width will be thicker. In figure 10 the normalized filter topology can be seen.

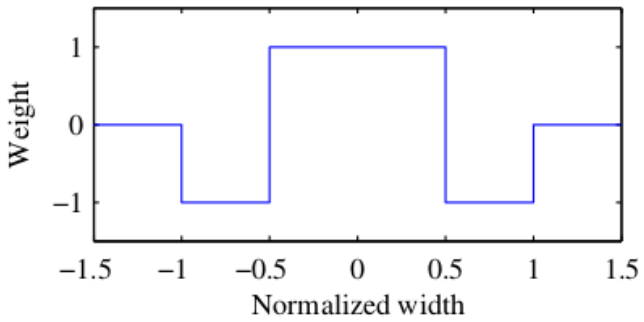


Figure 10: Visualization of progressive filter topology [9].

This algorithm was first written in Matlab for easy testing and modification. Every major step in the Matlab algorithm is shown in figure 12. The first image at the top left corner shows the input frame after it has been converted to gray-scale. At the top center is an image showing the progressive line filtering but it is hard to distinguish the pixel values because it is not normalized and the image is interpreting that a pixel value of 255 or more is full white. The filtering makes more sense after all the pixels below a certain threshold have been cleared leaving only the main lane visible. The threshold image can be seen at the top right in figure 12.

Using this filtered and cleaned up image we can extract key-points symbolizing possible points on a line. From the filtered image a specified number of rows are extracted. For every row clear local maximums are found and set to be key-points. Key-points having weak local maximums are ignored, the definition of weak is based on the global maximum. In figure 12 at the bottom left located key-points as well as the corresponding extracted rows can be seen. The next part was to calculate the line orientation for each key-point. This was done using the Eigen vector of the Hessian matrix. The Hessian matrix is a 2x2 matrix shown in equation 7.

$$H = \begin{bmatrix} F_{xx} & F_{xy} \\ F_{xy} & F_{yy} \end{bmatrix} \tag{7}$$

The Hessian matrix describes the local curvature of a function of many variables and in our case x and y were the variables. The elements such as F_{xx} is the second order discrete derivative of the x variable. These discrete derivatives were calculated using a Sobel operator also called kernel. An example of a 3x3 Sobel kernel for the first order discrete derivative of the variable x is shown in equation 8.

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{8}$$

In the lane detection algorithm the Sobel matrices were 9x9. The Sobel matrix was applied as a convolution, meaning it will start at the top left corner of the image and slide one pixel to the right for each calculation. One calculation means laying the filter on top of the image and for each corresponding element in the image the filter multiplies them together and sums up all multiplications. The sum will be the new pixel value which is saved in another image, see figure 11 for how this works. In order to get the second order derivative the filters were applied twice.

In the actual code this was done using an OpenCV provided function.

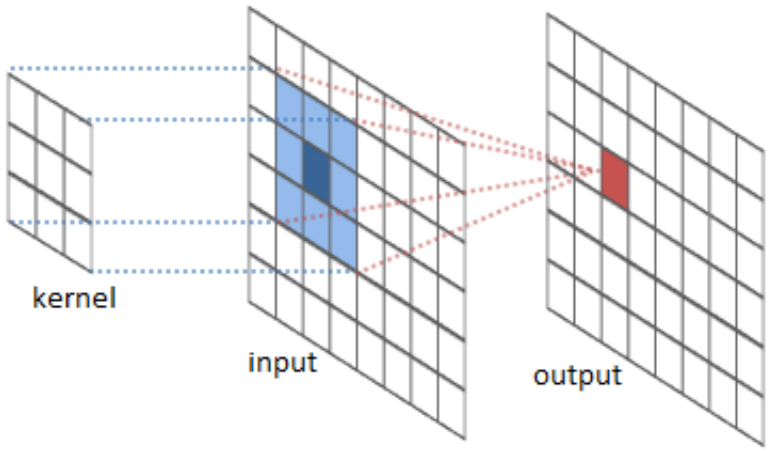


Figure 11: Image showing how a kernel is applied in order to do filtering [10].

The library called Eigen in C++ makes matrix computations easy and was used to calculate the Eigen vector. The orientation of the Eigen vector symbolizes the curvature of the line and is pointing towards the convex side of the line and with the use of some simple geometry this orientation was made pointing in parallel to the lines. In figure 12 at the bottom center image the orientations of each key-point is visualized.

Next step in the lane detection algorithm is to connect all of these key-points together in order to form clear line segments. This works by going through the key-points one by one and find valid connections. Assume the current key-point is one located in the middle of one of the lines. The algorithm then starts searching from where the so called path started and therefore checks the rows higher up in the image for a match. If a match is found the algorithm continues searching for the start of the path until no more valid key-points exist, when this happens the algorithm starts at the first key-point again, the one we began checking, and starts finding the end of the path. A match is only valid if the projected x coordinate and the angular displacement between the key-points are within certain bounds of the possible connecting key-point. Each path has a strength which symbolizes the reach of the path. The longer reach the path has the further away key-points can be connected. Every path has a starting strength but it will also grow stronger for each valid connection. By doing this even dashed lines could be formed into one single path. The lane connection algorithm was very robust, even with many outlining key-points originated from noise it will still form clear lines. The resulting paths were finally filtered such that short paths were removed. The final lines can be seen in the bottom right image in figure 12.

The final step in the lane detection algorithm is performing a so called bird-eye transformation. This transformation converts the pixel location into a real world position in front of the car. This can be done if the position and orientation of the camera is fixed and the ground is considered flat. Instead of dealing with the exact position, orientation and specifications of the camera two sets of four points was used instead. The first set of points corresponds to the four corners of the input image in real world measurements and the other set is the matching points but in the camera image in pixel measurements. See figure 13. These points were calibrated by looking if straight lines become straight in the lane detection algorithm and if the position of where the lines ends and starts were where they should have been. The transformed points were finally sent over ROS in a point cloud message over to the lane estimation algorithm.

The real algorithm was written in C++ and feedback images corresponding to the Matlab algorithm can be seen in figure 14. The average computation time of the lane detection for a

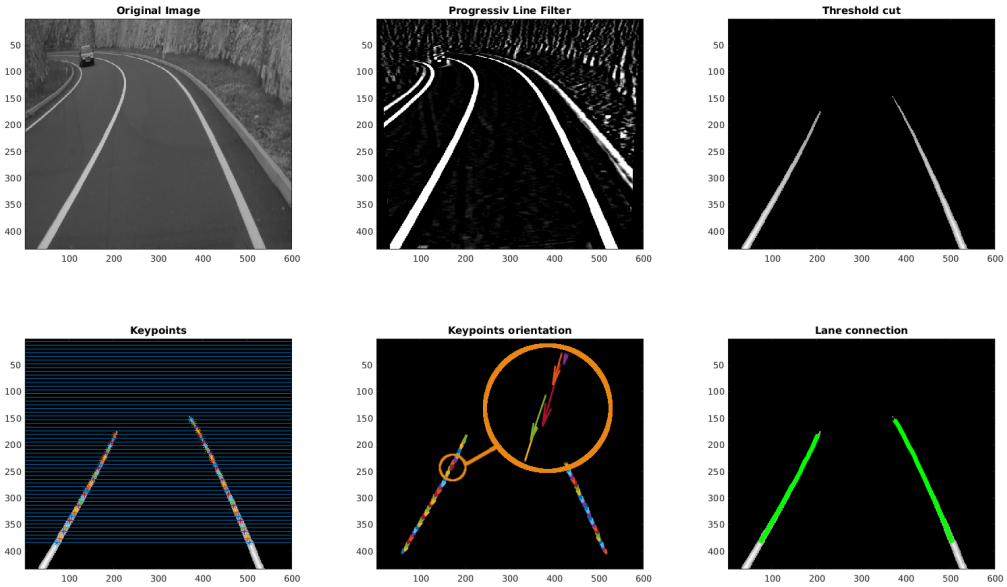


Figure 12: Image showing all steps in lane detection done in Matlab.

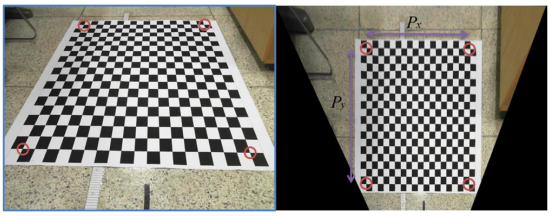


Figure 13: Image showing the principle of bird eye transformation [11].

single frame came out to be less than 2 milliseconds.

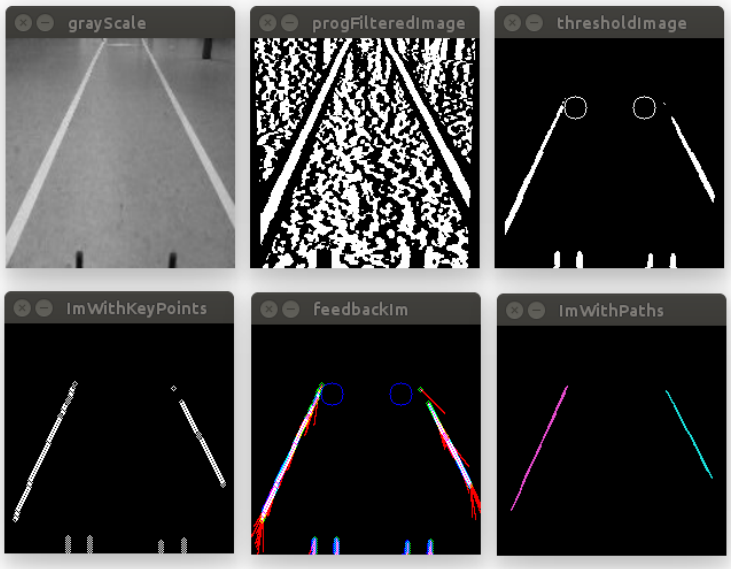


Figure 14: Image showing all steps in lane detection done in actual C++ program.

3.3 Lane estimation

The lane estimation is an essential part of the lane following as it deals with outliers and gives a better estimation of the true lane profile. The problem of having the lane following only relying on the lane detection output is that each frame and detection will not be an accurate representation of the probability distribution from where it was taken. As mentioned previously another important gain of using lane estimation is that it handles outliers, it will disregard detections which are not likely to be part of any tracking lane.

The first step in the lane estimation algorithm is a ROS callback for the lane detection output. Here all incoming point clouds containing the line features are gathered in a list of a preset size. The oldest detection will make room for the newest detection keeping the list to only contain a set of the newest observations. The main work is done in a timer callback called at a rate of 20 Hz. The first thing that is done in this callback is merging all points located in the list of line features into a new point cloud. This point cloud is relative to the fixed frame but is now transformed back into the cars coordinate system using the odometry information.

After the transformation euclidean clustering is done on the point cloud. This extracts clusters corresponding to the lines while at the same time removes cluster which are too small to be any line. The clusters are put in a list with the largest clusters on top. If at this point this is the first time the estimation is being done then a so called simple classification is done in order to find a lane to track. It will keep trying to find a lane to track until at least one, left or right line is found. This simple classification works by calculating the mean position of all the points in the cluster and if that position is to the left of the center of the car then it will be classified as part of the left line and likewise if the position is to the right of the car then it will be classified as part of the right line. Using these newly classified clusters an quadratic function is fitted to the data using the least square method. This simple classification is also done if the algorithm thinks it has completely lost the lane. While trying to find a starting lane a message saying to stop the car is sent to the motor controller.

Assuming that a lane is being tracked, e.g. either the left or right line is being tracked, then after the clustering each individual cluster is compared to the currently tracked lines. Considering the case of both left and right lines being tracked, then the cluster will be classified as one of the lines if the summed up error is small enough otherwise the cluster will be disregarded as a line belonging to another lane or just a false positive. The error is calculated by adding the error for each point in the cluster compared to the quadratic function of the corresponding line. Even though we explained the case when both lines are being tracked by the algorithm it will still work when only one line is being tracked. This works by mirroring the then tracked line to be the lost line. As an example assume that the right line is being tracked but that the left line is lost, this happens in the corners as the camera cannot see the inner left line. During the classification of the clusters the right line is being mirrored to the left side with the width of the lane being known beforehand. Now if a new cluster has a small enough error then it will count it as a part of the lost left line and will calculate the quadratic function.

The final step is done to calculate the car's latitude and angular displacement compared to the center-line of the lane. If only one line is being tracked then that line is shifted half a lane width in the latitude direction but if both lines are being tracked then an average of the two quadratic functions will be the center line. The estimated lane is also sent to Rviz in order to get feedback of what the car is thinking, a figure of how this looks like can be seen in figures 15 and 16.

3.4 Intersection and stop line detection

To detect a stop line/intersection the following approach was used. Start by converting the image from the camera to gray scale and then creating a binary image by applying a filter with a certain threshold for the bright pixels in the image. By using built-in functions in OpenCV horizontal lines could be detected. To remove noise in the image two morphology operators were

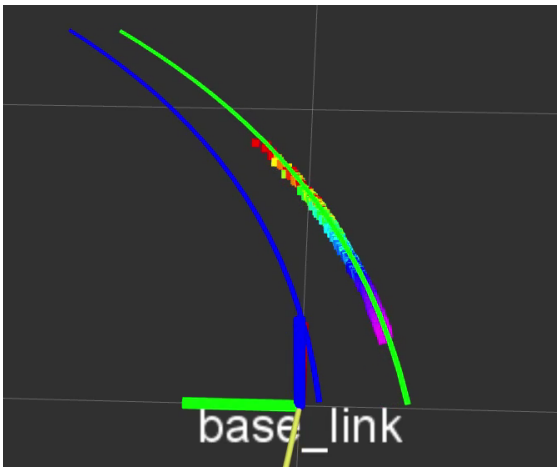


Figure 15: Image showing the visualization of the lane in a curve.

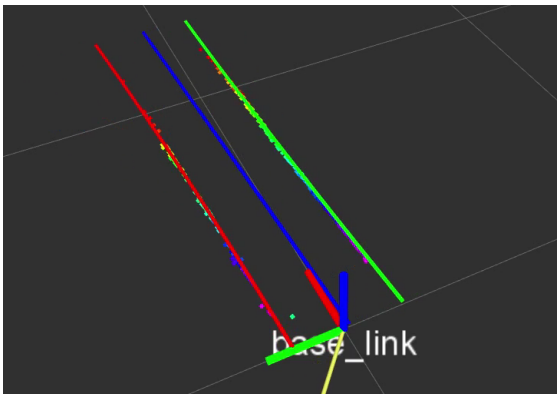


Figure 16: Image showing the visualization of the lane in a straight segment.

used; Dilation and Erosion. After all horizontal lines in the binary images has been detected the ones that form a rectangle needs to be selected. By applying a Canny edge filter to the image to be able to use OpenCV's findContours function. This function stores all the edges that form a contour in a vector.

Stop line

By calculating the number of sides each contour has and how their alignments are the ones forming a rectangle can be selected. Then iterate through each rectangle and see if their size and position in the image is within the threshold, if so it should be a stop line. When a potential stop line is found a rectangle is drawn on the image sent to the GUI and a message with distance to the stop line is sent to the Speed-Control Node. The distance is at the moment calculated by the amount of pixels there are from the bottom of the image to the rectangle. The result of finding a stop line can be seen in Figure 17.

Intersection

After the horizontal lines has been detected in the image and contours stored in a vector a bounding rectangle is drawn over each line. This makes it easier to compare length, height and position of each line. Start by iterating through the vector of lines and select the ones that has their left corner on the left edge of the image with some threshold. Save these lines in a vector.

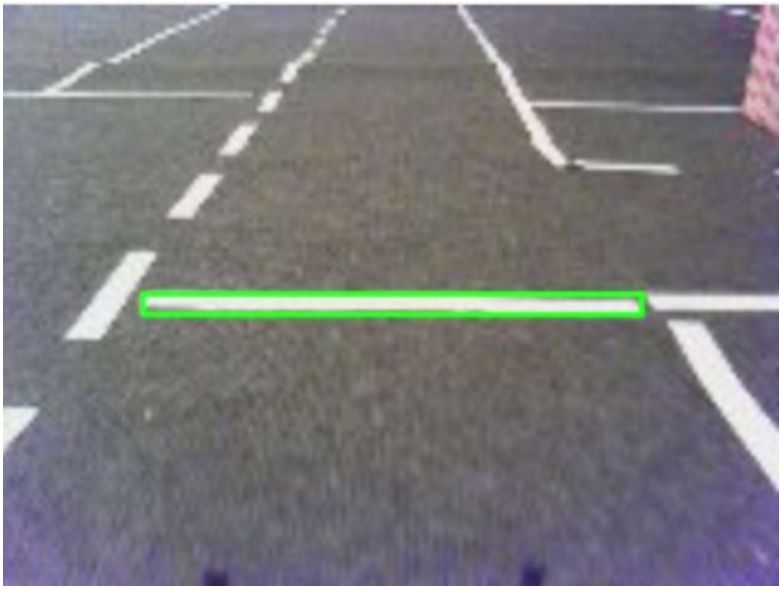


Figure 17: Stop line found on the testing track

Now iterate through the remaining lines and first find the lines with their right corner on the right edge of the image. If so compare the y-value of the line with the ones selected earlier, if they are similar and the length of both lines is above a certain threshold pair them together.

If the size of the vector for the paired lines is equal or larger than 2 check if any two pair is quite close to each other. If so draw the lines in the image and calculate the distance to the car. The distance is at the moment calculated by the amount of pixels there are from the bottom of the image to the closest rectangle. Due to the low resolution of the image from the camera the intersection detection did not fully work on the real track but it did however work well in the virtual environment. The result of the intersection detection in the virtual environment can be seen in Figure 18.

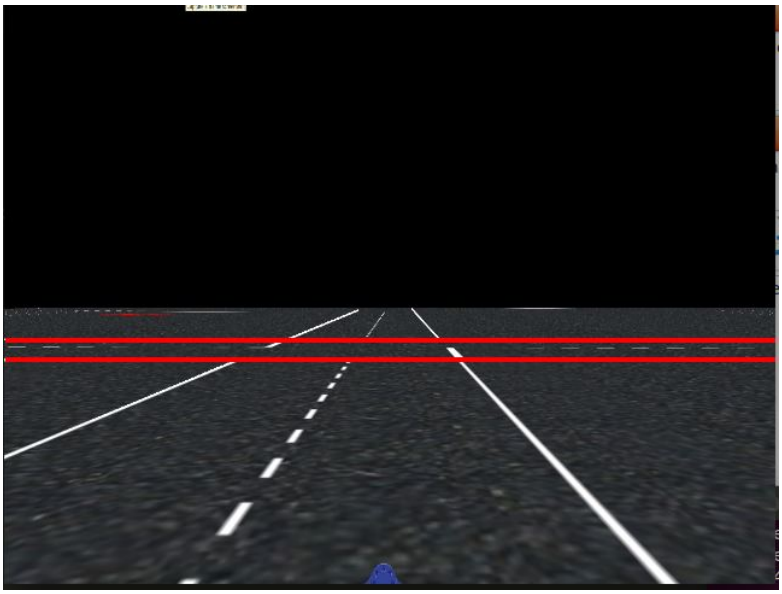


Figure 18: Intersection found in the virtual environment

3.5 Obstacle detection

For object detection data from the LIDAR was used. The LIDAR rotates 360 degrees and for every degree θ it stores the value of the distance d (in meters) to the closest solid matter in an array, resulting in a 360 element long array, figure 19. The LIDAR is mounted such that it gives 180 degrees when looking straight forward in the cars driving direction.

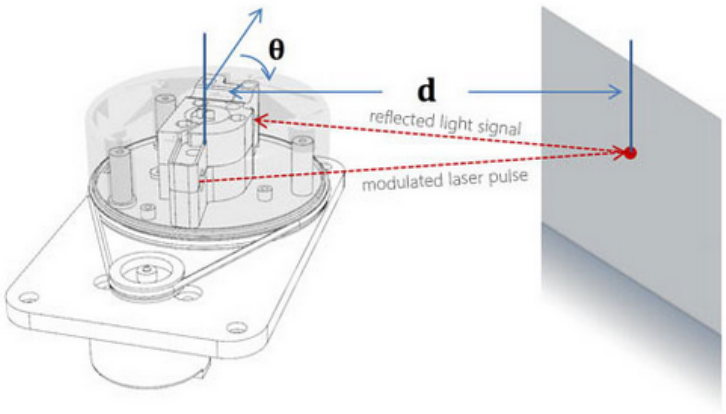


Figure 19: The LIDAR rotates and for every degree it stores the distance to solid matter, yielding a 360 element long array.

The resulting array is then used to check if there is an object in the car's lane. The first basic script of object detection only detects objects that are right in front of the car. Figure 20 shows

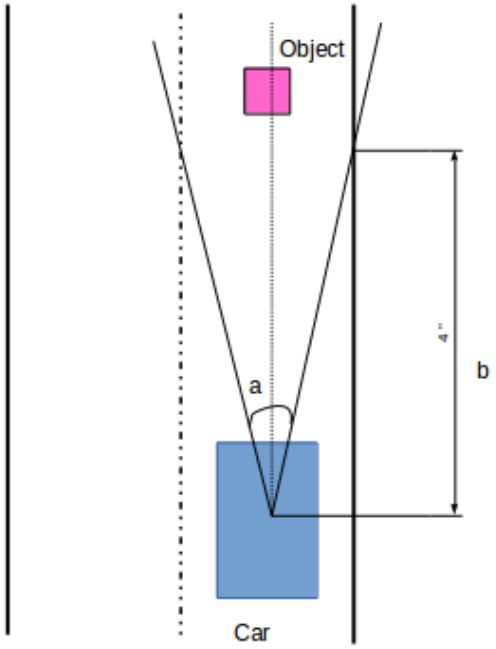


Figure 20: Region of interest for object detection is the lane the car is in, therefore angles covered by a is the only angles of interest. When a is set to 18 degrees it will fully cover the area of the lane straight in front of the car one meter from the car.

the car on the lane. We choose to look at angles $a = 20$ degrees, i.e. elements in the array stored

in place 170 to 190. This will cover the entire lane when the distance b is equal to 1 meter, i.e. all objects in the lane which are 1 meter away or more will be detected.

If the LIDAR array contains elements between 170 and 190 degrees that have a measured distance smaller than 1.5 meters it will store these elements in a new array and based on that array calculate the average distance to the detected object, the width of object and the angles between which the object was detected.

Possible extensions to this program would be to detect objects at all 360 degrees and then be able to determine if the object is on the road and in what lane, both for straight and curved roads.

4 Decision making

The decision making is handled by a state machine node on the motherboard. The state machine collects information from all the other nodes. With that information it then updates itself to the right states with a timer function that is set to run with the frequency of 20 Hz. The state machine consists of four concurrent state machines.

In the first one there is a total of four different driving states: idle, run, break, and stop. These are described in detail below and can be seen in Figure 21

1. Idle
   The idle state is the state that the vehicle is starting at. In this state the vehicle is waiting for information from the GUI and lane detection before it moves on to the running state.

2. Run
   The run state is the state where the vehicle is allowed to go forward at full speed. This state is entered when obstacles and stop lines either are too far away or not in sight for the vehicle, the lane is detected and data is recieved from all the LIDAR, camera and GUI. The maximum speed that is allowed is set by the GUI.

3. Break
   When the vehicle is in the break state it is supposed to perform a controlled breaking in which it slows down and stops at the desired distance from an object or a stop line. The breaking speed is calculated by taking the percentual distance left to the desired stopping spot times the current speed as seen in equation 9. Since the speed of the car is adjusted to the distance of the object it means that the vehicle is capable of following an object in front of it, like for example another car.

   $$v = \left(\frac{d}{D}\right) \times (v) \tag{9}$$

   where v is the velocity of the vehicle, D is the desired stopping distance from the object/stop line and d is the current distance to it.

4. Stop
   The stop state is entered when the vehicle is supposed to stop immediately. This happens when the lidar, camera or GUI stops responding, when no lanes are found or when the distance to a stop line or an object is too short for controlled breaking and therefore requires emergency braking. The vehicle also enters the stop state when the vehicle speed has reached zero.

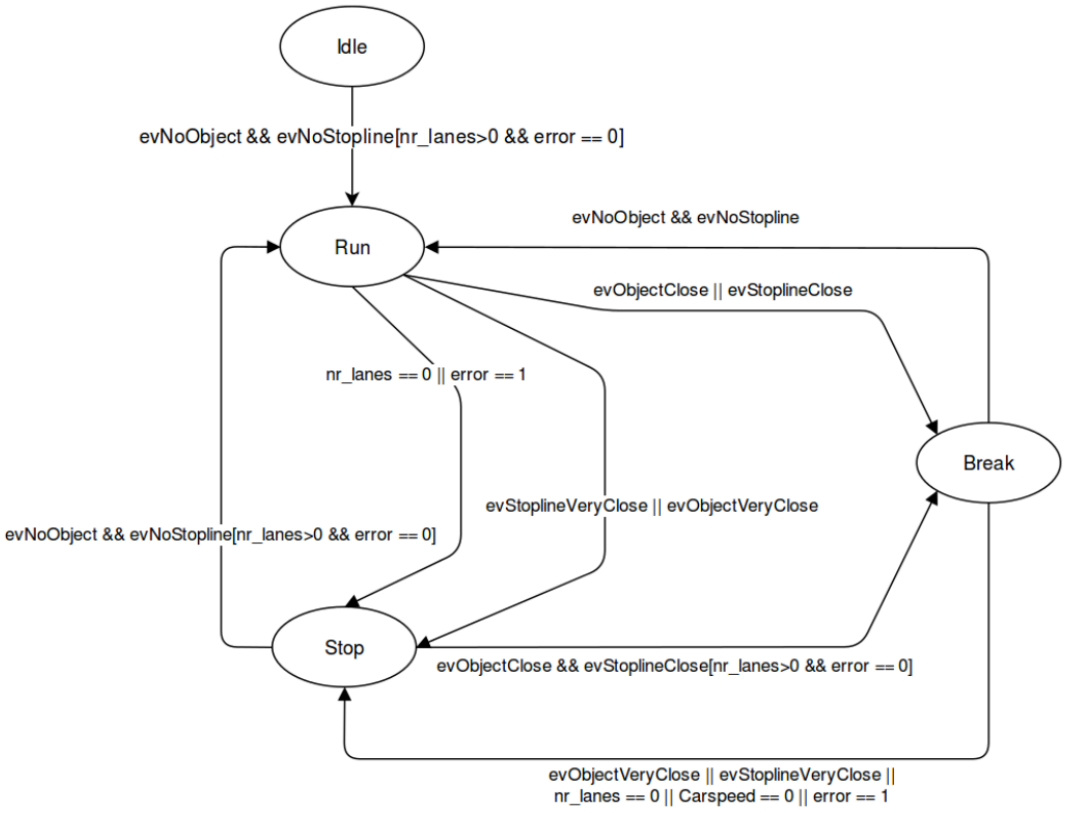


Figure 21: Statemachine of the different driving states.

The next state machine shows whether it is an object, stop line or something else that made the car stop and is shown in Figure 22 It is important if the car has moved to the brake or stop state. If it is an object that is in the way the vehicle should either drive around the object or stop in front of it and wait for it to be removed. This is controlled beforehand by a button from the GUI which is sent to the state machine. If it on the other hand is a stop line that got the car to stop then it should stop and wait for a decision from the GUI before it does something else. The two possible choices is to either drive straight forward or to turn to the left. If the car have stopped for some other reason than an object or a stopline then it will just stand still until the error is fixed and it can then transfer into the running state again.

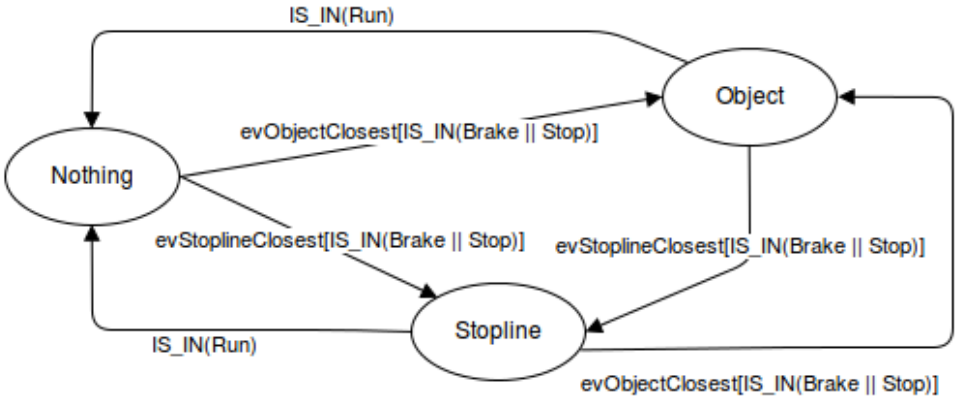


Figure 22: Statemachine of the different possible stops.

Lastly it also has two control states in manual and autonomous mode as shown in Figure 23. Those states indicate whether or not it is the car or the driver who has the responsibility to drive the car. It is determined by a push button via the GUI.

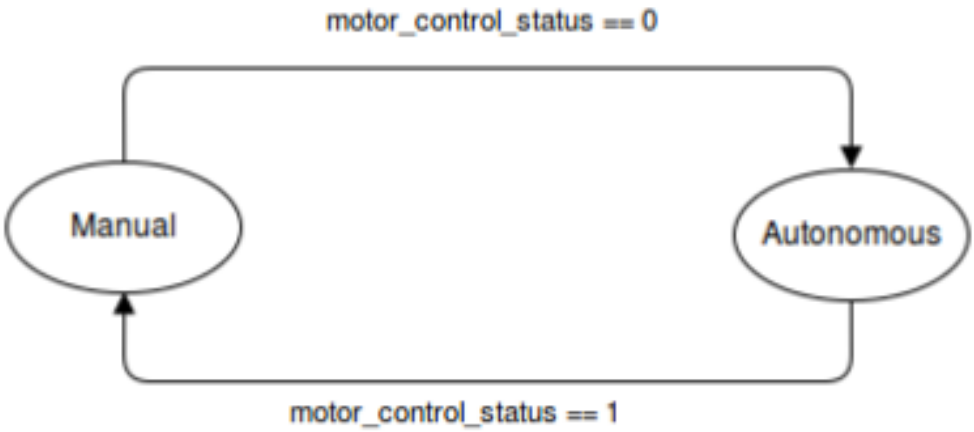


Figure 23: Statemachine of the control states.

5 Actuation

The actuation contains controlling of the motor, servo, buzzer and lights.

5.1 Motor and servo

The motor receives velocity commands directly from the state machine without any further calculations. The input to the servo is also directly from the state machine when the car is in manual mode. Except for being in manual mode when controlled over the GUI the car is also in manual mode in the following two situations:

1. Intersection handling, when the car approaches an intersection with a stop line it stops and wait for user command on how to steer. The user then inputs steering angle directly to the servo.

2. Obstacle avoidance, for obstacle avoidance the car can act in two different ways. Either it stops in front of the obstacle or it drives around it. For the latter scenario the turning angles are given from the state-machine.

Beside these two scenarios the input to the servo is given by a PID function in order to make the car steer smoother for lane following. The lane following needs to consider two factors; what is the lane displacement - i.e. how far away are the car from the center of the lane, and what is the turning angle displacement - i.e. the difference between the roads turning angle and the turning angle of the car. A PID controller has the following mathematical expression, equation 10:

$$u(t) \quad = K_P \cdot e(t) + K_I \cdot \int_0^t e(t)dt + K_D \cdot \frac{de(t)}{dt} \tag{10}$$

Where K_P, K_I and K_D are parameters to be tuned for good performance and $e(t)$ is the error to be minimized.

This was implemented as two separate PID controllers, one for angle displacement and one for center line displacement, where x is either angle or displacement, equation 11:

$$\begin{aligned} u_{Px}(t) &= K_{Px} \cdot e_x(t) \\ I_x &= I_x + I_x \cdot dt \\ u_{Ix}(t) &= K_{Ix} \cdot I_x \\ D_x &= \frac{e_x(t) - e_x(t-1)}{dt} \\ u_{Dx}(t) &= K_{Dx} \cdot D_x \\ u_x(t) &= u_{Px}(t) + u_{Ix}(t) + u_{Dx}(t) \end{aligned} \tag{11}$$

A maximum and minimum value for $u_x(t)$ was set in order to avoid anti windup. The parameters K_{Px}, K_{Ix} and K_{Dx} was tuned over the GUI to enable good steering.

5.2 Lights and buzzer

The vehicle is equipped with a buzzer, front lights, brake lights and blinkers front and back. The buzzer goes off when there is an error in the system. The front lights are automatically turned on when the car is started. Brake lights are turned on when the car is in braking mode. At this moment the blinkers are enabled but not in use. The idea is to turn them on for right and left turn.

Figure 24: Front lights and blinkers mounted on the vehicle.

6 Utilities

One of the requirements for the ELAD project was that it should be easy to continue improving the prototype for someone outside the project group. The following chapter describes the measures which have been taken to ensure this. Besides the measures described below TurtoiseSVN have been used for document management and Jira have been used for managing the scrum process.

6.1 Simulations

To be able to test different algorithms such as lane detection and stop line detection without having to test it on the real car, which can be time consuming, a virtual test environment was created in Matlab and Simulink. This was possible due to the integrated support for ROS in Matlab. This also mean that there is no need to write separate Matlab programs to test certain algorithms, one can directly write them in C++ and still use it together with the virtual environment created in Simulink. By using the 3D World Editor in Simulink a test track could be created. A basic CAD model of a car was created and imported into the 3D World editor. A camera view is placed in the cars local coordinate system which makes it possible for it to act like the camera on the real car. Steering and motor control was added to the car by applying it to the cars local coordinates. These were then transformed with a rotational matrix to global coordinates which makes it possible for the car to move in the world. No real dynamics was added to the simulated car other than the turning radius of the real car.

The Simulink program subscribes to the turn direction and motor speed nodes. The simulation acts like the camera in the system and therefore publishes to the same topic the camera on the real car would normally publish to. A real-time block was added to the simulation to make sure the camera frame rate stayed at the chosen frame rate. A screenshot of the camera view of the virtual environment is shown in Figure 25.

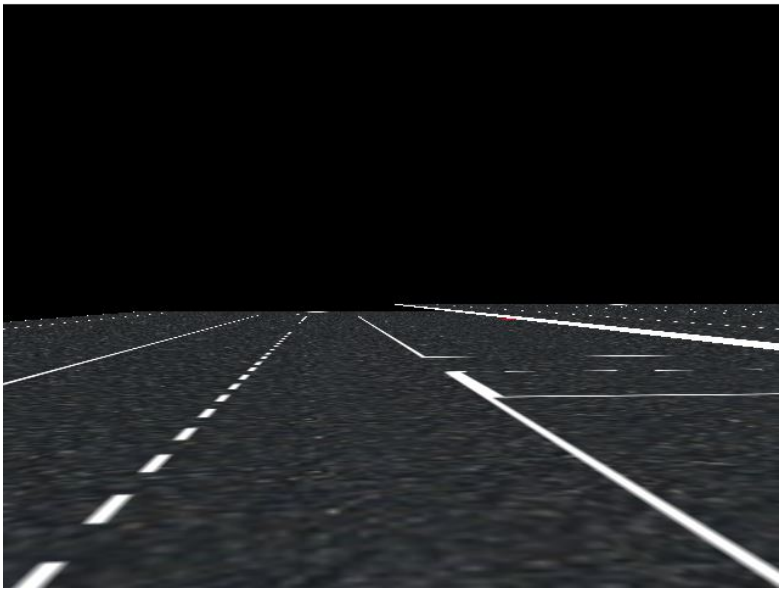


Figure 25: The road seen in the virtual environment

6.2 GUI

The GUI for the system was created to be able to monitor and control what happens in the car. It makes it possible for the user to see the raw camera stream from the car and also switch between different views like lane detection, stop lines and objects. The GUI contains a

log viewer which displays in text form the decisions that happens in the car and also displays relevant information for the user. The log data is stored locally on the host computer in a date stamped text file. The GUI also shows the status of each component in the car such as the LIDAR, camera, vehicle speed and battery level. It is also possible to take full control over the car for when it is needed. A screenshot of the GUI is shown in Figure 26.

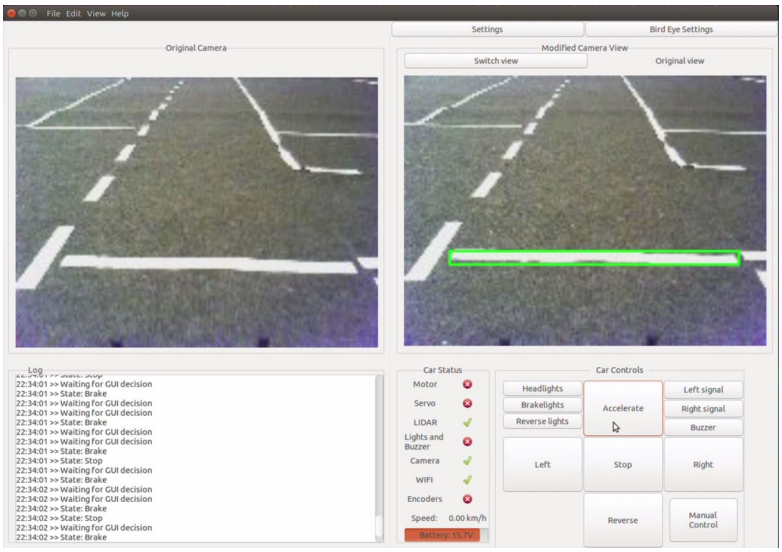


Figure 26: The GUI view when detecting a stop line

The GUI also makes it possible for the user to change parameter in different nodes during run-time. The settings window is accessed by pressing the settings button in the GUI. To be able to enable/disable certain functions in the car makes it easier to test certain functions and scenarios without the risk of other interfering functions. A screen of the settings window is shown in Figure 27.

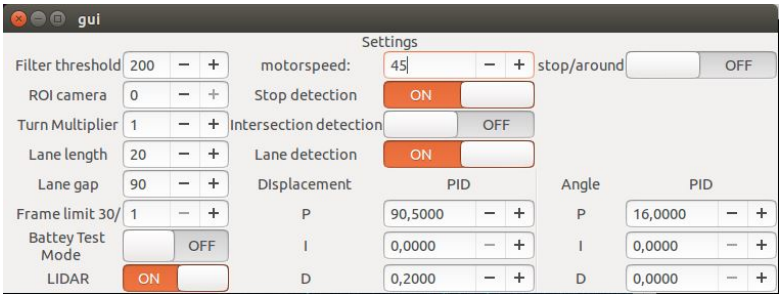


Figure 27: The different parameters that can be changed in run-time

The GUI was created using GTKMM which is the C++ interface for the GTK+ framework. The layout of the GUI was created in Glade User Interface Designer which simplifies the process of creating the layout. It creates a XML-file which then is interpreted in the C++ code and loaded dynamically. This means it it possible to change layout settings in Glade without recompiling the code. GTKMM was chosen as framework for the GUI due to it made it easier to combine it with ROS which also is written in C++. The GUI subscribes to many different topics in the ROS network including topics such as the camera image, sensor status and log text. The GUI also has many ROS publishers. Each button in the GUI is a unique publisher while the settings values and toggle buttons are distributed into different relevant publishers. The settings are check once each second and if something has changed from the previous check it publishes the new settings. This is to make sure the settings are always updated on the car and also not to take up unnecessary bandwidth on the network. The GUI is run on the host computer and

connected to the ROS core on the car via WI-FI.

6.3 Doxygen

Doxygen is the de facto standard tool for documenting C++ code [12]. All source code has been documented using this tool. The focus of the documentation is to describe what each class and method does and what inputs and outputs a method receives. If there is anything left to be done in the source code the documentation also describes what needs to be done and where this action is needed. By typing comments in a specific way in each code file Doxygen generates HTML files which structures all the code in a way to get a good overview of all the classes and functions in a project and also makes it possible to learn their functionality.

6.4 Project Documentation

A part of the project is to document how everything has been set up and how everything works so that it is possible for someone else to more or less replicate the project. We have therefore created a local web page where we describe how to assemble the car, how to install all the necessary software and also explains how each module works. It was created with HTML and CSS to make easy to use and available on all platforms. Some material from the web page can be found in the appendix.

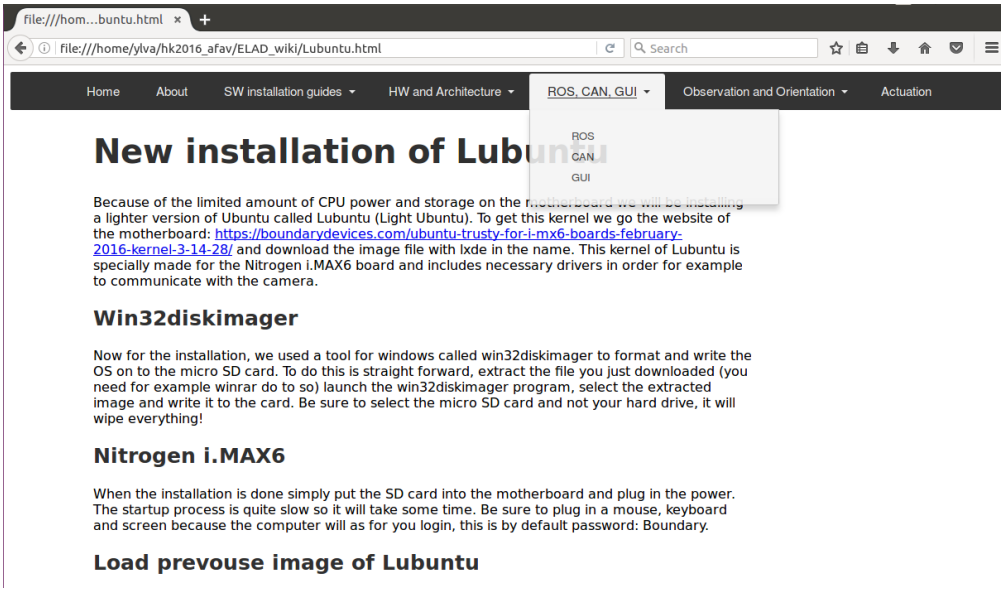


Figure 28: Project documentation page

7 Results

13 different test cases were formed from the set requirements and detailed explanations of each can be found in Appendix C. All test cases have an ID in which the first number is the number of the requirement it corresponds to. This chapter will go through each test case and state the outcome from the testing.

Stay within lane for 5m (ID: 13.01)

It managed to stay within the lane for 5 m.

Vehicle stops when loosing track of the lane (ID: 24.01)

The vehicle stopped 8/10 times when loosing track of the lane.

Stopping the vehicle via the interface (ID: 31.01)

The vehicle stopped every time the stop button in the interface was pressed.

Turn the vehicle using the GUI (ID: 31.02)

When pressing the button for turning left the vehicle turns left. When pressing the button for turning right the vehicle turns right.

Accelerate to max allowed speed when no obstacle (ID: 12.01)

The car accelerated to the maximum set speed when no obstacle was in front of it.

Drive 5 m in S curved lane (ID: 14.01)

Due to the bad turning ratio of the vehicle the initial S-curve was impossible to perform. However the vehicle drives through a wider 5 m long U-shaped curve with an outer radius of 1.8 meters while staying inside the lane.

Detect obstacle at maximum distance while stationary (ID: 15.01)

The vehicle detected obstacles at the set maximum distance of 1.5 m while stationary.

Detect obstacle at minimum distance while stationary (ID: 15.02)

The vehicle detected obstacles at the set minimum distance of 0.2 m while stationary.

Stop in front of object (ID: 16.01)

The vehicle stops at the desired distance from the objects at speeds up to 80% of the maximum speed.

Controlled deceleration for object (ID: 18.01)

The vehicle speed decreases as the vehicle approaches the obstacle.

Controlled deceleration for stop line (ID: 18.02)

The vehicle speed decreases as the car approaches the stop line.

Emergency braking (ID: 29.01)

The vehicle speed is set to 0 when an object appears closer than the emergency distance which is set to 30 cm.

Stop if sensor communication is lost (ID: 36.01)

When the GUI is turned off during driving the vehicle enters the stop state and stops.

Out of 54 requirements set for the project, where 18 was ranked as priority 1, 17 as priority 2 and 19 as priority three (where 1 is highest and 3 is lowest priority) 36 requirements were fulfilled. An additionally 7 requirements were partly met. Out of the priority 1 requirements only 4 wasn't fully met. In appendix XX all requirements and whether they've been fulfilled or not are shown. Requirements marked as green are fulfilled, yellow are partly fulfilled and white is not met.

8 Discussion and Conclusion

One design decision that we made in the beginning of the project was to use the hardware platform that we received from ÅF. We based that decision on that ÅF knew what they were doing and was confident in their own platform. We also did some basic research in the spring about the requirement on some components like the camera and the motherboard on the car. It was however hard to really know what was needed for the project and since there was no time to actually do anything with the components we decided to keep them as they were.

During the project we realized that the Nitrogen6 MAX had its limitations in both the operating system, it uses a version of Lubuntu, and computation power. Lubuntu is quite limited as an operating system which made it difficult for us to install certain packages which would have helped us. We also had to move the computation of the lane detection code from the motherboard on the car to a separate computer and had a lot of trouble getting the drivers for the camera module to work with ROS.

The choice of using the STM32 card was the right choice when we took that decision in the project. It made it possible for us to focus on the autonomous part of the project which is what ÅF really wanted. The biggest disadvantage of using the STM32 card is that it is custom made by ÅF and if anything was to happen it would take time/be hard to replace. Unfortunately, that is exactly what happened. One month before the end of the project the STM32 card caught fire for unknown reasons and we therefore had to borrow a copy of the card from the ÅF office in Linköping who also worked on the same platform. It would be better to use an of-the-shelf microcontroller like an Arduino or something similar. Even though the STM32 card saved us a lot of time in the project we had trouble understanding it sometimes due to the lack of documentation and difficulty in modifying the code on it. No comparison between an Arduino Uno and a STM32 microcontroller has been done by us so we do not know if it is capable of doing the same tasks as the STM32 one but it is definitely worth taking a look at it.

Since the Nitrogen6 MAX was not powerful enough of computing the lane detection and lane estimation in the required time frame we had to move the computation to the computer running the GUI. Moving that computation was no problem thanks to ROS, we just had to start the node on the host computer instead. When we moved it we feared that the latency due to transmitting the video stream over Wi-Fi, do the computations and then send back the results to the car would be too long for it to work. This was however not an issue due to the low resolution chosen for the video stream (160x120 pixels), we even managed to capture the video stream in 60 FPS, do the calculations on the host computer and actuate the car without a latency that would decrease the performance of the car. We however tried with a slightly higher resolution for the camera (320x240) which clearly improved the results of both the lane detection, stop line detection and actually made the intersection detection work, but this also meant four times the needed bandwidth over the Wi-fi and this increased the latency too much. The increase in pixels also increased the computation time of each image algorithm which also affected the time it took for one computation cycle. Another aspect of having computations off-board is that we needed to rely on a working Wi-Fi connection. This meant that we had to make sure to stop the car whenever we lost connection to the nodes on the host computer. It would have been better to either have a more powerful motherboard on the car or have multiple computers on the car communicating with cables instead of Wi-Fi.

A big issue we had with the chosen RC car for the project is that the steering angle of the car was very poor and non-symmetrical. The steering ratio was around +-45 degrees. With non-symmetrical we mean that the maximum turn angle to the left is higher than the maximum turn angle to the right. This affected the steering performance depending on which way the car had to turn. This forced us to redesign our original design for the track and also limited us from turning right in an intersection. Another problem we had with the car was the chosen DC motor. When the RC arrived it was equipped with a very large and powerful DC motor but

was replaced with a quite small DC motor. The motor was replaced because they had no need for the large motor and wanted a motor that could be controlled very exact. This motor had a gear box with a ratio of 1:20 and was used from September to mid-November. That's when the gear box broke. We had a replacement motor of the same model but with a gear ratio of 1:50 that was installed. The gear box on that motor broke the day after the final presentation. The motor should be replaced with a motor with a larger and more robust gear box.

For the vehicle functionality scripts there are some limitations. The lane detection algorithm can't tell the difference between a dotted line and a solid line. Furthermore, if no lanes exist there is a risk that the algorithm will find some other line in the image and interpret it as a lane causing the vehicle to move when it in fact should stand still. There have been some problems for the vehicle to keep itself in the middle of the lane and we believe this is because of the bad functioning of the servo together with the unbalanced steering. This also causes the vehicle to sometimes cross the outer line for the lane, which could have been fatal in real life applications. The stop line detection algorithm interprets the chassis as a stop line, this should be fixed.

Another problem that was encountered during the testing was that the camera quickly got warm. At the end of the presentation day at KTH, when the camera had been in use for several hours, a delay in sending images could be noticed, with the camera freezing an image sometimes. We believe that the overheating from this day caused the camera to break down, after the presentation day the camera never functioned as supposed anymore.

Even though there were some physical limitations the vehicle still managed to fulfill most of the important requirements and with some changes it could prove to be a good learning platform for autonomous driving.

9 Future work

The vehicle is meant to work as a platform for learning about and testing autonomous functions. This chapter will go through suggestions for future work that would further enable this.

Regarding hardware the camera and the motor needs to be replaced, they got worn out in the end of the project. The servo of the car also needs to be fixed. It is not proportional and yields a steeper steering to the left than to the right for the same steering angle.

The setup with the Nitrogen and the STM-32 card have not worked to full satisfaction. Replacing the Nitrogen to a motherboard able to run full Ubuntu and the specially made STM-32 card to some of-the-shelf card would facilitate further developments, making the platform easier to work with.

There were some requirements for this project regarding data logging that haven't been prioritized. Instead ROS bags have been used and they have worked satisfactory for offline testing of vehicle functionality.

When it comes to further improvements of the vehicle functionality it is recommended to work with creating a SLAM/positioning algorithm. The IMU is now drifting away in its measurements, which needs to be fixed before implementing SLAM. The functions could also be structured in a library of important functions such that they easily can be used as a foundation for further implementations.
Today the vehicle displays error code signals when something breaks down, implementing warning code signals would also be good.

Finally the functions discussed in the Discussion can be improved to overcome the problems stated there. Code to enable blinking the blinkers could also be implemented.

References

[1] H. Cheng, *Autonomous Intelligent Vehicles, Theory, Algorithms and Implementation.* London: Springer, 2011.

[2] Robopeak. (2014) Rplidar. Retrieved 2016-12-07. [Online]. Available: http://www.robotshop.com/media/files/pdf/datasheet-rplidar.pdf

[3] TRULY. (2009) Specification. Retrieved 2016-12-07. [Online]. Available: http://www.mouser.com/ds/2/607/CM8206-A500SA-E-242778.pdf

[4] FingerTech. Fingertech "gold spark" 16mm gearmotor. Retrieved 2016-12-07. [Online]. Available: http://www.fingertechrobotics.com/proddetail.php?prod=ft-spark16

[5] ——. (2014) Mpu-9250 product specification revision 1.0. Retrieved 2016-12-07. [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/5/5/0/MPU9250REV1.0.pdf

[6] Retrieved 2016-12-07. [Online]. Available: http://lubuntu.net

[7] (2014) About ros. Retrieved 2016-12-07. [Online]. Available: ros.org/about-ros

[8] (2014) Ros 101: Intro to the robot operating system. Retrieved 2016-12-07. [Online]. Available: http://robohub.org/ros-101-intro-to-the-robot-operating-system/

[9] A. S. Huang. (2010) Lane estimation for autonomous vehicles using vision and lidar. Retrieved 2016-12-13. [Online]. Available: http://rvsn.csail.mit.edu/Pubs/phd_ashuang_2010feb_laneestimation.pdf

[10] C. Olah. (2014) Understanding convolutions. Retrieved 2016-12-13. [Online]. Available: http://colah.github.io/posts/2014-07-Understanding-Convolutions/

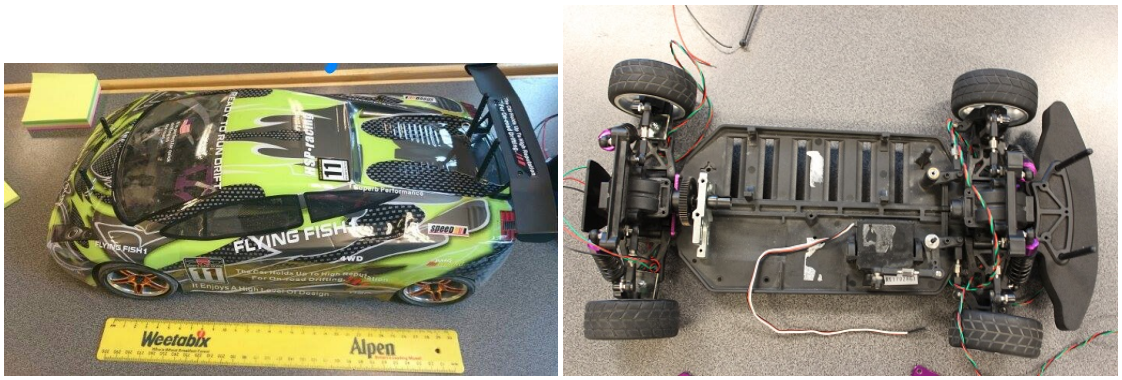
[11] C.-H. H. CJeisung Lee and M. Park. (2013) A vision-based automated guided vehicle system with marker recognition for indoor use. Retrieved 2016-12-14. [Online]. Available: http://www.mdpi.com/1424-8220/13/8/10052/htm

[12] Doxygen. (2016) Doxygen. Retrieved 2016-12-07. [Online]. Available: http://www.stack.nl/~dimitri/doxygen/

Appendix A

Assembly of car

1. **Disassemble car**
   Start by disassembling the original car, figure 29a, so it looks like figure 29b.



(a) RC car (b) RC car disassembled

Figure 29: The original car and the car after everything had been disassembled.

   When disassembling the car, make sure to remove the drive axes for the front wheels. One of these will be used to mount the motor.

2. **Motor**
   In order to mount the motor you need to 3D-print or buy an adapter from 3mm shaft to 5 mm. Then use one of the drive axis that was removed in the earlier step and the mount the motor as illustrated in figure 30.

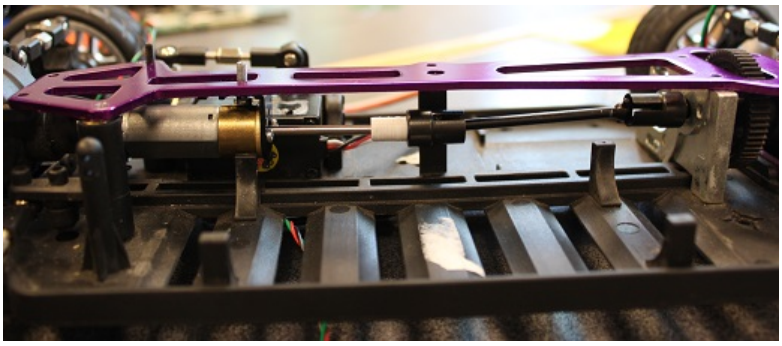
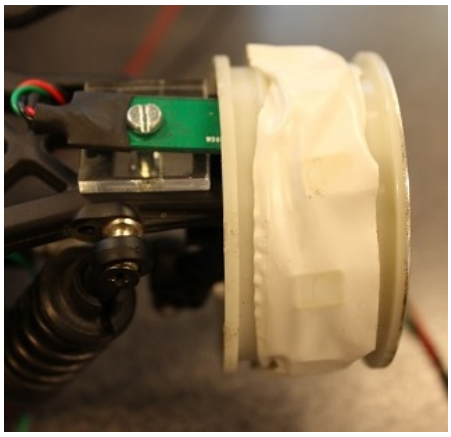


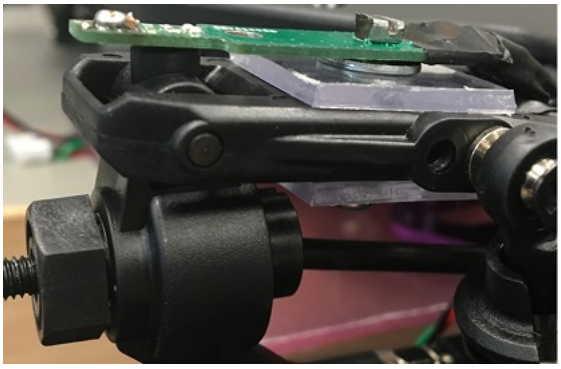
Figure 30: Mount the motor, the white part seen to the left in the figure is a 3d printed part, the rest are parts from the disassembled car.

   The motor mount will be screwed together on the same plate that the LIDAR will be mounted on. Leave it unscrewed for now. Make sure the drive shaft is inside both of the black cylinders. As you can see on figure 30 the one to the right is on the edge of the cylinder, it won't be like that when the mount is fastened on top plate.

3. **Encoders**
   Remove the rubber from the tire and tape the 10 magnets in an even distance between each other around the wheel as shown in figure 31a.The encoder is fastened on a small acrylic plastic plates on the back wheels as shown in figure 31b.



(a) Wheel magnets taped around the wheel.



(b) The wheel magnets are fastened on the wheel with tape.

Figure 31: The encoder is fastened on a small plastic plate on the back wheels.

One of the front wheels remove is a little bit tricky. Cut off the small plastic cylinder which can be seen in figure 31b. That part is in the way for the encoder. Now screw the encoder in the hole behind the cut off cylinder hole. It can be hard to tighten, some glue could be used to keep it steady. Now put the rubber back on the wheels.

4. **LIDAR** Laser cut an acrylic plastic plate so that the LIDAR can be mounted.
   (a) Start by screwing one corner of the LIDAR on the plate
   (b) Screw the motor mount on the plate so it's secured
   (c) Screw the screws to tighten the purple metallic plate and the plastic plate together on the car
   (d) Now screw the remaining screws for the LIDAR.

   The car should now look like figure 32.

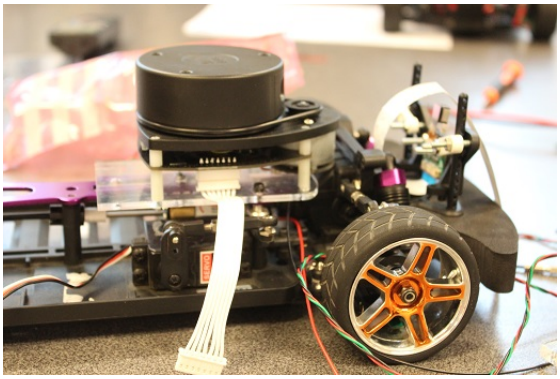
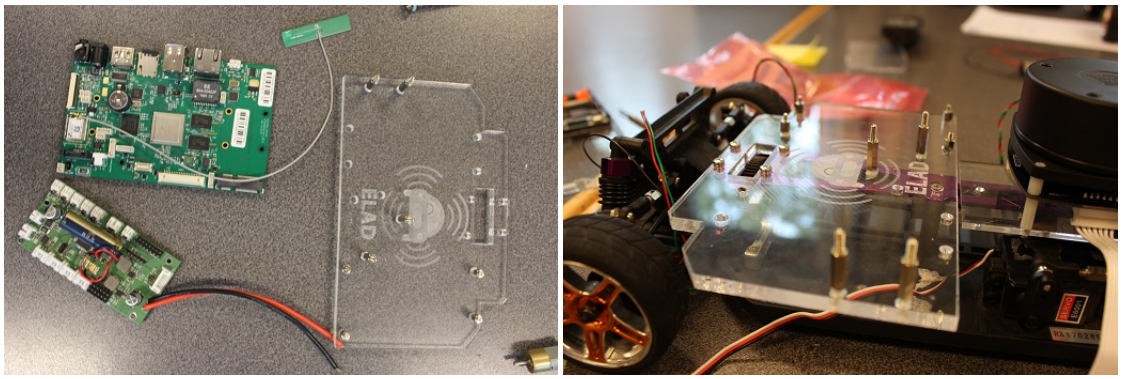


Figure 32: LIDAR

5. **PCB:s**
   Laser cut an acrylic plastic plate that will be used to mount the PCB and the Nitrogen 6Max motherboard, figure 33a. [DRAWING FOR PLATE] Use spacers to mount the cards. Short ones for the PCB and longer ones for the Nitrogen 6Max motherboard. Mount the plate on the car before fastening the PCBs, figure 33b.



(a) PCB cards (b) Mounted plate.

Figure 33: Mount the laser cut plate on the car and thereafter the PCB:s.

6. **Camera**
   In the beginning the camera was mounted like in the figure 34. After some time it was realized that the camera was too unstable to have a usable video feed. To solve this a

Figure 34: camera

camera mount specific for the camera head was designed and 3D printed. It clearly reduced the shakiness of the camera. The result can be seen in figure 35. On the part where the camera is mounted LEDS are also installed to work as headlight and flashers.

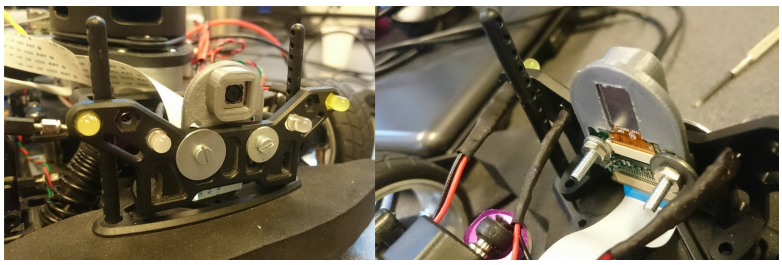


Figure 35: Camera head for stabilizing the camera

Later in the project when testing of the lane detection algorithm was ongoing a new camera head was made. The new camera head was mounted behind the LIDAR and gives the camera a better view of the road. It also enables tilting of the camera in different angles relative to the road. In figure 36 the final camera mount is viewed.

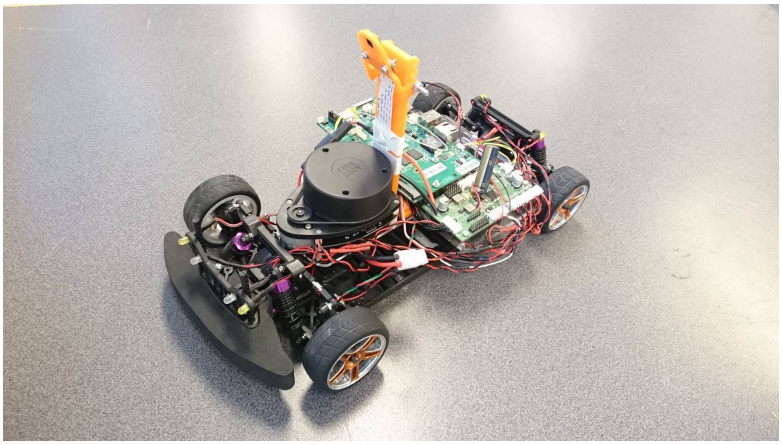


Figure 36: Final camera mount

7. **Cabling**
   When everything is fastened solder/crimp connectors to all encoders and LEDs. The LIDAR cable will need to be extended and equipped with different connectors.

8. **Chassis**
   To give the car a nicer look the original chassis was altered. Wholes for LIDAR, camera and lights were made and the chassis was spray painted in a gray nuance, see figure 37

Figure 37: The original chassis was altered to suit the car and painted in a gray nuance.

Appendix B

Authors	Date	Doc. ID	Approved by
Carl-Johan Sjöstedt carl-johan.sjostedt@afconsult.com **Jan Schifferdecker** jan.schifferdecker@afconsult.com **Xavier Dreux** xavier.dreux@afconsult.com	2016-06-30	ÅF-TECH-	
		Version PA1	**Information class** Internal

Requirements for ADEPT project

Contents

Purpose

ADEPT (Autonomous Driving Education PlaTform) is a competence project which aims to increase the knowledge about the design and inner workings of an autonomous car and its surroundings. To achieve this goal, a scale 1:10 car that is equipped with sensors well known from the real autonomous drive industry would be used.
A partnership between ÅF and KTH has been created and KTH would help ÅF to achieve this goal.

Definition

- A *straight double-way track* is defined by one straight solid line to the right and one straight dashed line to the left 30 cm apart. Another solid line on the far left (meeting lane) should also be 30 cm apart from the dashed centre line.
  See figure 1 - track number 1.
- A *S-shaped double-way track* is defined by one S-shaped solid line to the right and one S-shaped dashed line to the left 30 cm apart. Another solid line on the far left (meeting lane) should also be 30 cm apart from the dashed centre line.
  See track figure 1 – track number 2.
- A *8-shaped double-way* track is defined by a 8-shaped double-way track with a “simple” intersection in the middle.
  See figure 2 - track number 3.
- Safety area is defined by the distance around the vehicle needed either for stopping the vehicle in front of an obstacle or for avoiding side and back collisions with other vehicles.
  See figure 3.

Requirements

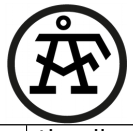
General

Elad-01	The vehicle should convey a ÅF feeling. The vehicle shall be used for official ÅF presentation	Prio 3
Elad-02	The vehicle shall be built in such a way that adding simple sensors such as buttons should be easy.	Prio 3
Elad-03	Implementation shall be done on-board/off-board. *Not Decided yet*	
Elad-04	A usable library of important functions shall be written which can be used as a foundation for further implementation. (a) Function such as sensor data acquirement and data processing shall be primarily focused on.	Prio 1
Elad-05	Vehicle shall be able to receive commands and send sensor data to a host computer after turning on the system.	Prio 1
Elad-06	When the host computer for the vehicle sends an order to the vehicle, the response time shall be as fast as possible.	Prio 1
Elad-07	The vehicle shall be capable of working during 15 min normal use.	Prio 1
Elad-08	The system shall give a warning when the remaining running time of the vehicle is less than 5 minutes.	Prio 3
Elad-09	The vehicle should shut down if no further orders are received from the host computer after 5 min.	Prio 3
Elad-10	The vehicle shall have an error code signal capable of informing that an error has occurred and if so what type.	Prio 2
Elad-11	The vehicle shall have a warning code signal capable of informing that a warning has occurred and if so what type.	Prio 2

Vehicle functionality

Elad-12	When there is no obstacle in the safety area, the vehicle should accelerate to the maximum speed of that current road type.	Prio 2
Elad-13	Vehicle shall be capable of driving 5 meters while staying within a straight double-way track.	Prio 1
Elad-14	Vehicle shall be capable of driving 5 meters while staying within a S-shaped double-way track.	Prio 2
Elad-15	Vehicle shall be able to detect a 20cm x 20cm x 20cm white stationary box at a distance of minimum 10cm and maximum 5m and at a maximum speed of 10km/h.	Prio 2
Elad-16	Vehicle shall be able to stop in front of a 20cm x 20cm x 20cm stationary box when driving on a straight one-way track. After the vehicle has stopped, the distance between vehicle and box shall be no farther than 20cm and no less than 10cm.	Prio 2
Elad-17	The vehicle deceleration shall be controlled by parameter	Prio 2
Elad-18	In case the vehicle has to stop, it shall start to brake at the required distance it would take to come to a complete stop with the current speed and with a parameterized deceleration.	Prio 2
Elad-19	Vehicle shall be able to detect a stop line present on the right side of a straight double-way track.	Prio 3

Elad-20	Vehicle shall be able to stop in front of a stop line when driving on a straight double-way track. After vehicle has stopped, the distance between vehicle and stop line shall be no farther than 10cm and no less than 1cm. See figure 4 track number 7.	Prio 3
Elad-21	On a priority straight double-way track, the vehicle shall be capable of driving through an intersection without stopping (see figure 4 track number 4).	Prio 3
Elad-22	After having immobilized the vehicle at an intersection on a non-priority straight double-way track, the vehicle shall be capable of crossing the intersection (see figure 4 track number 5).	Prio 3
Elad-23	Vehicle shall be able to drive around a 8-shaped double-way track.	Prio 3
Elad-24	The vehicle shall be capable of determining if it has lost track of the lane it was previously following. In this case, the vehicle shall stop immediately and report its state to the interface.	Prio 1
Elad-25	Vehicle shall be able to detect a 20cm x 20cm x 20cm white stationary box located on the other side of a straight double-way track and drive beside it without stopping	Prio 3
Elad-26	When an stationary obstacle blocks the current driving lane, the vehicle shall slow down to 3km/h and proceed to drive by obstacle without crossing the left white solid line. After passing the stationary obstacle, the vehicle shall come back to the initial driving lane without crossing the right white solid line. During the overtaking, the vehicle shall not drive over the safety area of the stationary obstacle which is defined figure 3 with, in this case, DFront = DBack = 15cm and DLeft = DRight = 5 cm. *In this case, it is assumed that no vehicle is coming on the opposite lane.*	Prio 3
Elad-27	When an stationary obstacle blocks the current driving lane, the vehicle shall slow down to 3km/h and proceed to drive by obstacle only if the opposite lane is clear otherwise the vehicle shall come to a complete stop in front of obstacle.	Prio 3
Elad-28	When two or more vehicles are waiting at a intersection, the vehicle shall be capable of deciding of which vehicle goes first and wait for its turn in order to follow traffic rules. (a) As soon as the vehicle has detected an intersection which requires direction information, it shall request for direction (right, left, go forward) to the interface. (b) The vehicle shall wait at the intersection until it gets direction order from the interface. (c) The vehicle shall be able to detect vehicles waiting at the intersection which has the same size as the vehicle in context. It shall know how many and where they are located. (c) The vehicle shall be able to determine while waiting in the intersection other vehicles states, such as arriving, waiting, driving left, driving right, driving straight, their speed and their distance to the vehicle (d) The vehicle shall continuously calculate its safety area depending on the environment and the direction order. (e) When it has been considered “safe”, the vehicle shall take off in	Prio 3



	the direction ordered by the interface.	
Elad-29	Vehicle shall use emergency braking if normal braking will not be enough to stop at a minimum distance of 10cm in front of a 20cm x 20cm x 20cm stationary box when driving on a straight double-way track. (a) Emergency braking shall use the maximum braking acceleration that the car is capable of. (b) Emergency braking state shall be reported to the interface. (c) When the distance to the object is enough for the normal braking to come to a complete stop within a minimum distance to obstacle of 10cm then switch back to normal braking mode.	Prio 2

Interface

Elad-30	An interface shall be implemented in the host computer.	Prio 1
Elad-31	The user shall be able to take over the control of the vehicle via the interface.	Prio 1
Elad-32	Vehicle information such as speed, acceleration, deceleration shall be displayed in the interface.	Prio 2
Elad-33	An image of what the vehicle sees and detects shall be displayed in the interface. Detected objects shall clearly be identified.	Prio 2
Elad-34	Vehicle decisions shall be displayed in the interface.	Prio 1
Elad-35	When requested, the user shall be able to give order to the vehicle.	Prio 3
Elad-36	If one of the Lidar, camera and WIFI components no longer can communicate with host computer, vehicle shall slow down and stop in 1 second and give a warning.	Prio 2
Elad-37	It shall be easy to install and use the interface program to any host computer and connect to the vehicle. No license shall be required.	Prio 3

Data logging

Elad-38	Lidar data shall be saved continuously on the host computer. A SLAM algorithm shall then be used in order to play back what the robot saw.	Prio 2
Elad-39	The camera data shall be recorded and saved on the host computer in a automatic generated time stamped file and folder structure.	Prio 2
Elad-40	All other sensor input shall be recorded and saved on the host computer in a automatic generated time stamped file and folder structure.	Prio 2
Elad-41	Logged data from all components shall be synchronized.	Prio 3
Elad-42	Logging data shall be documented, structured and archived	Prio 2

Verifying

Elad-43	A model of the car in a virtual world shall give users the capability of trying out new algorithms without having to test on the actual vehicle directly. (a) The virtual world shall include tracks suitable for the performance requirements.	Prio 2

	(b) The dynamics of the vehicle should behave similar to how they are in real life.	
Elad-44	All requirements from vehicle functionality paragraph shall be covered by a test case.	Prio 1
Elad-45	Execution of test cases shall be automatized as much as possible.	Prio 3
Elad-46	A test result report shall be provided.	Prio 1

Documentation

Elad-47	A document with building instruction shall be provided. It shall be illustrated with pictures.	Prio 1
Elad-48	The code shall be commented and structured	Prio 1
Elad-49	The platform should be documented so that a new user could pick up and continue development.	Prio 1
Elad-50	SW version shall be archived and documented.	Prio 1
Elad-51	A test case specification shall be provided. A new version shall be created for each new software and describes correlation between test cases and software version.	Prio 3

Process and tools

Elad-52	The project shall be developed according to Agile process.	Prio 1
Elad-53	A tool such as Jira or similar shall be used. (a) All tasks shall be reported in the tool (b) Duration shall be estimated (c) Spent time for each task shall be logged (d) Report and retrospective shall be done at the end of each sprint (e) a 2 weeks sprint shall be used	Prio 1
Elad-54	The development shall be done with Robot Operating System (ROS) framework.	Prio 1

Appendix C

Priority 1 Tests

ID	13-01
Title	Stay within lane for 5m.
Pre-Conditions	Battery of vehicle should be fully charged
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Run vehicle 3. Monitor the vehicle to see whether it stays within the lanes
Expected results	The vehicle drives 5m without human interference and none of its wheels goes outside the lane markings.

ID	24-01
Title	Vehicle stops when loosing track
Pre-Conditions	Battery of vehicle should be fully charged. Cover 1 m of the lanes of the straight test track with dark paper. The coverage should start 1 m into the track. Place vehicle in front of the covered part of the track. Open GUI and make sure that no lanes are found. Move car backwards and forwards until the exact position where the lanes are no longer seen is found. Mark where the front of the car is located at this spot
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Run vehicle 3. Monitor vehicle to see if it stops 4. Look for state report on the GUI 5. Measure distance to the marked spot
Expected results	The vehicle will go forward until it reaches the covered part of the track. The front of the car will be within 10 cm of the marked spot where the lanes are no longer visible. The vehicle has reported that it has stopped to the GUI.

ID	31-01
Title	Stopping the vehicle via the interface
Pre-Conditions	Battery of vehicle should be fully charged. GUI should be open
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Run the vehicle for 1 m without human interference 3. Stop the vehicle using the GUI
Expected results	Vehicle stops within 1 second after the stop button is pressed on

	the interface.

ID	31-02
Title	Turn the vehicle using the GUI
Pre-Conditions	Battery of vehicle should be fully charged. GUI should be open.
Test steps	1. Place vehicle in front of the stop line on the 8-curved test track 2. Run the vehicle one lap around the track using the controls on the GUI
Expected results	The vehicle can be driven one lap around the track using the GUI without crossing over the lanes.

Priority 2 Tests

ID	12-01
Title	Accelerate to max allowed speed when no obstacles
Pre-Conditions	Place the vehicle in the right lane of the straight track. Make sure that there are no obstacles in the near proximity of the course. Set the lowest speed as maximum speed.
Test steps	1. Run the vehicle 2. Look at the motor speed 3. Stop the vehicle manually when it has driven through the straight track 4. Raise the speed one step and do the tests again
Expected results	The vehicle accelerates up to the set max speed before it is stopped.

ID	14-01
Title	Drive 5 m in S curved lane
Pre-Conditions	Battery should be fully charged.
Test steps	4. Place vehicle at the beginning of the straight test track in the middle of the right lane 5. Run vehicle 6. Monitor the vehicle to see whether it stays within the lanes
Expected results	The vehicle drives 5m without human interference and none of its wheels goes outside the lane markings.

ID	15-01
Title	Detect obstacle at maximum distance while stationary
Pre-Conditions	Place a white box of the size 20cm x 20cm x 20cm in the middle

	of the right lane on the straight test track and place the car in the same lane with it's front 1.5 m from the box.
Test steps	1. Start object detection node 2. Look for results
Expected results	The object should be detected within 2 seconds of start up

ID	15-02
Title	Detect obstacle at minimum distance while stationary
Pre-Conditions	Place a white box of the size 20cm x 20cm x 20cm in the middle of the right lane on the straight test track and place the car in the same lane with it's front 0.3 m from the box.
Test steps	1. Start object detection node 2. Look for results
Expected results	The object should be detected within 2 seconds of start up

ID	16-01
Title	Stop in front of object
Pre-Conditions	Place a white box of the size 20cm x 20cm x 20cm in the middle of the right lane 3 m into the straight test track. The speed should be set to the lowest level.
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Start the vehicle 3. See whether it stops in front of the object 4. If it stops, measure the distance between the front of the car and the object 5. Raise the speed by one step and redo the steps
Expected results	The vehicle stops in front of the object with a distance of 10-20 cm between the front of the vehicle and the object.

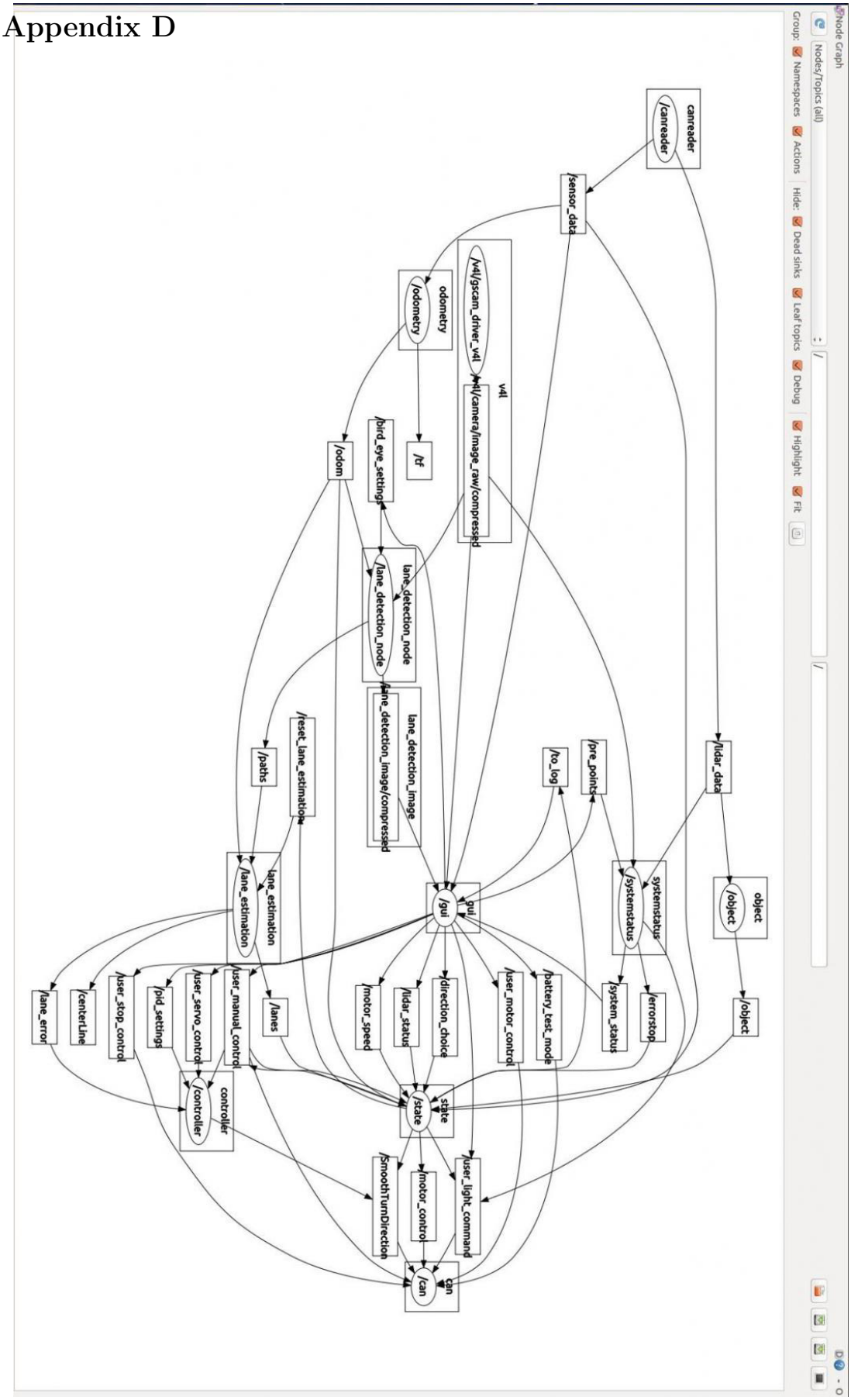
ID	18-01
Title	Controlled deceleration for object
Pre-Conditions	Place a white box of the size 20cm x 20cm x 20cm in the middle of the right lane 3 m into the straight test track.
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Start the vehicle 3. Look at the set motor speed and see how it changes.
Expected results	The motor speed lowers from the running speed to zero by (vehicle speed)=(distance left to object/desired breaking distance) * (vehicle speed)

ID	18-02
Title	Controlled deceleration for stop line
Pre-Conditions	Place a white stop line on the right lane 3 m into the straight test track.
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Start the vehicle 3. Look at the set motor speed and see how it changes.
Expected results	The motor speed lowers from the running speed to zero by (vehicle speed)=(distance left to object/desired breaking distance) * (vehicle speed)

ID	29-01
Title	Emergency braking
Pre-Conditions	GUI should be open
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Run the vehicle. When the vehicle has run 1 m place an object x m away from the vehicle (x=distance where normal braking will not work
Expected results	Vehicle stops at a minimum distance of 10 cm to the object. The GUI shows that emergency braking has been activated

ID	36-01
Title	Stop if sensor communication lost
Pre-Conditions	Have GUI connected
Test steps	1. Place vehicle at the beginning of the straight test track in the middle of the right lane 2. Run the vehicle 3. When the vehicle has travelled one meter close down the GUI
Expected results	The vehicle should now have lost contact with the GUI and will stop.

Appendix D


Node Graph
Nodes/Topics (all)
Group: Namespaces Actions Hide: Dead sinks Leaf topics Debug Highlight Fit
canreader
/canreader
/sensor_data
/v4l/gscam_driver_v4l
v4l
/v4l/camera/image_raw/compressed
odometry
/odometry
/tf
/odom
/bird_eye_setting
/lane_detection_node
lane_detection_node
lane_detection_image
/lane_detection_image/compressed
/reset_lane_estimation
/paths
/lane_estimation
lane_estimation
/lanes
/lane_error
/centerLine
/user_stop_control
/pid_settings
/user_servo_control
/user_manual_control
/controller
controller
/gui
gui
/to_log
/pre_points
/lidar_data
/object
object
/systemstatus
systemstatus
/system_status
/errorstop
/battery_test_mode
/user_motor_control
/direction_choice
/lidar_status
/motor_speed
/state
state
/user_light_command
/motor_control
/SmoothTurnDirection
/can
can