PROJECT IN COMPUTER SCIENCE

# Self-driven Miniature Vehicle for Carolo Cup

ATTILA NAGY

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden November, 2013

# Contents

## Abstract

Carolo Cup is an international student competition for self-driven miniature vehicles organized annually in Germany. Such vehicles above providing certain functionalities must fulfill extra requirements, such as: time-constraints, and performance. The basic functionalities required for the Carolo Cup competition are lane following, crash avoidance, overtaking, intersection handling and parking. This report provides solutions for one of these functionalities, both describing its theoretical background and presenting a working implementation. The functionality covered by this report is lane following.



Keywords: carolocup, lanefollowing, image processing

# 1 Introduction

This report provides solutions for lane following functionality of self-driven miniature vehicles. The lane following contains the following parts:

- image processing,
- clustering of lines,
- pattern matching on clusters,
- defining input data for the control algorithm.

## 1.1 Image Processing

Image processing involves the processing of the frames from the camera, filtering noise on the frames, transforming the filtered frames to bird's eye view, and performing a Hough line transformation on the bird's eye view image to get the line shapes from the frame.

## 1.2 Clustering of lines

A density-based spatial clustering algorithm was used to group the neighbouring lines into clusters by their distance and minimum numbers of lines. This algorithm is an alternated DBSCAN (density-based spatial clustering of applications with noise)[Wik] algorithm. The classical DBSCAN algorithm operates on a set points and returns a set of clusters where each point is uniquely assigned to exactly one cluster. This version of the algorithm does not care about the fact that these points were connected once, and therefore should belong to the same cluster. The algorithm used in this project, on the other hand, operates on a set of lines and returns a set of clusters of points. This alternation of the algorithm provides more accurate clustering for the purpose of this project and requires less computation since we only iterate through half of the points.

## 1.3 Pattern Matching on Clusters

After having the clusters, there is only one relevant step remained, which is finding the clusters that contain the one dashed line and the two solid lines from the lane. This is done by finding the greatest difference on the x and y axes between any pair of points in the cluster and checking if these values fall within a predefined range.

## 1.4 Input data for the control algorithm

The control algorithm operates on two lines, the dashed line and the solid line on the right. Therefore, the clusters containing the dashed and solid lines must be transformed into lines that can be fed to the control algorithm. This transformation is done by checking the Euclidean distance between each pair of points in the given cluster to find the one with the greatest value. The solid line on the left is only used during overtaking.

## 1.5 Structure of the Report

This chapter gave a brief introduction to the topic and structure of this report. The next chapter, chapter 2, will give a thorough description about the image processing, line clustering and pattern matching methods. Chapter 3 shows the performance data measured on the target hardware. Chapter 4 provides a discussion covering deficiencies in the current implementation and design and propose possible solutions for these deficiencies. Lastly, chapter 5 contains the final conclusions of this project and summarizes its results.

# 2 Theory

This chapter gives an overview on the theoretical background of the methods used during image processing, clustering the lines and pattern matching.

## 2.1 Image Processing

The image processing contains the following steps:

- converting the image captured from the camera into grayscale,
- applying threshold function,
- finding edges in the filtered image using the Canny edge detection,
- transforming the edges to a bird's eye view image,
- finding line segments in the bird's eye view image by a probabilistic Hough transform algorithm.

All these steps except the bird's eye view transformation were performed by some certain OpenCV[Opea] function calls. Figure 2.1 shows the input and output images for each step. The Hough transformation image shows the Hough lines with red color and the end points of each line with blue.

As for the bird's eye view transformation[MP12], it has two separate phases. The first phase is to generate the transformation matrix M, while the second phase is to fed this matrix M to the warpPerspective[Opeb] OpenCV function. The following equations show the calculation of this matrix M:

$$A1 = \begin{bmatrix} 1 & 0 & -\frac{img.width}{2} \\ 0 & 1 & -\frac{img.height}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} A2 = \begin{bmatrix} f & 0 & \frac{img.width}{2} \\ 0 & f & \frac{img.height}{2} \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & distance \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y = \begin{bmatrix} 0 & cos(\beta) & -sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sin(\beta) & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_z = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 & 0 \\ sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R = R_x * R_y * R_z$$

$$M = A2 * (T * (R * A1))$$

First of all, the matrix *A1* shifts the image in the coordinate system such that its center is located at (0,0). Secondly, the rotational matrix $R$ can be calculated from $R_x, R_y$ and $R_z$ matrices. Thirdly, the translation matrix $T$ can adjust the height of the camera position by the *distance* parameter.

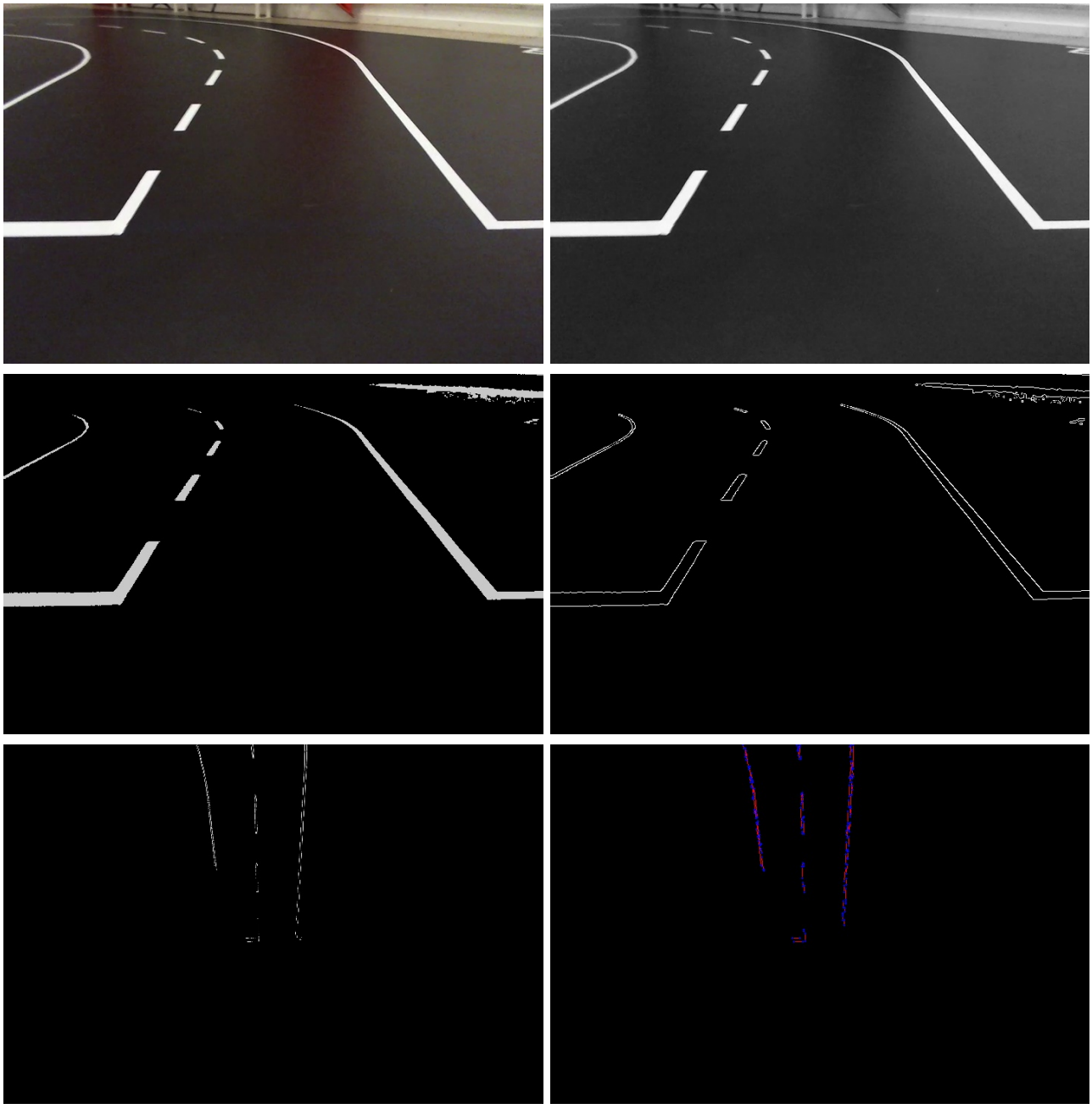

*Figure 2.1: Stages of image processing (top-left: original, top-right: grayscale, middle-left: threshold, middle-right: canny edge-detection, bottom-left: bird's eye view, bottom-right: Hough transform)*

Lastly, the warpPerspective OpenCV function is called with the WARP_INVERSE_MAP parameter which calculates the final image as follows:

$$BirdView(x, y) = warpPerspective(inputImg, M, WARP\_INVERSE\_MAP) =$$

$$= inputImg * (\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}})$$

## 2.2 Line Clustering

For clustering lines a modified DBSCAN[Wik] algorithm was used. Originally DBSCAN operates on a set of points and returns a set of clusters of points. Since in our case the input data is a set of lines, this algorithm must be adapted to this type of input data. It is done by checking if any end point of a line is density-connected to any point of a cluster, and if so, then it adds both end points to that cluster, not just the one that is actually connected to the cluster. Two points are density-connected if there is a third point such that those two points are reachable from that third point and each of these points has a sufficient amount of neighbours to form a cluster. A cluster satisfies two properties:

- all points within the cluster are mutually density-connected,
- if a point is density-connected to any point of the cluster, it is part of the cluster as well.

There are two parameters for this algorithm. The minimum number of lines required to form a cluster, and epsilon which defines the minimum distance between clusters. Code 1 shows the pseudo code for the DBSCAN clustering algorithm used in this project.

## 2.3 Pattern Matching

Pattern matching is needed to identify the clusters that cover the dashed line and the two solid lines on the lane. The current implementation uses a very basic approach. That is checking the maximum difference on the x and y axes between each pair of nodes and comparing these maximum values to a predefined threshold value. The maximum x value must be lower than the threshold for x, while the maximum y value must be higher than the threshold of y. In case of dashed lines, the maximum x value must fall into a range of acceptable values, since the length of dashes is fix and known. The only aim of this basic check is to find the line shaped clusters.

**Code 1** DBSCAN clustering algorithm

```
DBSCAN(D, eps, MinLines)
   C = 0
   for each unvisited line L in dataset D
      mark L as visited
      NeighborLines = regionQuery(L, eps)
      if sizeof(NeighborLines) < MinLines
         mark L as NOISE
      else
         C = next cluster
         expandCluster(L, NeighborLines, C, eps, MinLines)

expandCluster(L, NeighborLines, C, eps, MinLines)
   add L to cluster C
   for each point L' in NeighborLines
      if L' is not visited
         mark L' as visited
         NeighborLines' = regionQuery(L', eps)
         if sizeof(NeighborLines') >= MinLines
            NeighborLines = NeighborLines joined with NeighborLines'
      if L' is not yet member of any cluster
         add L' to cluster C

regionQuery(L, eps)
   return all lines within L's eps-neighborhood (including L)
```

# 3 Results

This chapter shows the periods of time that each step during the image processing, line clustering and generating input data for the control algorithm consumed. The input data for the control algorithm is the one dashed line and two solid lines from the lane. In addition, an example for clustering and pattern matching on an arbitrary image of a lane is shown in section 3.2.

## 3.1 Run-time Properties

All the measurements presented in table 3.1 were performed by using a 1.5Ghz Core2 Duo T5250 Intel processor, and executed on a subset of the frames of a recorded video. The frames that were blank after the Canny edge detection were excluded from the data set. Table 3.1 shows approximate average values measured on the selected set of frames.

*Table 3.1: Run-time of each steps*

| Step | Period $ms$ |
|---|---|
| grayscale | 2 |
| threshold | 0.2 |
| Canny | 8 |
| bird view (calc. M matrix) | 0.05 |
| bird view (warpPerspective) | 27 |
| Hough transform | 5 |
| DBSCAN (100 lines) | 7.5 |
| Finding lines for control alg. | 3 |
| Overall | 52.7 |

## 3.2 Clustering and Pattern Matching

Figure 3.1 shows the steps for defining the input data for the control algorithm. These steps are clustering the Hough lines, left picture on figure 3.1, and then pattern matching on the clusters to find the two solid lines and the one dashed line, right picture. On the clustering figure, each cluster is shown by a different color. It can be seen that each dash in the dashed line were put into a different cluster. By this strategy a simple pattern matching on the greatest difference on the x and y axes of any pair of points in a cluster is satisfactory to decide whether this given cluster covers a dash in a dashed line or not.

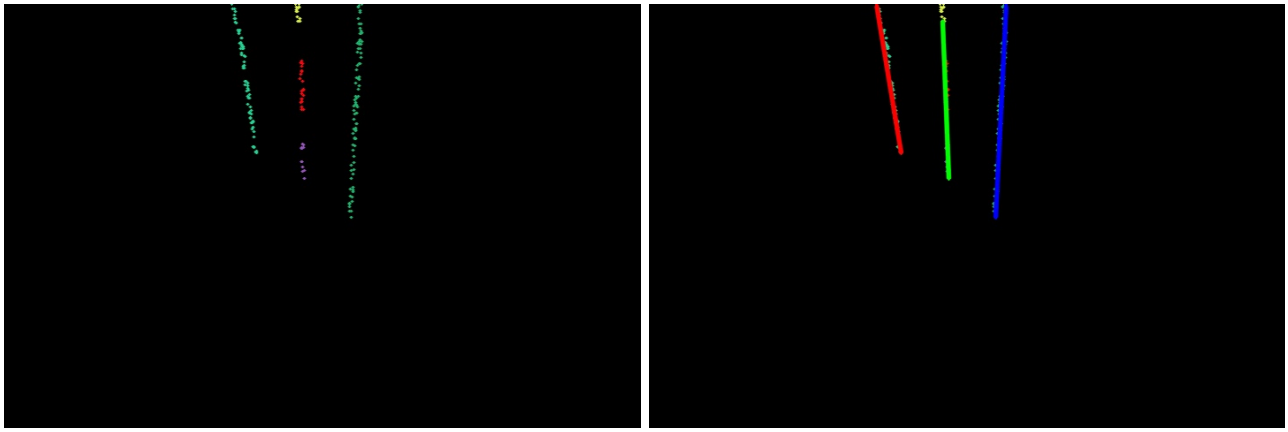

*Figure 3.1: Steps for defining input data for control algorithm. (left: clusters, right: 3 lines of input data)*

# 4 Discussion

This chapter gives a detailed discussion about the results showed in chapter 3 highlighting on lurking anomalies and providing possible solutions for these issues.

As for the bird's eye view, its transformation is the most computation demanding from all the image processing procedures and clustering algorithm. Therefore, it is of utmost importance to find an alternative solution for this transformation. To gain a more precise view of this issue, separate tests were performed for the warpPerspective OpenCV function and for the calculation of the transformation matrix M as well. Since, the latter possesses negligible run-time property compared to the former function's property, an alternative, less computational implementation is highly recommended for this warpPerspective function. To further emphasize the significance of this matter, it must be pointed out, as table 3.1 shows, that more than half of the computation is consumed only by this function, and this statement is hardware independent.

Furthermore, the lines detected by the Canny edge-detection algorithm, as figures 2.1 show, are distorted due to the bird's eye view transformation. This distortion can be explained by the order of the image processing steps. Namely, the Canny edge-detection precedes the bird's eye view transformation which is succeeded by the Hough line transformation. This sequence allows the bird's eye view transformation to distort the detected edges. However, the sequence of steps where the bird's eye view transformation and Canny edge-detection are switched would lead to a distortion free bird's eye view transformation. This modification is relevant for the detection of the two solid lines.

Lastly, the current implementation of pattern matching is somewhat limited, since it can only detect straight lines that are situated in a relatively vertical position. This limitation does not affect the processing of straight lines, since we can assume that the camera always points approximately the same direction as the lane goes. Although this strategy is not applicable in curve scenarios. For these cases a separate curve fitting algorithm should be applied on the clusters to detected the solid and dashed curved lines' angles and compute the optimal line.

# 5 Conclusion

This chapter summarizes the content of this report and gives a short conclusion about its main parts.

After a brief introduction on the topic of image processing, clustering and pattern matching, the report discussed the design and actual implementation of each of these topics showing the output images of all the steps of image processing on an arbitrary lane. This discussion was followed by a chapter where each image processing step's run-time property were shown with an example on the resulting clusters and the input data for the control algorithm. The final discussion was based on these results and pointed out several drawbacks with possible solution for them. This discussion mainly covered the topic of image processing and patter matching. Since the ideas and design presented in this report were just the first step in a continuous development process, it has its limitations and requires improvements.

# References


[MP12] M.Venkatesh and P.Vijayakumar. Transformation Technique (2012).

[Opea] OpenCV. URL: http://opencv.org.

[Opeb] OpenCV. *Geometric Image Transformations*. URL: http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html.

[Wik] Wikipedia. *DBSCAN*. URL: http://en.wikipedia.org/wiki/DBSCAN.