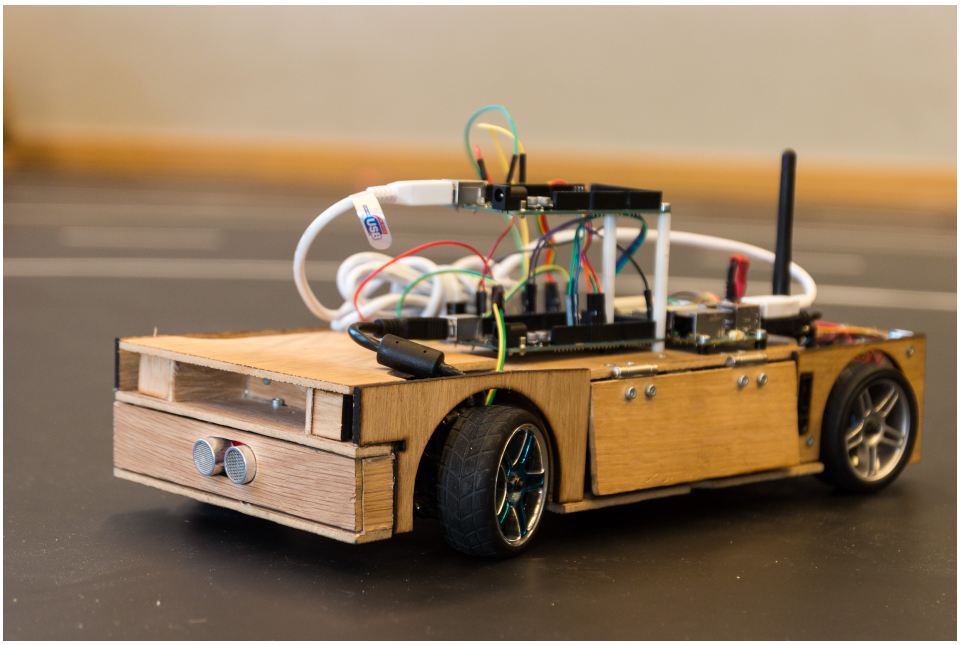
The LaneFollowing Module

Technical report for Carolo-Cup 2015

Jonas Hemlin
Brian Mwambazi
Andy Moise Philogene

June 15, 2015



Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg University
Gothenburg, Sweden 2015

Abstract

CaroloCup is held annually in Germany where students compete with self-produced autonomously miniature vehicles. The cars are evaluated in several disciplines where the basic functionality required is tested; parallel parking, lane following, overtaking, and intersection handling. Team MOOSE from Chalmers University of Technology and Gothenburg University competed in 2015 edition of CaroloCup and this report covers the work of a subgroup of the MOOSE team. This report describe the existing implementation of the previous year Legendary team and the improvement made by team MOOSE for the image processing of the camera frame on which the road markings are detected and also for the trajectories derived of how the car should move based on the found road markings.

Keywords: carolocup, lane following, autonomous vehicle.

Contents

1
Introduction

Carolo Cup provides student teams from universities the opportunity to develop and implement 1:10 scaled autonomous model vehicles with akerman steering, and compete with other teams. The teams compete in different disciplines characterized under static and dynamic events. The challenge is to realize the best possible vehicle control in different scenarios inspired by a realistic environment. The competition is divided into different disciplines, as follows:

- Parking: The car is expected to park along the right side of a straight strip, three different lengths 700, 630 and 550mm are distributed along the right side. Obstacles emulating parked cars can be present in these spaces increasing the challenge. A maximum of 200 points can be scored in this event.

- Lane-Following: The car is expected to drive on a scaled track, representing a rural road. The track has lane marking on either side and also dashed lines in the center. The team is allowed 2 minutes to continuously drive on this road. Point are awarded based on the total distance travelled and violations committed. A total of 200 points can be secured in this event. To increase the complexity, the organisers can remove one or more lane markings.

- Lane-Following with Obstacle Avoidance: This discipline extends on the previously done lane following. The car is expected to drive for a maximum of 3 minutes along the same road but also be able to avoid stationary and dynamic obstacles along the road. The car must follow driving rules by giving proper indications while overtaking, stopping at an intersection before proceeding etc. A maximum of 250 points can be secured in this discipline.

- Presentation: This is the only static discipline at the competition, here the team is expected to present and motivate technical and non-technical choices made for the car. The presentation is then compared with the running car to assign points to the team. A maximum of 350 points can be gained. This report provides an general system overview of hardware and software. Furthermore, we put highlight at bottlenecks that we experienced and consider important for next years students to consider.

This report presents the design and implementation of the LaneDetector module used in the 2015 Carolo cup project.First we give a background in section 2 then we will describe the design and implementation in section 3. Furthermore we will have the discussion in section 4 and end with the conclusion in section 5.

2
Background

In this chapter, we introduce the necessary background needed in order to understand the basic concepts in the report. Section 2.1 introduces image processing and the different concepts and algorithms which were used to achieve machine vision. Section 2.2 we give details about the type of hardware we used on this project.

2.1 Image Processing

Image processing is a type of signal dispensation where the input is an image like a video frame and the output maybe either an image or a set of characteristics and properties associated with the image. In general, most image processing systems treat an image as a two dimensional signal.

2.1.1 Image Representation

In digital systems a raw image (or video frame) can be represented as a two dimensional array or matrix where each entry is a pixel. A pixel itself is defined by fixed number of bits and the range of this number varies depending on whether the image is in gray scale or color. The number of bits is directly proportional to the number of distinct colors represented by the pixel. Thus the total size of an image depends on two factors; the number of pixels (resolution) and the number of distinct colors a pixel can display. Usually in computer vision to increase processing speed the image size is scaled down by varying these two factors. For instance, a color image can be converted to gray scale or the image can be cropped. The common coordinate system in images and used in this project has its origin in the top left corner of the image. In this system the X coordinate represent the width of the image and the Y coordinate represent the height.

2.1.2 Contour Detection

A contour is a closed curve joining all continuous points along the boundary having the same color or intensity. Thus given a blue object sitting on a green background the contour in this image will be the curve bounding the blue ball. Figure 2.1 illustrates contour detection on a color image.

Different contour detection methods with different properties exist in literature, in this project the detection function used from the image processing library OpenCV was based on the algorithm proposed by Suzuki and Be [2].

Figure 2.1: Effect of contour detection on a color image.(a) shows the original image while (b) shows the detected contours [1]

2.1.3 Polygon Approximation

In most cases covering lane detection, it is not a necessary requirement to have a perfect representation of the detected contour.Infact only an approximate outline of the contour would suffice as long as the overall shape is maintained. Polygon Approximation is an algorithm for reducing the number of points in a given curve that is represented by a series of points. The algorithm is also know as the Ramer–Douglas–Peucker algorithm, after the original authors. The idea behind the algorithm is; given a curve composed of line segments, to find a similar curve but with fewer points. It is a very useful function as it can be used to reduce the amount of data needed to represent a shape in an image. Thus increasing the processing speed as fewer points need to be traversed.

Figure 2.2: Effect of Polygon Approximation applied on the map of Europe.(a) shows the original map (b) shows the resulting approximation with less points used [3]

2.1.4 Bounding Box

Given a contour or a set of points in an image, a minimum bounding box can be found that encloses the points. The two variants of this algorithms where employed in this project. The first variant calculates and returns an upright bounding rectangle for a set of points. On the other hand the second calculates a rectangle bounding the

minimum area enclosing the set of points. The rectangle returned from this variant does not need to be upright.

2.2 OpenCV

In the project implementation we used the *OpenCV* [4] as the primary library for image processing. This is a cross-platform open source computer vision library written in C and C++ and has interfaces to other programming languages. It was designed for computational efficiency with a strong focus on real time applications. To achieve this the core of *OpenCV* is written in optimized C and can take advantage of multi-core processor architectures. With over 500 functions, the goal of *OpenCV* is to provide a simple-to-use computer vision infrastructure that can be leveraged in building complex vision applications quickly. The library has a rich Application Programming Interface (API) with a comprehensive documentation [5].

2.3 Hardware Limitation

The performance of an image processing solution is hugely influenced by the type of hardware used in the solution. The fundamental components that directly dictate the design of the solution include the CPU, Ram and Camera. In this project the ODROID-X2 [6] computer board was used as the main processing component. The board's consist of a 1.7GHz Quad core ARM CPU with 1 megabyte Level-two(L2) cache and 2 gigabytes DDR2 ram with 880Mega data rate. The design of our image processing solution had to fit within the constraints of this hardware. The camera used was a Ueye Ids with a wide lens having about 115 degrees viewing angle.

Despite having a four core processor, only one core was effectively used in running the the main image processing. This is because part of the system's non-functional requirements is real time performance and the nature of the problems our solution was trying to solve could not be parallelized without introducing unpredictable latency in the system.In order for the car to be reasonably responsive, the lane detection module had to process about 30 to 40 frames per second, this translates to about 33 to 25 milliseconds processing time per frame respectively. Any per frame processing higher than that will have consequences on how fast the car responds in lane following. The algorithms used in this module were chosen so as to fit in within the these constraints.

3

Design and Implementation

This chapter describes the software design toward the implementation of Lane-following. This algorithm is a continuation of the previous year "Legendary Car" students, while improving the overall stability of the software and introducing new concepts. The lane-following component is also an important basis for overtaking. As of now, the major code to achieve such behavior is already implement but needs further support at the Driver side.

In the remaining sections, we will present the requirements to achieve a stable lane detection/following, then explain the detection and filtering phases and finally present the derivation of the trajectory.

3.1 Requirements

The lane-following discipline is the most critical software implementation, as other components heavily rely on its accuracy.

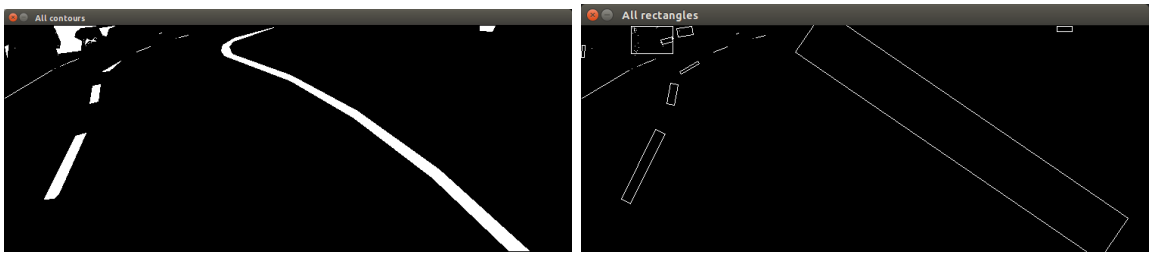
This module primarily receives input video frame by frame, originating from the uEYE camera with specifications defined in Section 2.3 and emits Goal Lines output in a data structure to the Driver for steering control. It is itself divided into two high level components: Lane-detector and Line-detector. The Lane-detection connects to the Supercomponent, from which it gets configuration and interfaces with the Proxy module to receive camera data over shared memory. This class also sets general parameters of the road and image processing, such as configuration data, thresholds, heuristics, etc. For each frame received, it invokes the Line-detector class to return line data. The Line-detector class is doing all the heavy image processing and this transformation will be explained further in the remaining sections.

This module, being critical in the steering decision, has to make all computation within a deadline. This deadline depends on the speed of the car, the hardware processor in use and the camera's frame per second rate. The faster the car drives, the more frames will be discarded in the steering decision.

3.2 Detection Phase

The image processing starts by detecting all possible shapes on the frames. The image is received in a gray-scale format from the Proxy. In order to reduce noise from the frames, we apply a binary threshold operation [7] , which means that if the gray pixel is higher than the set level it is transformed to the maximum value, otherwise the minimum value. This threshold is closely related to the light intensity

in the room. Thus, as an additional feature this year, we equipped the camera with a dynamic light sensitivity sensor to adjust this threshold value. After thresholding, we apply a Canny algorithm [8] to find the contour of each object. The result provides an array of irregular shapes. Next these shapes are fitted into the smallest rectangle possible. At the end of this stage, we have an array of rectangular objects from the image, as displayed in Figure 3.1 .



(a) Contours of the objects **(b)** Fitted rectangles of contours

Figure 3.1: Result computed rectangles from image

3.3 Classification Phase

The rectangles generated are not all acceptable ones. In this phase, we classify the rectangles into data structures that provide meaningful information at the later stages of the processing. The rectangles are classified into solid or dash lines. This classification is achieved based on the ratio of the length and width of the rectangles or simply their length compared to maximum rectangle length parameter. These parameters need to be calibrated on the track to find an optimal trade-off on the straight lines and curves.

At this phase, we also classify the state of the road. We currently distinguish three states: start-box, normal, intersection. The area position and angle of the rectangles is used for the intersection or start box distinction and in the future for obstacles in overtaking mode. If the rectangle has a considerable area and is directed at a relatively flat angle at a certain position on the screen, the intersection mode is triggered. The road trajectory will be calculated differently at this state, as it will be explained in subsection 3.5.2.

3.4 Filtering Phase

The markings go through a series of filters to drop the lines that are not supposed to be use to calculate the trajectory. This process is highly important as disregarding good lines may leave the car with no trajectory while allowing improper lines may lead the car to follow an erroneous trajectory. Figure 3.2 shows the lines that are retained as relevant after the filtering phase.

General Filtering

The validity of the lines are checked based on the angles and relative positions between solid and dash lines or simply their position on the screen. It is unnecessary to consider lines that are too far on the left or right side of the screen beyond the solid lines, which may lead to an inaccurate calculation of the vanishing point. Lines that are too far ahead are also filtered out as the car should be guided by lines that are the closest to it.

Figure 3.2: Road Markings retained after the general filtering stage.

Characteristic Filtering

In the last stage of this phase the lines are filtered with respect to road characteristics. The purpose of this stage is to detect the three types of road markings that are used to derive the trajectories in the next phase. The three road markings are the left, dashed, and right road marking. It is not certain that the dashed and solid lines found prior this stage results in that all three road markings are found, but the algorithm aim to find all three. The algorithm only maps one line as a right road marking and one line as a right road marking, but maps as many as possible as dashed road markings. This is as the right and left line in the frame most probably is detected as one line, which is contrary to the dashed line which, naturally, is detected and categorized as several dashed lines. In the algorithm the dashed road markings are first targeted and then the left and right road markings assessed.

In the first part of the characteristic filtering stage the algorithm aim to find a dashed curve among all lines which are classified as dashed lines. Before the lines are processed they are sorted in descending order based on the y-coordinate of the point that is lowest in the frame. Then the algorithm pics the first dashed line in the vector and add this as the first line in the dashed curve. All other dashed lines are evaluated against this one in the vector order; i.e. the dashed line that is first evaluated against the picked line is the line that is second lowest on the frame. The algorithm successively works through the vector of dashed lines and if a number of conditions are met, the dashed line is appended to the dashed curve and the remaining lines in the vector will then be evaluated against the newly appended line. When the entire vector has been traversed, a dashed curve is found if the dashed-curve vector has a size of at least two.

As there can be a lot of interference in the frame it is not certain that the dashed curve found is the real dashed curve presented in the raw frame, the algorithm to find the dashed curve is executed until no dashes can be put together as a dashed curve. The dashed lines already used in a dashed curve are removed from the set of dashed curves before the algorithm is run again. When the algorithm has finished, several dashed curves may have been found. To choose which dashed curve to use, they are evaluated against the dashed curve used most previously. Some classified dashes may not be a part of a dashed curve and those can potentially be miss-classified solid lines. If such an unused dash line overlap with the picked dashed curve, it is reclassified as a solid line and appended to the solid lines pool.

If a dashed curve can not be found or if there only is one line classified as a dash, a different algorithm is applied which aim to map one classified dashed line to the dashed road marking.

The left and right road markings are found in an easier manner, and the algorithm for them is very similar. These algorithms assume that the left and right road markings lean inwards; the left road marking must have an angle of more than 90 degrees relative the base of the frame and a clock-wise degree notation, similar must an right road marking have a angle of less than 90 degrees. This assumption makes it easy to classify the solid lines as either aspirants to the right or left road marking. If taking the left road marking for example, the solid line picked is the one among the aspiring lines that is the most to the right in the frame. Also, this line is evaluated against the previously solid line which was used as the left road marking, which prevents that a line is picked that is far off from where the left road marking is assumed to be. The right road marking is chosen similarly.

The major improvement made to this stage is the development of the dashed curve. The former algorithm was only capable of picking one dashed line and use it as a dashed road marking, whereas now it is more certain that the real dashed line on the road has been found as we form a dashed curve out of multiple dashes.

3.5 Trajectory Phase

Trajectories of the car's future movements are derived for each lane based on the lines provided by the filtering phase. A trajectory is formed by a collection of goal lines, where a goal line consists of a road section's vanishing point and a point that is placed in the middle of the lane. When the algorithm is not in intersection state (i.e. in normal state), two types of road markings has to be detected by the preceding phase in order to derive a goal line. However, if only one of the three (left, dash or right) road markings have been found it is possible for the algorithm to estimate the missing marking. If the algorithm is in intersection state, the road markings can not be used to derive trajectories due to that an intersection introduces a lot of noise and that the lines are easily misinterpret. Instead is the large rectangle used to derive the trajectories. Figure 3.3 illustrates the trajectories for the right and left lane, where the right trajectory is marked with purple and the left trajectory with orange.

Our larger improvements are two with respect to the trajectory phase. The former algorithm had a trajectory consisting of one goal line whereas it is now possible

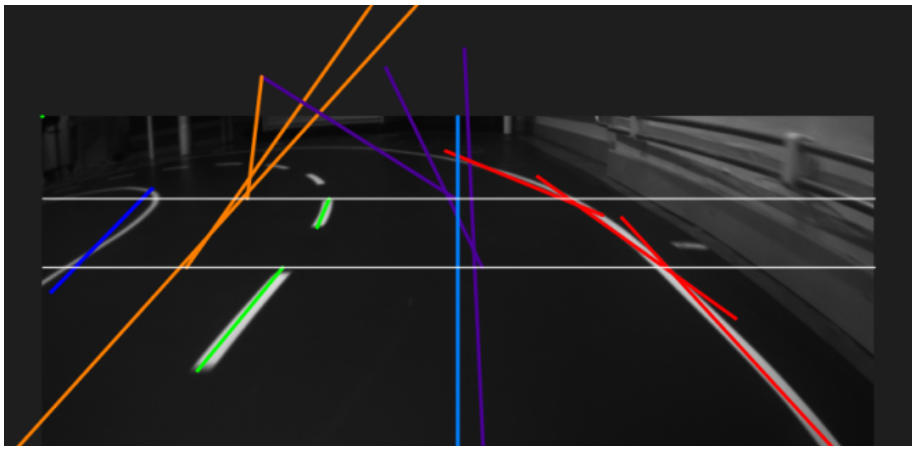


Figure 3.3: Presentation of the trajectories as well as the road markings used to calculate them.

to have a trajectory consisting of arbitrary number of goal lines. Additionally, trajectories for both lanes are provided, where the former algorithm only provided a trajectory for the right lane. The second large improvement is that it is now possible to get a trajectory while in intersection state, which was not possible before.

3.5.1 Normal State

In the normal state consists the created trajectory of one or several goal lines. First is the lines split and organized in several road sections. Second is lines estimated if there is not at least two different types of road markings present in respective road section. Last is two goal line derived per found road section, forming one trajectory for each lane.

Splitting of Solid Lines

The solid lines are split with respect to the dashed curve and then organized in road sections. The top point's y-coordinate of every dash in the dashed curve are used as a position where the solid lines are split, but if no dashed curve is found the solid lines are split at default y-coordinates. A road section is formed from a tuple of the road markings residing in the same y-space below the first split, between two splits, or above the highest split.

The polygon that the line was derived from is used to split it. The points forming the polygon are placed in different road sections (bins) by comparing the points' y-coordinates with the split points. All points in one bin are then used to create a line. However, this scheme produces lines with slopes that differs to much from the raw frame and to increase the slope's accuracy are the closest points in the closets bins included before converting the points to a line. In Figure 3.3 is the result of a split with three sections seen. The horizontal white lines separates the sections and as explained has the resulting lines some overlap. The left solid line in the figure has such a small number of points, giving that the whole line is present in both sections.

Estimation of Road Markings

If a road section only consists of one type of road markings, another type of road marking is estimated. The current algorithm favors road marking estimations that give the most accurate right lane trajectory, as the overtaking is not fully developed. Table 3.1 give information when and which road marking that is estimated. To estimate a road marking are two parameters predicted: the lane size at the bottom of the frame, and the angle of the estimated road marking. The "Legendary Team" made a MATLAB function that was fed empirical data to derive relationships between the different road markings, and those relationships are the main component in estimating a road marking.

Case	Provided road marking(s)			Estimated road marking
	Left	Dash	Right	
E1	-	-	-	-
E2	-	-	X	Dash
E3	-	X	-	Right
E4	-	X	X	-
E5	X	-	-	Dash
E6	X	-	X	-
E7	X	X	-	-
E8	X	X	X	-

Table 3.1: The inputs that can be given to the estimation stage as well as the stage's action is presented.

Compute Goal Lines

For each road section is one left lane and one right lane goal line produced and the procedure differs depending on the road markings given from the estimation stage. The stage has two parts, where the first part is to calculate the goal lines possible given the input and the second part uses the calculated goal line to specifically give a left lane and a right lane goal line. Table 3.2 presents the four possible input cases and the goal line(s) directly calculated from them, which constitute the first part. In any of the case G1-G3 is the second part executed due to that we don't have calculated both a right lane and a left lane goal line.

In the second part is the already calculated goal line used as a base. A goal line consists of two points: the road section's vanishing point and a base point which is in the middle of the lane in the bottom of the road section (closest to the car). If the vanishing point of the road is assumed to be the same as a lane's vanishing point it is possible to create the desired goal line by modifying the calculated goal line's base point. The current algorithm assumes this and for example is a right goal line calculated in case G3 by shifting the base point from the left lanes center to the right lanes center . The value to shift with originates from the lane width, which calculated in and provided by the estimation stage.

Case	Provided road marking(s)			Directly calculated goal line(s)
	Left	Dash	Right	
G1	-	X	X	Right lane
G2	X	-	X	Whole road
G3	X	X	-	Left lane
G4	X	X	X	Left & Right lane

Table 3.2: A presentation of the input cases to the Compute Goal Lines stage as well as the output of it's first part.

A goal line's two points are computed by employing linear equations and two different road markings. The notation of the linear equations are of slope-intercept form, as shown by Equation 3.1. A_n and B_n are derived from the road marking's slope and one if it's two points. Then is the vanishing point calculated by Equation 3.2 and the base point by Equation 3.3. Road marking$_1$ is the left road marking whereas road marking$_2$ is to the right relative to the lane which goal line is being computed. The goal line shall begin at the bottom of the road section thus the base point y-value is set to the split position, or 0 if the current road section is the closest one to the car. The base points x-value is composed of the road section$_1$'s lower points' x-value added with half the lane width. If any of the road markings are estimated is the predicted lane width provided from the estimation phase used, instead of relying the lane width calculation on the estimated bottom point.

$$y = A_n \times x + B_n \tag{3.1}$$

$$\begin{aligned} x_{vp} &= \frac{B_1 - B_2}{A_2 - A_1} \\ y_{vp} &= A_2 \times x_{vp} + B_2 \end{aligned} \tag{3.2}$$

$$\begin{aligned} y_{bp} &= y_{bottom\_of\_road\_section} \\ x_{bp} &= \frac{y_{bp} - B_1}{A_1} + lane\_width \times 0.5 \end{aligned} \tag{3.3}$$

Creating the Trajectory

A trajectory is composed of a vector of goal lines and a vector of switch points. One goal line is valid given on value on the y-axis and these points define where the validity of the goal lines change. In Figure 3.3 are the switching points the points where the line are split. A such simple approach does not always give a smooth curve and to produce a better result a goal line's position and slope can be considered.

A line denoting the car's current position and heading is computed and providing along with the trajectories and the two vectors of switch points to the algorithm deciding driving decisions. The line marking the car's current position and heading is a vertical line in the middle of the frame and is used by the next algorithm to get a relation between a chosen goal line and the car's current position and heading.

3.5.2 Intersection State

When the algorithm is in the intersection state, the behavior for the trajectory phase is different. The different behavior is needed due to that an intersection introduces a lot of noise and that the lines are easily misinterpreted. Trajectories are produced only if the car passes a stop line on its way into the intersection. A big rectangle is then present incorporating the stop line and the big rectangle's rotation and position is used to compute a goal line. This goal line is used as the right lane trajectory and by shifting the goal line's points is the left lane trajectory produced. Figure 3.4 illustrates the two trajectories computed when in intersection state as well as the line denoting the car's current position and heading. The trajectories are then provided to the algorithm deciding driving decisions.

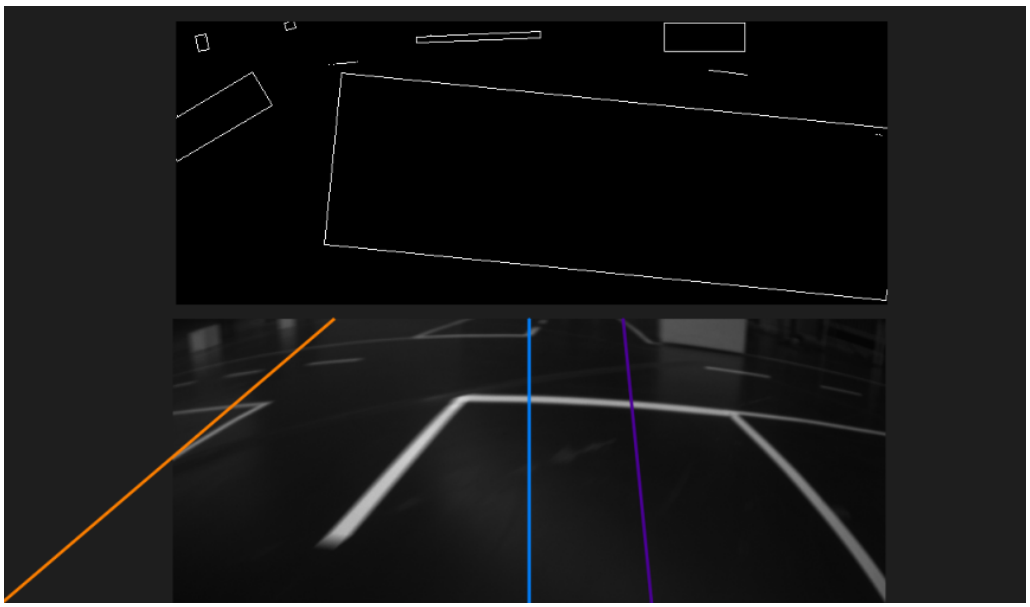


Figure 3.4: Presentation of trajectories provided in intersection mode as well as the rectangle used to compute them.

The algorithm goes back to normal state providing certain conditions. Basically these conditions are met when the car picks up appropriate road markings that close to it. Figure 3.5 shows the conditions that trigger the transition in a state machine-style diagram.

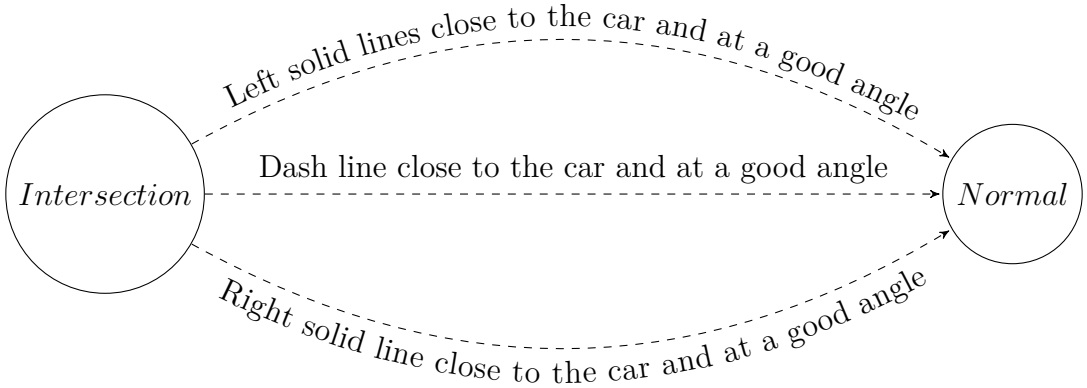


Figure 3.5: Transition from Intersection to Normal state

4

Discussion

In this chapter, we discuss the algorithm and lesson learned from the competition experience. The first task that was carried out was to refactor the algorithm in a more modularized fashion. We organized the code to the different phases described in this report so that each component could be inspected and improved separately. Following are several sections which bring up points of discussion regarding the mentioned algorithm's phases.

4.1 Detection Phase

The detection phase, although at an early stage of the algorithm, presents some challenges and issues. The main challenge during the competition was the fact that the ambient light of the environment is not constant at all. When coming at a certain angle, the car faces different beams of light. The dynamic thresholding technique in Section 3.2 helps massively, but sometimes this change can cause some lane markings to be discarded. As this solution was introduced very late in the process, more extensive tests need to be performed. Increasing or decreasing the speed of the car in such situation is always a trade-off. The safety measure adopted is that the Driver module slows down the car when no trajectory is supplied to the module, which gives a higher chance of detecting lines and also prevent that the car from dashing off the road during trickier scenarios.

4.2 Classification Phase

The classification phase from the previous team was well thought of and designed. We observed few cases that cause instability in the execution of the algorithm. The main problems from the Legendary team design is that the car wrongly classify lines when it is at a curve or when other markings are connected to the lines, such as parking markings. Also, at the curve, dash lines become longer than usual or curves are considered as big area thus intersection, due to the camera type and the perspective view of the car.

The main solution we implemented was to break the markings up to a certain level on the screen, at a later stage of the processing. This idea makes the contours shape more relevant and makes the vanishing point calculation more accurate. This approach is discussed in the following section.

4.3 Filtering Phase

Our improvement within the characteristic filtering to look at several dashed lines and connect them has one major advantage. The major benefit is that it now is more certain that the right dashes actually are found. Before when only treating the dashes individually, it was not possible to have this degree of confidence as we can now when looking at the dashes as a group.

There are some shortcomings in the current algorithm. One is the assumption about the angle of the left and right road markings. In some situations where there are sharp turns and the car is positioned in an inner-curve this angle assumption does not hold for the solid line detected in the inner-curve. This results in that either is the line filtered away or it is wrongly mapped to an incorrect road marking due to the angle assumption. If the line is incorrectly mapped and no dashed road marking is detected, the car's trajectory will be incorrectly derived and the car may potentially drive off the road. Although, this angle assumption makes the algorithm simple and effective and these cases are quite rare, but the algorithm need some minor tweak to handle these situations.

Another limitation is within the creation of the dashed curve. As it is performed now, one dashed line can only be within one dashed curve and it is in the first curve the line fits into. This is a limitation since it may fit better in another dashed curve. We have tried setting up the checks within the function in such a way to minimize that lines are appended to dashed lines which they do not belong to. Also, to create dashed curves of all possible combination of dashed lines require more computational power which we believe it is better to spend elsewhere. This belief is based upon that the dashed curve in its current implementation performs well and we seldom saw any faults in the decisions made by the algorithm.

When the car approaches a missing section of left or the right road marking it is probable that the solid lines are detected as dashed. This as they will be seen as short lines due to that some sections are missing. It is hard to handle this fault during the classification phase and it is more straight forward to perform these tests during the characteristic filtering stage. What happens is that the short left or right road marking is classified as a dash line. If this incorrectly classified dashed line is mapped to the dashed road marking, the car's trajectory will be severely incorrect and the car may turn off the road. Although, currently we do not perform any such tests to detect these incorrectly classified lines and we rely entirely on that the speed of the car will perform as a filter. By this we mean that the next trajectory derived probably will not include this faulty line due to it now has been passed, resulting in that the car will steer back into the lane when using this new trajectory.

What can be done to prevent that the incorrectly classified dashed line is picked in the issue described above or that the solid lines are mapped to the wrong road mapping due to its angle is to store information from several frames back. This is something we have not improved, but it has a large potential to make the algorithm during this stage more stable. To keep track of where the road markings most likely will be in the frame which is currently processed, the process to discard lines which clearly should not be mapped to a respective road marking becomes easier. In the current implementation and also for the last years implementation only the last used

road marking is stored, which can not be trusted to a high degree to show where the next road marking also should be found. This as the last road marking may have been incorrectly picked, but if a history is kept of lets say the last ten or twenty used road markings one can trust the assumed position of new road markings to a higher degree.

4.4 Trajectory Phase

In this section we have divided the discussion between estimations and trajectories.

4.4.1 Estimations

The accuracy of the road marking estimation is in several cases not that good. When trajectories with several road sections were introduced several new cases were added where estimation have to be performed. As we have not modified the core of how estimations are performed to specially support the new cases it is not strange that the estimation accuracy of the new cases are not that good. Also, the estimation accuracy for the cases which are covered by the algorithm do not perform good in road sections which are far from the car, which can be seen in Figure 4.1 and Figure 4.2. This as the original algorithm did not have to consider a distance from the car, which now is the case. We made a simple work-around from this which was not that good, which make us propose that it is best to look into the MATLAB function created by the Legendary team and redesign it to also support the new cases as well as road sections a distance from the car.

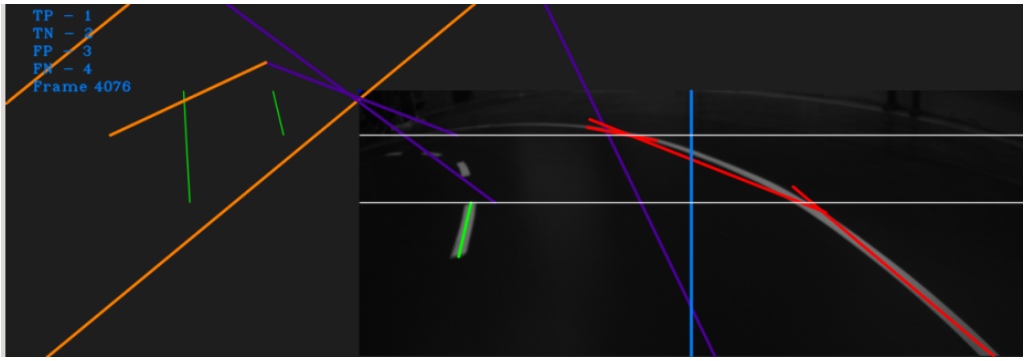


Figure 4.1: Presentation of where the dashed road markings is not correctly estimated given right road markings. The slim green lines are estimations for dashed road markings, which are not that accurate.

4.4.2 Trajectories

The trajectory passed to the driver can be more sophisticated. Now the trajectory consists of a vector of goal lines and a vector of switch points. From the two vectors, a curve can be derived that can be passed to the Driver module. This will give a better basis for deciding upon what steering angle to chose. To create this

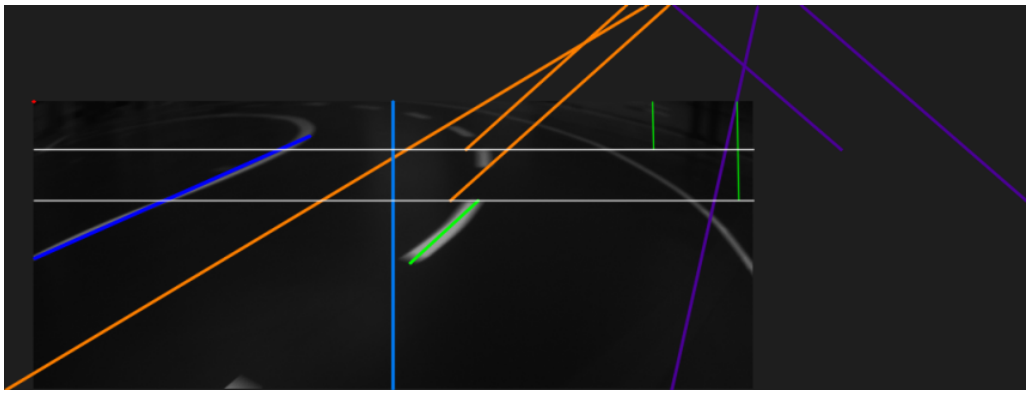


Figure 4.2: Presentation of where the dashed road markings is not correctly estimated given left road markings. The slim green lines are estimations for dashed road markings, which are not that accurate.

trajectory curve our idea is to identify at least three points that lies on this curve, which are converted into bird view by transforms. When converted to bird view it is substantially easier to map a curve to the points. Also, we believe that the driver decisions may become more accurate if they originate from this bird-view trajectory curve.

We implemented the code so it would not matter in which lane the car was positioned and this is so the code can be used for overtaking and obstacle avoidance. To support for overtaking and obstacle avoidance, the code provides one trajectory for each lane. However, such scenarios have not been tested, as the Driver module did not support that.

A fault we have seen is goal lines that points downwards. This happens when the goal line is computed of two lines whose vanishing point resides below their road section. This behavior is displayed in Figure 4.3, where the goal line of the third road section is pointing downwards. This issue have at least two reasons. The first reason is that the detected lines seldom are precisely aligned with the markings found on the original frame. The other reason is that the lines are skewed because of the fish-eye effect the lens has. To overcome this fault, a simple solution is to rotate the down-pointing goal line 180 degrees. Although, this maybe does not solve all types of this fault and has to be tested.

The contribution with a trajectory including several goal lines was unfortunately not used to the extent it was presented in Section 3.5 during the competition. This as the parts within the algorithm that decides on driver decisions (the Driver module) did not support trajectory prediction and that car lacked the needed processing power. What was used at the competition was a trajectory that was composed of only one goal line. As the Driver module do not support predictions nor overtaking of obstacles, the trajectory with several goal lines could not be extensively tested and it remains as future work to evaluate this improvement.

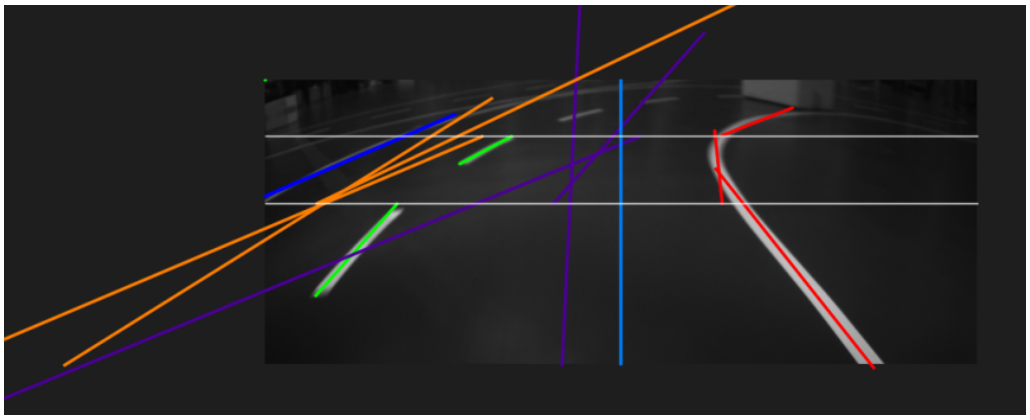


Figure 4.3: A case where the goal lines for the third road section is pointing in the wrong direction.

4.5 Intersection handling

Intersection handling remains a challenging task for us. The first approach we took was to split the big rectangles to get a better shape of the contour that was detected. The challenge was that not all intersection have stop line. In that case the big rectangle was detected on the side of the screen in small amount of frames. Splitting the rectangles wouldn't give any deterministic information that we are actually in an intersection.

The solution we have implemented is an improvement from last year, however the car still fails sometimes. What matters the most regarding passing an intersection smoothly is what angle the car has when it goes into the intersection. If this angle is bad, say more than maybe 30-35 degrees from the "right" angle, it may be really hard - if not impossible - for the car to detect the lines needed to transition to the normal state and straighten up with regard to the lane. An improvement from last year was to provide a trajectory within the intersection state to try to give the car a better angle into the intersection. Even better would be if one can derive a trajectory based on lines present at the far side of the intersection, as this would give a more accurate trajectory. Although, the level of interference is very high in intersections and to extract trustworthy data from that environment is a delicate task, which may not be possible due to that information needed possibly is discarded in the detection process.

A fault which we encountered was that the car became stuck in the intersection mode. This happened even though the car must have seen the proper lines needed to change back to the normal state. To solve this we applied a somewhat dirty fix which consists of a simple time-out which forces the algorithm back to the normal stage after 0.8 s. This fault did occur not only in intersections, but in any place where the intersection mode may be triggered, as in the startbox. Unfortunately, we do not know why this happens.

4.6 Performance

In this section, we show the performance of the different stages of the algorithm. This version tested is the algorithm that was used at the competition, with the calculation of a trajectory with only one goal line on the right lane being computed. The other goal lines were not calculated due to a lack of computing resources on the car. The tests are performed on recorded videos of the track that we had built in-house, using the OpenDavinci framework on an Intel dual-core 2,40GHz computer. The ideal methodology would be to get the results with the algorithm running on the actual car, but unfortunately the track was dismantled soon after the competition.

As each stage of the algorithm was clearly encapsulated in a function this year, it becomes straightforward for us to compute their running time on each frame. Table 4.1 gives the averaging running time in microseconds at each stage for different scenario. Table 4.2 uses the same data but shows how much computational resources are spent at each stage in terms of percentage.

	Contour	GetRect	Classif.	First Filters	Charact. Filter	Trajectory	Total(µs)
Straight Line	1770.23	1223.15	28.92	9.00	654.46	887.92	4573.69
left Curve	1657.92	1261.46	167.92	5.85	615.31	778.00	4486.46
Right Curve	1939.89	1631.39	109.06	11.50	986.00	1104.06	5781.89
S Curve	1824.61	1392.56	137.78	7.22	744.50	929.56	5036.22
Intersection	1060.47	1069.84	188.95	5.32	508.74	636.00	3469.32
Missing Markings	1207.28	1241.00	19.78	6.89	568.11	630.39	3673.44

Table 4.1: Presentation of the time each stage takes in microseconds.

From this empirical data, we can identify stages that are slowing down the algorithm. The first two stages to get the contours of the images and bounding rectangles are using OpenCV libraries and offer little room for improvement. The last two stages of characteristic filtering and trajectory calculation are the ones that can be optimized the most. These stages mostly loop through all the detected lines mostly in an order of $O(n^2)$.

We have considered to compare the results between different scenarios, but we realized that this data is affected by the position where the recordings were done and other noise. For example, a straight lane that was recorded further away from other lanes will require less processing resource than a lane recorded near other road markings or wall nearby. In general, the higher the number of markings on the road, the higher the computational time will be in the first two stages. Since these stages consume the most resources, they will directly impact the overall computation.

	Contour	GetRect	Classif.	First Filters	Charact. Filter	Trajectory
Straight Line	38.70	26.74	0.63	0.20	14.31	19.41
left Curve	36.96	28.12	3.74	0.13	13.72	17.34
Right Curve	33.55	28.22	1.89	0.20	17.05	19.10
S Curve	36.23	27.65	2.74	0.14	14.78	18.46
Intersection	30.57	30.84	5.45	0.15	14.67	18.33
Missing Markings	32.87	33.78	0.54	0.19	15.47	17.16

Table 4.2: Presentation of the time each stage takes as a percentage of the total running time of the module.

5

Conclusion

In this report we presented the work done on the lane following module of the CaroloCup 2015 car. The work on the module was a build up from the previous year's implementation aimed at improving the algorithm as well as introduce new capabilities. Our contributions to the module resulted in that the car follows the lane smoother than previous years thanks to more accurate road marking detection and trajectory calculation. Also, the car managed to handle the intersection at the competition in most tries, something that was improved somewhat from last year. We have also provided a basis for overtaking by deriving trajectories for both the left and right lane. Additionally, we have proposed suggestions for the incoming students in order to achieve better result in that discipline.

Bibliography

[1] OpenCV. Finding contours in your image. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html

[2] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," Computer Vision, Graphics, and Image Processing, vol. 30, no. 1, pp. 32 – 46, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0734189X85900167

[3] S.-T. Wu, A. C. G. d. Silva, and M. R. G. Marquez, "The Douglas-peucker algorithm: sufficiency conditions for non-self-intersections," Journal of the Brazilian Computer Society, vol. 9, pp. 67 – 84, 04 2004. [Online]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0104-65002004000100006&nrm=iso

[4] OpenCV. Opencv. [Online]. Available: http://opencv.org

[5] ——. Opencv doc. [Online]. Available: http://docs.opencv.org/

[6] Hardkernel. Ordroid-x2. [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G135235611947

[7] OpenCV. Threshold binary. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html#threshold-binary

[8] ——. Canny edge detector. [Online]. Available: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html