CHALMERS

UNIVERSITY OF TECHNOLOGY

Self-driven Miniature Vehicle for Carolo Cup 2014

Technical Report

Viktor Botev
botev@student.chalmers.se

May 22, 2014

Department of Computer Science and Engineering
Göteborg, Sweden

Abstract

Carolo Cup is an international student competition for self-driven miniature vehicles organized annually in Germany. The competition is inspired by the DARPA Grand Challenge. The vehicles should fulfill requirements such as: time-constraints, performance, and reliability. The basic functionalities required for the Carolo Cup competition are lane following, overtaking, intersection handling and parking. This report summarizes the work of team Legendary on those scenarios, but mainly focuses on one of the functionalities - lane following. The report covers the theoretical research and the practical implementation.

Keywords: carolocup, lane following, image processing, autonomous parking, autonomous vehicle

Contents

1 Introduction

1.1 Carolo Cup Competition

The competition Carolo Cup provides student teams from universities the opportunity to develop and implement autonomous model vehicles. The challenge is to realize the best possible vehicle control in different scenarios inspired by a realistic environment. The annual competition allows presenting the studentś skills to a jury of experts from industry and academia, and to compete with other student teams.

The student team is hired by a fictitious car manufacturer to develop, produce, and demonstrate a possible cost-effective and energy-efficient concept of an autonomous vehicle by using a model vehicle on a scale of 1:10. In the competition, certain tasks must be performed as quickly and reliable as possible; furthermore, the concept must be explained in a presentation.

Each concept is compared and evaluated with the concepts from other participating teams. Therefore, different static and dynamic disciplines are used to compete for a total of 1,000 points. [7]

1.2 Hardware

According to the rules the hardware must be a vehicle in 1:10 scale with electric drive and certain dimensions for the chassis. But the important part is that there are no restrictions for sensors, so the team could decide what type of sensors and which particular type suits them best. Of course all decision have to be backed up with data and defended in the static competition. The chassis requirements are mainly to give the same chances for the teams to perform on the competition. There is one important point about the chassis concept that for the parking the slots are predefined, so the smaller the length of the vehicle the better, because it allows more freedom there.

Considering the embedded systems there no limitation also, but important factors are weight, power consumption, and temperature. Obviously putting a normal desktop computer on top of the 1:10 scale car would make it very difficult to maneuver and maintain proper speed. But considering the complexity of the tasks important factors are processing power and development complexity.

1.3 Lane following

The lane following is the first dynamic discipline. It test if the car is trained well enough to follow a route lane. Challenges are lane marking recognition, dealing with limited camera view, missing markings and intersection. The road has predefined properties - width, color. The lane marking are white and the route color is black. Although that makes the task easier the lighting conditions could be a very big challenge as well. The car should travel as many meters as possible for 3 minutes. Some penalties are applied for going of the lane or for human interaction.

1.4 Parking

The second dynamic discipline is parking. It checks the ability of the car to park autonomously. The car should be able follow a lane (or at least go on a straight line), detect a proper spot for parking, and make the necessary maneuvers to go into it. The challenges are detecting the exact space in the parking slot, detect if the parking slot is already taken, and during the maneuvers to cope with errors that might occur during execution. The sensors should be able to detect the obstacles very close to the car as well as those far away from it. The sampling rates might be a challenge too, because when the car is too close to the obstacles it might hit them before they detect that the distance is too small.

1.5 Overtaking

The third and last dynamic discipline is overtaking. It is the most challenging discipline, because it combines to some extend the challenges from the previous two and adds also some more complexities. The car is suppose to run on the track and do the lane following as described above, but this time some obstacles are placed on its way (just like parked or moving cars). The car must overtake the obstacles without hitting them and return back to its lane. There is one moving obstacle that should be overtaken while it is running. And finally there is an intersection, where the car should stop and give right to a vehicle on the right, if there is one. The challenges here are to calculate the right speed for the sensor and processing power, so that the car is not reacting too late to the sensor data and running into the objects.

1.6 Static competition

Besides the dynamic competition there is also a static competition. It is basically a presentation of the concepts implemented in the car. The static competition is worth 35% of all points, so it should be taken seriously. The presentation is timed for 20 minutes and it should cover all the aspects of the car - hardware concept, cost, power consumption, software architecture, lane following concept and control, parking and control, and overtaking strategy and control. Everything should be present clearly and with as much details as possible, but the time constraints must be followed.

2 Technical Overview

2.1 Hardware Concept

Hardware concept is really important, cause it puts the major limitation and gives the major possibility of the solution to the problems. In other words it has great impact. Part of the hardware concept are goals and requirements, sensor layout and manufacturing technologies. In order to solve the problems and challenges given by the competition the Legendary chose to use - four wheel drive flyfish car for chassis, camera with wide angle, LIDAR sensor, four infrared sensors and two ultrasound senors for sensor layout, three control boards - Odroid X2 and two Arduino Mega, for actuators - servo motor and Cheetah super fast motor for the engine. In order for the car to able to compete in all of the disciplines a mechanism to switch programs was needed. It was implemented in the hardware by adding a touch display on the back of the car.

The goals for the hardware are:

- To supply power and steering actuators in an easy to control manner
- To supply easy to use low level components such as actuators and sensor interface. That is the combination of Arduino Mega and chosen components are used.
- To supply redundant sensor layout for covering the necessary angles of view
- To supply enough processing power for the operation needed
- To supply separation of concerns for fault tolerant reasons to ensure reliability on the car

The manufacturing process were as followed - first everything was designed with a software tool CATHIA 5 and the CAD files are used to manufacture the parts. Two machines were used Laser cutter and CNC machine basically depending on the material.

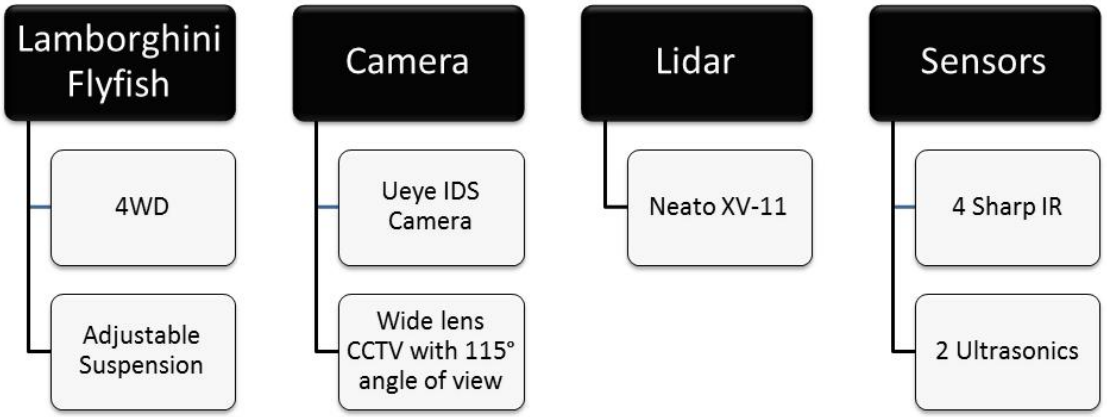


Figure 1: Car Layout

2.2 Electronics Concept

From electronics point of view the main goal is to allow the hardware and software to work together. It contains mainly wiring and some consideration for the power supply. The system was build to have fuses for the main components - motor, Odroid, servo, rest of the system. A main switch is considered to disconnect the power from the battery from entering the system. That protects from accidental click on a button and from discharging the battery unnecessarily if something is left running. There are two circuits in the system one 5V and 6V circuit. The 6V is providing power to the servos, because at this voltage they give maximum torque and the steering will be faster, and 5V for the sensor and rest of the system. To implement that two voltage regulators were needed which translate the 7.4V from the battery to the respective voltage. The battery was chosen to allow a long time of testing without recharging - up to 1 hour and a half. In addition to the battery a buzzer is added that starts beeping when it is time for the battery to be charged, that prevents over discharge of the battery, which could extend its life time. Two Arduinos were chosen one for gathering sensor data and one for executing the low level commands on the actuators mainly for separating the critical functionality for operating the vehicle from the sensor gathering and allow more bandwidth in the communication between the devices.

The design of the Electrical system was made by a tool called TinyCAD. All the wires and connections between the components were specified there and then it was used to build and verify the system later.

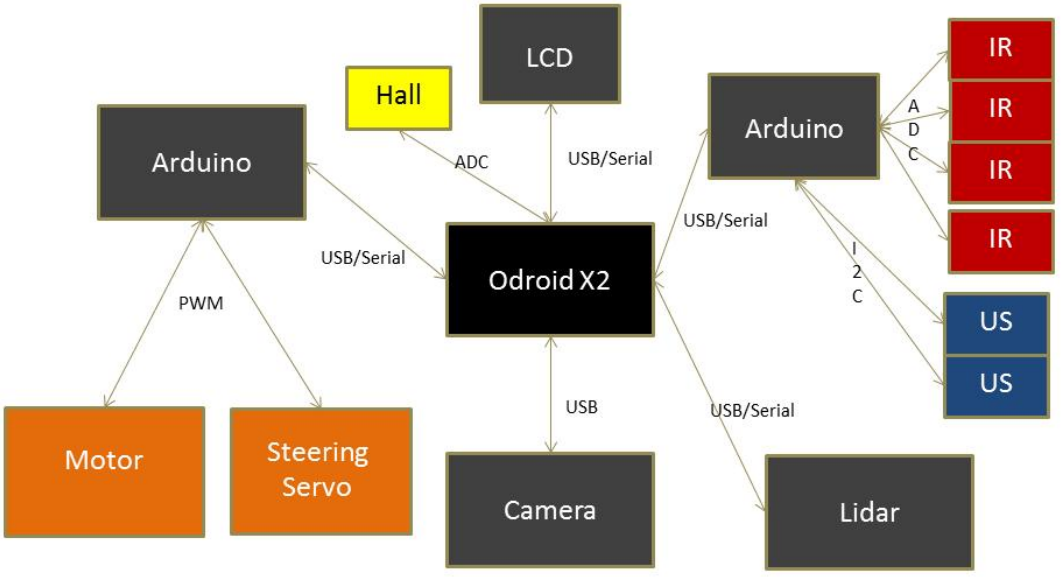


Figure 2: Electronics Layout

2.3 Software Concept

The Software Concept contains mainly the architecture and organization of the software components. The software needs to be organized in a way that several tasks are executing at the same time and that the information processed by the control modules is as fresh as possible. For those reason the processing power of the four core Odroid X2 has to be utilize, also the communication between the boards needs to be taken care and the least but not last the low level software on the Arduinos need to be developed.

The operating system used was Debian Wheezy. The reason was mainly because it is the most stable distribution for the Odroid and it provides compatibility with the camera driver. For the parallelism and processes management the Legendary team uses Open DaVinci platform. It provides process scheduling, parallelism abstraction and possibility to build plug-able components. It also has a distributed nature and the components could be run on different machines and still communicate with each other with UDP messages. The components are written in C++ language. A big help to the implementation gives the OpenCV library for image processing.

For the low level control the Arduino language was used, because it has already build-in several components for operation with hardware. The communication between Arduino and Odroid was done via Serial communication and a simple custom made protocol.

Finally for simulations of algorithms the MATLAB software was used. Main functions needed were curve fittings and neural networks model. The results from those simulations are then translated to C++ and put in the car within an Open DaVinci component.

To allow switching between different programs commands are received from a touch display and a bash script is processing those commands by powering up the necessary components for the desired program. A stop command is also available for shutting down all running components.

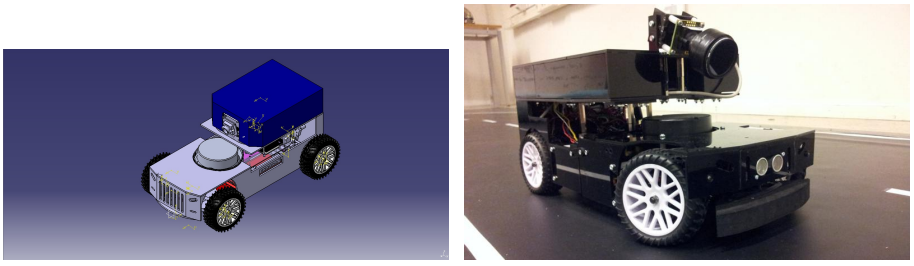
3 Implementation

In this section the Legendary car solution will be described in details. The structure will follow the competition structure. The implementation and the competition results will be discussed also.

3.1 Hardware

The hardware solution was four wheel drive Lamborghini Flyfish with Cheetah 60A motor with Sensored ESC. The motor allowed the car to be very heavy (less than 6kg) and still maintaining speeds of upto 4m/s, which is more than the maximum speed ever achieved at the competition. The actual car weight at the end was 3,5kg and the speed we managed to run - 1,2 m/s. The steering servo chosen provides torque of 6.5 kg*cm and speed of 40 ms/10°. The usual compensation angle within the track was 5 degrees, which allow the car to compensate every 20ms. The limitation actually was not the servo since the software compensates every 40ms.

The chassis was modified to carry the components. It is basically three floor. The first floor is made mainly using the original floor with small modification for adding fusses and voltage converters over the battery. That floor also carries the servo, the battery and the motor. The second floor is carrying the LIDAR sensor, the motor controller board, power interface for all the components, the transmitter plus the two ultra sound sensors and all four infrared sensors. The last floor is for placing the camera, the sensor gathering board plus the Odroid. The floors are aluminum plates made by a CNC machine. Besides the floors another part of the chassis is the body. All body parts are made by Laser cutter. The material for them is plastic. The challenge with the body shell was to cover and protect all the components according to the required IP11 standard and at the same time to let the LIDAR operate in as much degrees as possible. The current solution made the car much more heavy, but was very easy to manufacture and allowed flexibility during development.



(a) Design in CATHIA (b) Final car solution

Figure 3: Design and final solution for Lamborghini Legendary

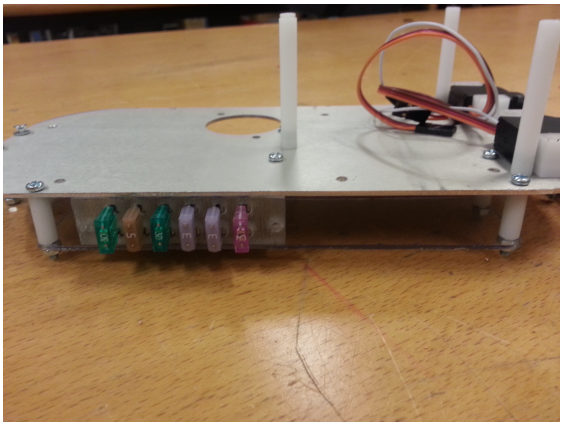


Figure 4: Second floor plate

The sensors we are chosen in a priority manner. First the camera is really important and considered as with no alternative since the lane markings are hard to detect in another way. It also gives a lot of possibilities for detecting obstacles and other properties. A problem was identified from previous year that the camera cannot see the road when turning that is why a wide angle lens was added to provide 115 degrees of view, which allowed a presence of two lanes even on the sharpest curve. In order to solve the problem for reflections and light conditions an adjustable polarized filter for professional cameras was mounted in front of the camera. For the parking infrared sensors are chosen to detect the gap. They were mounted on the side right next to the tires to allow the

blind zone of 3cm to be insignificant for the measurements. Two more infrared were mounted on the back to ensure parallel parking. For back-up two ultra sounds sensors were mounted - one on the front and one on the back. They ensure proper maneuvers inside the parking slot. Finally for the overtaking a big angle of view is needed and processing of data as far from the objects as possible is important. That is why a LIDAR sensor is chosen to detect the obstacles from a distance and allow reasonable speeds. Besides the sensors responsible for the environment - two hall effect sensors were used to detect the wheel speeds of the car. One is used for feedback speed control for the car - to ensure steady speed. The second one is used for feedback for the parking algorithm.

Finally the brain of the car was decided to be located within an Odroid X2 board. The reasons for that is the good processing power and memory for a low cost. In the competition car cost and energy efficiency is also an important component. For those two the Odroid is much better choice than an Intel processor. Besides though the Odroid created a lot of problems because of its incompatibility with the camera driver and because of not enough processing power for running both the LIDAR and the camera at the same time. The low level control boards are chosen to be Arduinos basically because of their simplicity and robustness. The communication is decided to be serial.

3.2 Lane following

Lane following is the most challenging discipline and it is also a prerequisite to the other two. Although for the parking is needed only following a straight line, which most teams just hard code. There were two algorithms alternatives considered for the lane following implementation. The first one investigated was to built an algorithm around the Bird-Eye transformation [3] or also called Top-view transformation of the images in order to get parallel lines for the road markings and have a good relative distances between them. The problem with that method is that the transformation took too much processing power - each image was processed for around 200ms. The algorithm included thresholding, Canny contour finding, bird eye transformation, Hough transformation to find lines, and DBSCAN classification for lines. Those techniques will be analyzed in details later. The problem was that the bird eye transformation took 70% of the processing time and the algorithm performed not very well on curves. The 200ms will limit the car to speeds to 0.5 m/s and that only in case that every image is perfect, which was not acceptable. For that reason another algorithm was investigated based on finding the vanishing point [4] of the road lines. The original algorithm used in the resource [4] was developed for calculating a vanishing point for a big number of lines, this also make the calculations slow. An image was processed on the Odroid for around 120ms, to speed up the processing the Hough transformation was replaced by in-house built algorithm for finding the lines and only on those lines a simple geometry equation was used to find the vanishing point [1]. After that the algorithm took only around 20ms of execution time.

The Legendary team addressed the problem by splitting it into smaller problems in the following sequence:

1. Detect lines on the road
2. Find road vanishing point

3. Find goal position and orientation
4. Find current position and orientation
5. Calculate compensation for the error in the position
6. Send decision to the low level control board
7. Execute the decision

3.2.1 Detect lines on the road

The image received from the camera is in gray scale. In order to avoid noise from light conditions we use thresholding [2], which means that above some gray level the color is considered white and below it it is considered white. The thresholding level has to be calibrated before letting the car go. It is also related to the polarized filter, so they both needs to be configured before running. After the thresholding a Canny algorithm [8] is used to find the contours of the objects on the image. When contours are available all of them are fitted into the smallest rectangular possible [9]. This transfers the image into a set of rectangles. Those three are pretty straight forward, because all those methods are widely discussed in the literature and they are part of several libraries. In the current implementation the OpenCV is used as a source for those functions. The next step was

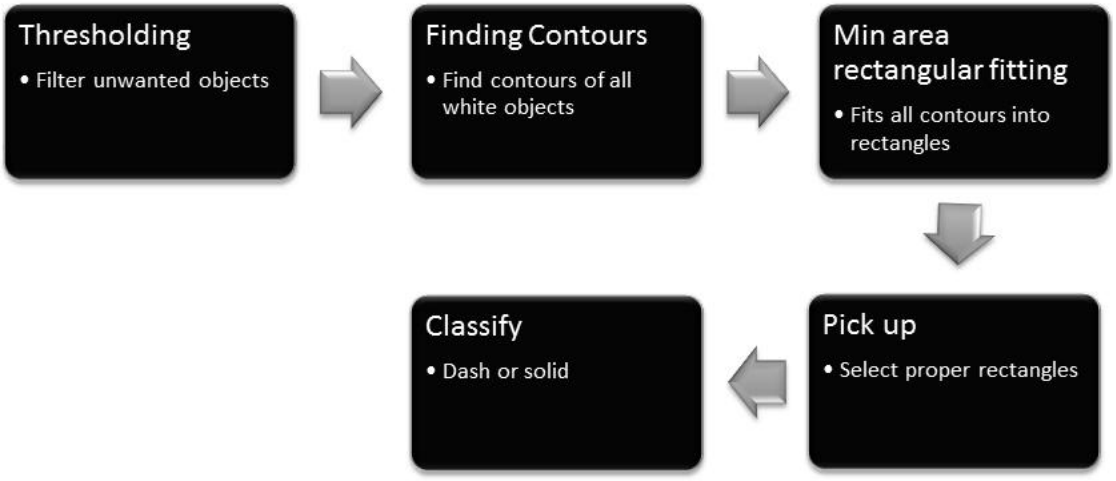


Figure 5: Detect lines algorithm overview

to select the rectangles that are containing lane markings - both dash and solid lines markings. For this purpose the Legendary team used a heuristic function based on the ratio between short and long side of the rectangle, the rectangle area and minimum angle of the rectangle orientation to the x axis. The reason for the ratio is that the lane markings have a very high ratio between long and short side - the proper parameter for selecting a rectangle has to be calibrated before the car starts running. The are of the rectangle is important, because too big rectangles are classified either as obstacles or as intersection - this heuristic was empirically found out during testing. The value for the are is predefined so that the sharpest curve is still classified as lane marking but the

intersection is not, then a safety factor of 1.2 is used to define the value. This parameter does not need calibration. The final step is to distinguish between all the rectangles in the set of lane markings which ones are dash lines and which solid lines. This is done by another heuristic function based again on ratio between rectangle sides but this time also taking into consideration the maximum length of a dash line possible. Both parameters needs to be calibrated during before running the car.

(a) Pick up lane markings (b) Classify lane markings

Figure 6: Results from pick up and classify heuristic functions

On Figure 6 you can see the results from the pick up and classification procedures. For simplicity later on in the implementation the rectangles are replaced by a line passing through the middle points of the shortest sides of the rectangle. On the Figure 6a the found lane markings are marked with black lines on top of the raw gray image, and on the Figure 6b the found dash lines are in purple and the found solid lines in green.

Challenge in this algorithm was to handle intersections and start/stop lines. The reason for that is that the contours formed and the rectangles fitted on them could not be approximated as lines in the same way as normal lane markings(line through the middle of the short sides of a rectangle). To deal with this situations the program simply ignores rectangles with very big area.

3.2.2 Find goal and current car orientation

In this section will describe how from the picked up lines the orientation of the current orientation of the car is described in the environment and how a goal position and orientation could be selected. We are combining three steps of the algorithm in one mainly because they are simple and are very related to each other. The current position of the car is defined as the lower mid point of the image and current car orientation is pointing straight 90 degrees up from that point. How is this position related to the environment? In order to understand that we need to find an appropriate road model that should approximate the lane markings we got into a road. For that purpose we first find the vanishing point of the lane markings. Since on the image we have less then 10 lines there is no need for statistical calculations to get the vanishing point, instead we approximate first the dash lines into one line and then cross section it with the solid line(s). If there are two solid lines we pick up the average point from the two points. After the vanishing point has been found the orientation of the car to follow the profile of the road will be define by the line between the current position and the vanishing point, that will be the goal orientation. The goal position will lie on this line as well, but we have to take into consideration current speed to be able to identify a point. So the

algorithm checks the current speed and based on that decides which point to pick up as a goal point.

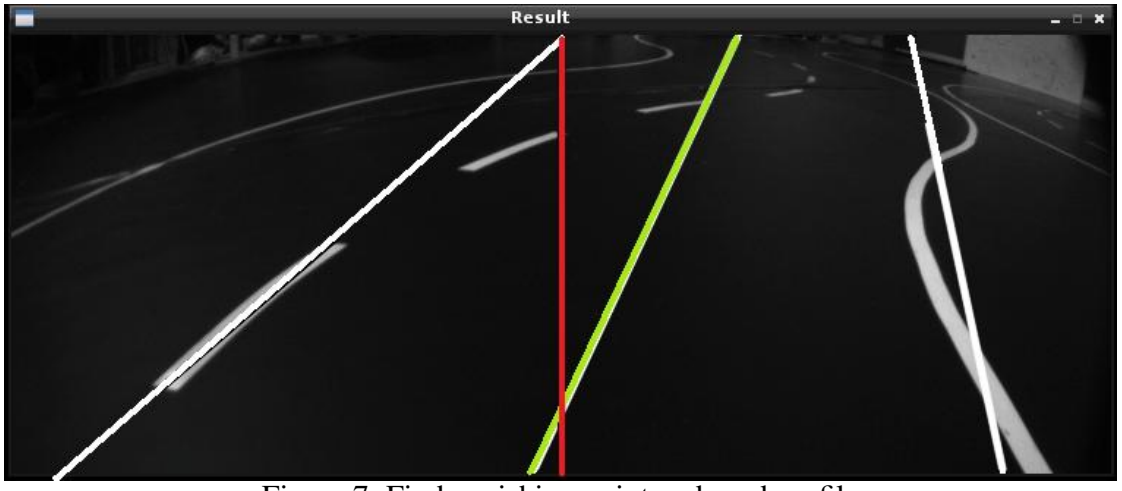


Figure 7: Find vanishing point and road profile

On Figure 7 the road profile is identified by the white lines, the vanishing point is up out of the picture. The red line is showing current car orientation, the green one on the other hand the goal car orientation. At this time the car is driving with 0.5 m/s, so the point that will be selected will be lying on the green line with y coordinate equal to 3/4th of the height of the image. The speed rule is simple if car is standing still the goal point is current position. If the car is driving 2m/s the goal is at the top of the image on the green line. All speeds between those are considered proportional and the corresponding y coordinate is used.

A big challenge for this algorithm is to calculate the vanishing point when only one lane marking is recognized. In order to deal with the that it tries to estimate the position of the other markings by predicting the road size on the bottom of the image and the road angle at the vanishing point. The prediction is done by fitting empirical data to a function in MATLAB and using that function during the running. The success rate of the function is around 70%. Another option that the Legendary team tried is to use neural networks to predict those parameters but due to the limited time this idea was discarded.

3.2.3 Calculate compensation for the error in the position

After the current and goal position have been defined we need to find out what steering angle and what speed needs to be send to the low level car control in order to reach the goal. The algorithm for determining the necessary steering angle is a PID control. The parameters for it are tuned with an algorithm called twiddle [5]. The integral part incorporates the speed. The speed is predefined before running and its static while driving on curves to assure steady state cornering. But on straights the car increases the speed and decreases again at the end of the straight. To decide whether the car is on straight or not, it is checking the compensation decided from the PID, if the compensation is less than certain small angle for a couple of milliseconds it assumes that this is a

straight line. When the compensation become bigger than that predefined small angle it returns back to its default speed.

3.2.4 Send decision to the low level control board

After found the desired steering and speed have to be send to the low level board. For that a simple in-house built protocol is used. The communication is serial and the protocol is text. Each command has a prefix letter after that a value and a terminating character to ensure completeness of the message. Example commands are:

- s10/ - steering turn 10 degrees right (s - steering, 10 - value, / - terminating character)
- ff5/ - frequency forward 0.5 m/s
- fr5/ - frequency backward 0.5 m/s
- ir/ - turn on indicators right
- il/ - turn on indicators left
- ia/ - turn on indicators all
- is/ - turn off indicators all
- b+/ - activate brake
- b-/ - release brake

3.2.5 Execute the decision

After send to the low level control board the commands are translated into appropriate PWM signals to control the servos for motor and steering. Also some digital signals for LED indicators are used. The main code in the Arduino board though is for decoding the incoming command and making sure that the message is valid.

3.2.6 Calibration procedure

For proper operation of the algorithm a proper calibration of the important parameters is needed. The parameters to be configured are the thresholding parameter, the ratio for the rectangles between short and long side to chose if this is dash or solid line - ratio and max length, the parameters for estimating road paramteres - dash angle at straight line, road size at straight line, road angle at straight line. To configure the first parameter important the car has to do at least one lap with the current settings and the settings are dependent on the polarized filter as well. So what we do is we run one lap and monitor online and do adjustments, if for one lap we do need to do adjustments the thresholding parameter is set. For the ratio and max length we do the same. For the last three the car is put on a straight line on the goal trajectory and the parameters are recorded.

3.2.7 Code organization

The code is organized into two high level Open DaVinci components - Lane Detector and Driver - plus one low level component executing on the Arduino board. The lane following starting script is powering up the Open DaVinci with its supercomponent running at 20Hz frequency, the lanedetector at 40Hz and the driver at 30Hz. The Arduino is listening for commands every 10ms to ensure proper driving of the car.

3.3 Parking

3.3.1 Concept

The parking discipline should be incorporated together with the lane following. The car should follow the right lane closer to the solid right marking and monitor for parking slots. Unfortunately due to not enough time in team Legendary we decided to make the parking without the lane following. The car just go straight. The problem here is that even when giving a fully straight command the car is not always going completely straight and there is no driver to compensate for that. This might result in not detecting a spot or wrongly choosing a taken spot. But that risk was taken because a solution that works sometimes is better than no solution. Besides that the parking algorithm was designed as a state machine.

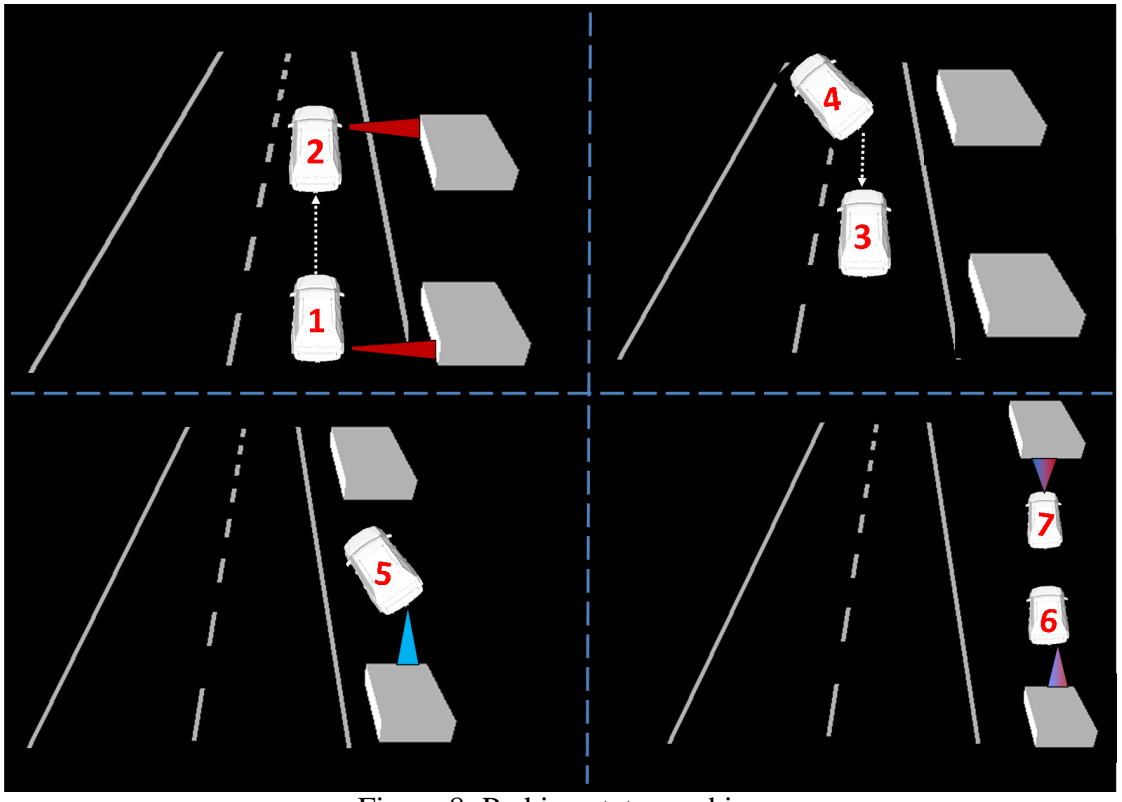


Figure 8: Parking state machine

1. Searching state - the car is going straight searching for a spot to park. The implementation has the possibility to be configured for a concrete spot size or for

taking the first in which the car could fit in.

2. In a possible spot - the car has detected that a spot is there and is now waiting for the obstacle marking the end of the spot. The detection is done by the infrared sensors on the side. When both sensors are not detecting anything then on the side of the car the place is free. If both are detecting something then the place is considered obstacle. If the front is not detecting and the rear is detecting we are entering a spot.

3. Spot detected - the car has detected end of a spot (Front infrared is detecting obstacle and rear is not). Then the car checks if the spot is desired for parking, if it is then the car turns a right indicator to show that the parking is started and the state machine goes to the next state, otherwise it returns to the "Searching state".

4. Drive left forward - the car turns its wheels maximum to the left and goes until it reaches the dash marking (In this solution it uses hard coded values from the wheel encoders, but in a real solution it should use the camera to detect when it is close to the middle of the road).

5. Drive straight backwards - after that the car goes straight backwards until the front wheels are both out of the road (Same the wheel encoders are used, but instead camera could be used or other sensors).

6. Drive left backward - the car turns its wheels maximum to the left and drives backward until the sensors on the back detect that the obstacle is too close, or until the two infrared sensors on the back detect the same value. In the first case compensation forward is needed, on the other hand in the second case we just finished parking.

7. Drive right forward - the car turns the wheels maximum to the right and goes forward until the front ultrasound sensor detects that the car is too close to an obstacle or the two back infrared sensors detect that the parking is finished. In the first case we go to the previous state.

8. Parking finished - the indicators are lighten all to indicate the finish of parking.

The speed for the parking was set to 0.5 m/s because the sensors update rate did not allow faster speeds especially in states 6 and 7, possibility for faster speed was available for the searching state, but for simplicity the speed was set static.

3.3.2 Code organization

The code is organized into two high level Open DaVinci components - Driver and Sensors - plus one low level component executing on the Arduino board. The driver was run at 40Hz and the Sensors at 50Hz. The Arduino is listening for commands every 10ms to ensure proper driving of the car. The same commands as those described for lane following are used.

3.4 Overtaking

Overtaking discipline is the most challenging dynamic discipline on the competition. The lane following should be working good, an algorithm for overtaking static and dynamic obstacles is needed and an intersection handling as well. Due to limited resources that the Legendary team has the program for this discipline was not finished, so the car was not ready and did not participate in this competition. Although the idea behind the algorithm is as follows.

3.4.1 Avoid obstacles

The overtaking maneuver is thought to be a state machine as shown on Figure 9.

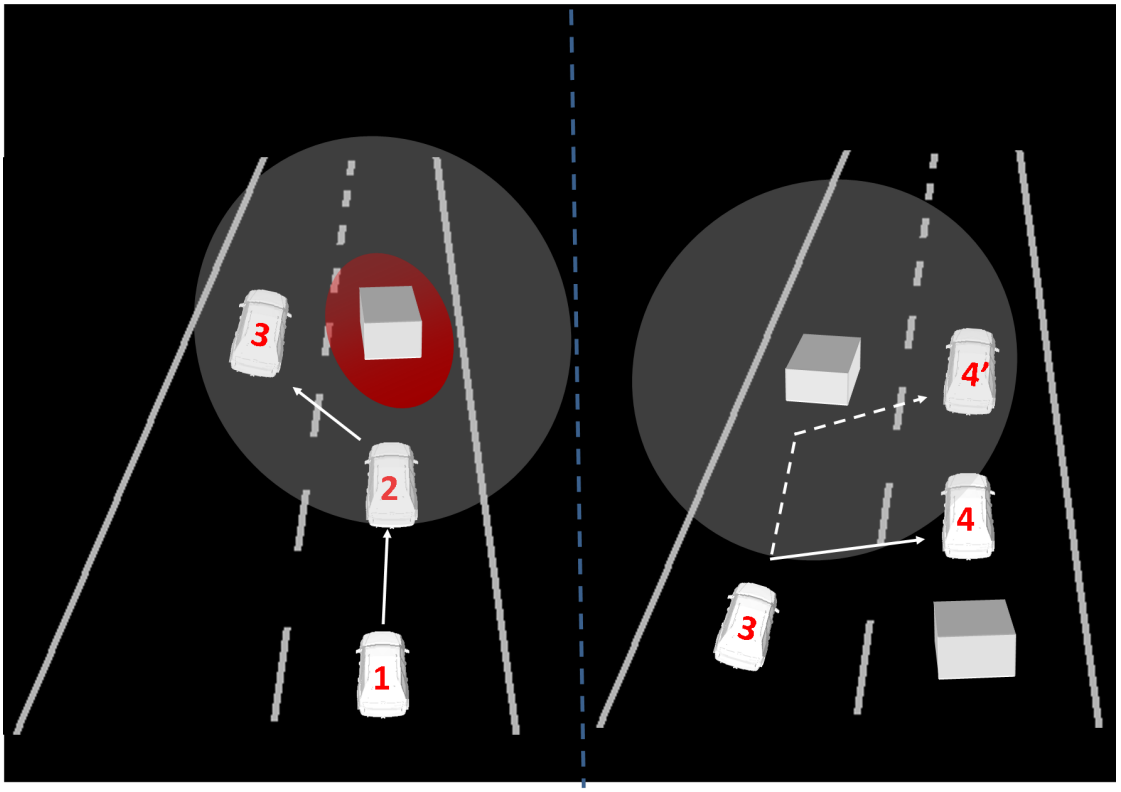


Figure 9: Overtaking state machine

1. Lane following - in this state the car is simply executing the program for lane following and in parallel scanning the environment. Since the overtaking has not been implemented there are two possible approaches to use the Lidar for scanning the environment or to use ultrasound and infrared sensors similar to parking scenario. The Lidar could provide much more data and much faster but there are also some limitations in the Odroid for processing that much data. In fact running the Lidar together with the lane following algorithm might overload the processor. Same also goes for the other sensors. So in general the speed needs to be limited to 0.5 m/s with the current software and hardware.

2. Obstacle detected - in this state some of the sensors have indicated obstacle on the lane. This means that the car is waiting for appropriate distance at which the

maneuver needs to be done. This distance is approximately 20 cm before the obstacle. That gives the car 400ms to the the maneuver.

3. Switch lane - when this 20 cm are reached the car gets a command to switch the lane, this affects the goal calculations, now the next goal will be placed in the left lane instead of the right lane. The car has 8 cycles to correct if something goes wrong, before it hits the obstacle.

4. Obstacle overtaken - the sensors have detected that the right lane is currently free, so the car can go back. Again the car waits for 20 cm distance before gets back. The only time it doesn't is if it detects something on the road in front in the left lane, then it immediately goes in the right lane.

Although not implemented the concept is pretty simple and should work. The big challenge will be to make sure that the processor won't be overloaded.

3.4.2 Intersection handling

The intersection handling is also handled by a state machine, although it is very simple. First the car is in a normal lane-following state, we won't consider this state as part of the state machine. After that in order to enter intersection handling and go to the first state the car must detect that there is an intersection upfront. That happens when the lane detector finds a big rectangle (with an area bigger than a certain threshold) and that this rectangle is lying on the right side of the dash lines. At that point the car enters the state machine for intersection.

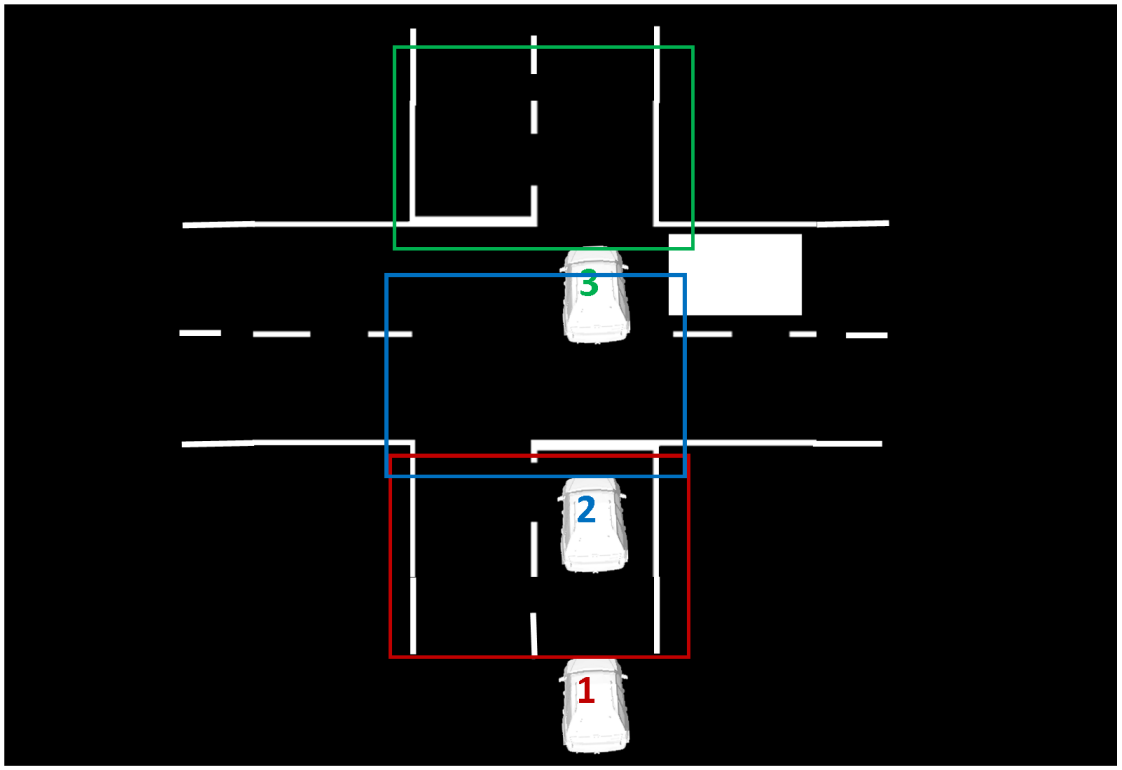


Figure 10: Intersection Handling

1. Intersection upfront - this state as shown on Figure 10 is due when the car is within the boundaries of the red rectangle. When the car stop seeing the stop line, it goes to the next state. In this state the car reduces speed to 0.5 m/s and follows the lane normally.

2. Stop/start state - when the code enters this state it immediately sends the stop command and the car stops. It waits 2 sec and during that checking the spot mark with a big white rectangle if there is an obstacle or not. If there is no obstacle after the two seconds the car continues, otherwise it waits for the obstacle to move from the spot. When the car is allowed to continue it doesn't do lane following any more. It is just going straight with a speed of 0.5 m/s. The reason for that is that it could pick up wrong path if the car is not position properly. For example if the car is slightly pointing to the left it could turn left instead of going forward. The blue rectangle approximates the place where the car is in "stop/start" state. To go to the next state the car needs to successfully identify at least two dash lines and one solid line.

3. End intersection - when the car goes to this state it reverts back to the speed before going to "intersection upfront" state and turns on the normal lane following.

3.4.3 Code organization

This functionality was not fully implemented, but the initial idea was to have the code organized into four high level Open DaVinci components - Lane detector, Driver, Lidar and Sensors - plus one low level component executing on the Arduino board. The plan was to run the lanedetector at 30Hz, the driver at 20Hz and the Lidar and the Sensors at 30Hz. The Arduino is listening for commands every 10ms to ensure proper driving of the car. The same commands as those described for lane following are used. One problem was that with this configuration the processor gets overloaded, so we even if we had the code we would have been forced to use only one of the components Lidar or Sensors.

3.5 Static competition

In the static discipline the team needed to make a presentation about the car. The presentation structure was provided by the jury and it should be in the following order - Car Concept(Hardware, Software, Cost and Energy Consumption), Lane following (Concept and Control), Parking (Concept and Control) and Overtaking (Concept and Control). The judgment was based not only on the presentation itself and the content but also on the skill of the presenter. The idea initially was to have 20 pages 1 for each minute, but it was discarded because it brings too much information on the slides and makes it difficult to explain. The final presentation had 35 pages with the title and questions pages. Unfortunately this discipline was very underestimated by the team and there was actually no time for practicing. The presentation was ready a couple of hours before the competition. Besides that the content was good and some of the parts were very appreciated by the committee. But because there was no practicing the presentation

had to finish a few slides before the end and the concept for intersection handling was not presented to the judges.

4 Competition Results

The competition days were full of surprises and most of them cost Legendary team a lot of time and stretched nerves. The team finished on 9th position out of 13 teams. The car managed to compete successfully only in one dynamic discipline - the lane following. There were several problems preventing it from competing in the parking and the car was not ready at all for overtaking.

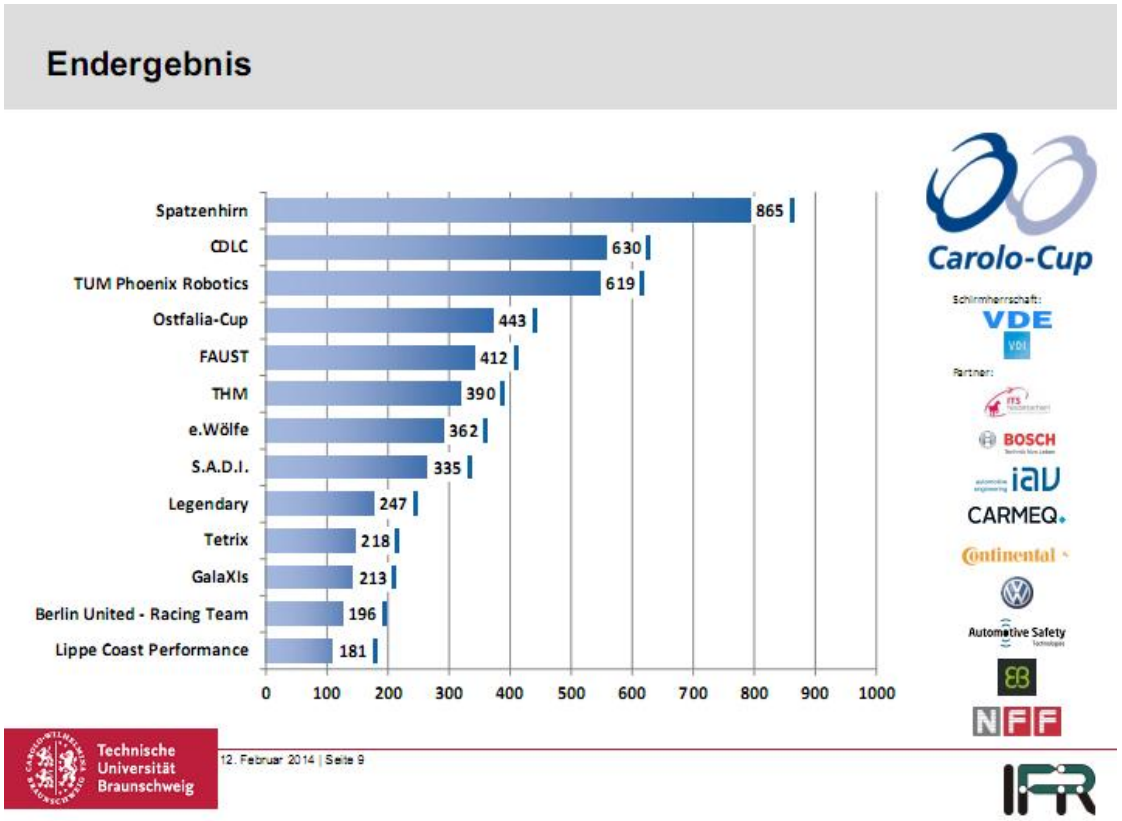


Figure 11: Overall Competition Results - Carolo Cup 2014 [6]

4.1 Lane Following

In the lane following the car manged to drive 106 meters for the 3 minutes time slot. The car speed was set to 0.9m/s. There were three interactions by the remote control on spots where there were missing lines and one on the spot for intersection. The interesting thing is that just before the competition during the calibration and training session the car was able to drive 1.5m/s without significant problems. The reason for lowering the speed was low battery and hoping for better precision in driving in lower speed. Unfortunately this performance get the Legendary team only 18 points out of 200 ending up in 9th place.

4.2 Static competition

The static discipline was the competition that gave the most of the results. The amount of points got was 229 out of 350. Parts of the concept that impressed the judges were Hardware Concept, Software Concept, Cost and Energy Efficiency. On the algorithmic part Legendary team get a lot of points on Lane following, but the both the parking and the overtaking concepts were lower mainly as satisfactory for the parking and poor for the overtaking. The main reason for that was that the time was not enough for the team to present the intersection handling part of the concept. That automatically meant 30 points less. For the presentation skills of the presenter again good score was achieved the only problem was again time management, which cost 20 points. So very easily lost 50 points, which could have placed the team a lot more higher in the competition results. The final place was 8th.

5 Discussion

In this section the focus will be on the flaws and benefits of the used technologies and algorithms, also on what should be preserved for later use and what should be discarded.

The Legendary team considered a first priority getting a running car for testing the software as soon as possible. It turn out later on that this was the best decision and probably if it was not done at the beginning the team would not be able to compete at all. There were several problems especially when building the shell and that time might have been spent somewhere else, but in general hardware design and implementation was always ahead of the software, which eased the work for making the algorithms as practical and as close to the actual environment as possible.

The second priority was the lane following as a prerequisite to all other disciplines. In fact it turn out that focusing at the beginning on the parking would have probably got more points. The reason for that is that even with hard coded pattern for parking the car could park pretty fast - just before the competition the Legendary car managed to park for 12 sec, which would have gotten the team around 100 points. But the algorithm did not work on the competition because the software was not mature enough. There were problems with processor overloading, commands missing or flooded buffers in the low level Arduino board, which could have been solved and instead of gaining 18 points in lane following the team would have had 100 in parking. The parking algorithm development took only 3-5 days and could have given some motivation and inspiration to the team, that the car could actually compete in at least one event. The lane following on the other hand is a very complicated challenge. There are a lot of factors that need to be considered and developing a stable algorithm requires time. Starting from scratch for lane following is pretty difficult, it also depends a lot on hardware changes and electronics changes. Besides that the technique for lane following developed by the Legendary team has a lot of advantages - it is simple and straight forward and performance wise it is very fast compared to other more complex algorithms. It is also build on top of a stable and easy to use platform such as Open DaVinci. The image processing on the Odroid board took in worst case 20ms, which allows processing of 50fps, since most of the cameras have 30fps it gives room for improvement and refinement. Possible

suggestions:

1. Detect lines on the road. Everything till and including the pick up in the algorithm worked perfect. The car always finds enough lines to make a decision, the problem in this part of the lane detection arises in the classification of the lines. In other words the question is a line a solid or dash was not answered correctly by the heuristic function in 100% of the cases. In fact the percentage was around 70% and because of that some rules were added to discard lines that are difficult to classify, which increased the proper classification of dash lines - up to 90%. That on the other hand created more problems when it comes to calculating vanishing point since it increased the cases when we have only dash line recognized. A suggestion here is to use neural network trained on test data to classify the lines into dash and solid. The neural networks are very good in classification problems when the classes for classification are not very much.

2. Vanishing point calculations. An improvement could be done when estimating the vanishing point in the case of only one lane present. As mentioned in section 3.2.2 the success rate of the empirical function was around 70%. A good idea is again neural networks, but it might be tricky to define the properties that need to be fed into it. Another option could be some Markov chain probabilistic approach that predicts the lane positions depending on the previous image and tries to match the lines with the corresponding predictions. Same idea but with Kalman filter could also be investigated. Finally more object tracking algorithms could be analyzed and fused with the current algorithm, that will allow object detection to be done only one enough information is present and otherwise just track the objects.

3. PID control. The PID control tuned with the twiddle technique worked really good with very little effort, but other algorithms might be considered if the speed needs to be increased in order to keep the precision of the driving.

4. Serial communication protocol. There were some problems with the serial communication. Sometimes the car processes old commands and sometimes doesn't execute certain commands. For that purpose at the competition CRC check sum was introduced and also message termination sign. Clearing buffers after processing a command was also done to make sure that old commands are not executed. Communication between the boards needs to be tested to the limits and the protocols needs to be reliable, otherwise the car would not obey the algorithms and will ruin the performance.

A few weeks before the competition the parking was put as a priority and must to have feature. The time was definitely insufficient that is why the car actually didn't do any lane following, but just executes a "go forward" command. There so many things that could go wrong with that command (like not aligned wheels, bad start positioning, etc.). It is much better to add simple lane following or use the sensors to correct the car. Also proper sensor should be chosen - in the Legendary car the infrared sensors on the side could detect only up to 30cm, and an obstacle could be 25cm inside so if the car is 5cm inside the road it might not be able to detect that obstacle and try to park in a taken spot. The speed also needs to be considered - the infrared and ultrasonics are pretty

slow sensors so it is difficult for them to operate correctly in high speeds. And finally instead of a hard coded pattern following a trajectory and compensating could save a lot and make a good base for lane following understanding later if parking is chosen as first to implement.

At the end a few words about the static competition. The static discipline is giving the most points, so it deserves a proper attention and should not be underestimated. It is obvious that you cannot start with building the presentation since you do not have anything to present, but it should be started at least one month before the competition. The presenter needs to definitely time the presentation and try to speak loudly, clearly and slowly (people there are not native speakers). It will be good if this presentation is held to people that do not know anything about the car in order to get an objective feedback if they understand what you are talking about. A simple calculation shows by itself that with good presentation - 270p and good parking - 100p, that is already 370 points as you can see on Figure 11 it actually means 7th place.

Final suggestion is that the next year students focus on two disciplines and try to make them as good as possible without making any effort on the third, maybe only a concept for the presentation, because two of them are already very challenging for just six months. Most of teams work on those concepts for 3-5 years.

6 Conclusion

This report summarizes the work of the Legendary Team and preparation of their miniature vehicle for the Carolo Cup 2014 competition. As first year competitors and starting everything from scratch it builds a good base for the next teams from Chalmers. Some of the ideas are innovative and could be extended and used from them, some algorithms are over simplified and some more work is needed, but in general the solution suffices the expectations. Since the team was only 3 and half people the work done was on the limit of what was possible to be done.

References

[1] C. Akinlar B. Benligiray, C. Topal. Video-based lane detection using a fast vanishing point estimation method. *2012 IEEE International Symposium on Multimedia*, 2012.

[2] Bradski and Kaehler. Basic thresholding operations - http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html.

[3] M.Venkatesh and P.Vijayakumar. A simple birds eye view transformation technique. *International Journal of Scientific and Engineering Research, Volume 3, Issue 5W*, 2012.

[4] Marcos Nieto. Vanishing point - https://marcosnietoblog.wordpress.com/2012/03/31/vanishing-point-detection-c-source-code/.

[5] Udacity Team Sebastian Thrun. Twiddle for pid - http://www.youtube.com/watch?v=2uq2bszdvxs.

[6] Carolo-Cup Organization Team. Carolo cup results - https://wiki.ifr.ing.tu-bs.de/carolocup/system/files/ergebnisse.pdf.

[7] Carolo-Cup Organization Team. Carolo cup rules - http://www.carolo-cup.de/fileadmin/user_upload/2014/regelwerk/hauptwettbewerb2014.pdf.

[8] OpenCV Team. Canny edge detector - http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_

[9] OpenCV Team. Minimum area rectangular fitting - http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html.