Finding the Convex Hull with Application to Distance Measurements for a Formula-Student Racecar

Benedikt Schlereth

July 20,2020

Contents

Abstract

To find cones on the track for autonomous racing a computer has to visualize them and has to know how far away they are. To detect the cones inside an image, the algorithm extracts colors from the image. To smooth out the edges of detected faces, the convex hull is calculated. Afterwards it looks for shapes that are describing the desired object. Knowing the dimension and proportions of the cone, a distance estimation can be done. Finally, a margin will be added to know how good the distance was estimated. All of this makes it possible to scan the track ahead and convert it to a 3D map.

1 Introduction

Formula-Student is an international design and construction competition where teams from all around the world are constructing and then compete with their single seat Formula race cars. With the new Formula- Student Driverless class, teams are encouraged to develop their own self driving cars. My motivation is to kick off the development of our own Driverless Vehicle by implementing the first steps by providing a way to detect the boundaries of a race track. The track markings are strictly defined in the competitions rules. It is important to know that the left border of the track is marked with small blue cones and the right one with small yellow cones. Start, finish and timekeeping lines are marked with big orange cones. It is also important to know, that the maximum distance between two cones in driving direction is 5m, in corners it can be expected to be less [2]. Knowing the rules is important to understand the track-layout and be able to make an estimation on how far ahead a self-driving-vehicle needs to detect the boundaries as well as how exactly each position of a cone must be known.

2 Steps to Extract the Features

To find the cones inside a picture one could create a dataset of 100+ cones and then train a neural network. But working with a neural network brings the difficulties of choosing a network that is efficient to run on a mobile platform (inference-cost) while detecting everything reliable (accuracy). Most of the algorithms are designed to detect different objects, which is not necessary for this application. Only three different colors need to be detected, the shape will always be the same. With knowing the features, a fast object detection can be realized and is used for this project. To visualize the process a scene with green cones and different lighting is chosen. The picture was taken from a height of $1,2m$ and the distance between the cones is $2m$ with the nearest at $2m$ and the farthest at $10m$ away from the camera. The resolution is 1920 by 1080 pixels. The resolution is chosen in a way that a CMOS camera can deliver 30 frames per second (fps).

Figure 1: example image

2.1 Color Filtering

The most obvious feature is the color. So the first step is to implement a color filter. For this example it is green but in the future a separate color filter for blue, yellow and orange can easily be implemented.

```
1 hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
2
3 # value for colorfilter
4 lower_green = np.array([30, 100, 0])
5 upper_green = np.array([90, 255, 255])
6
7 mask = cv2.inRange(hsv, lower_green, upper_green)
8 res = cv2.bitwise_and(hsv, hsv, mask=mask)
9
10 blur = cv2.GaussianBlur(res, (15, 15), 0)
11 return blur, mask
```

The image provided for the color filter needs to be converted from its RGB values to the HSV color space. The HSV does not describes the color with an eight-Bit value for each red, green and blue (RGB), but with a circle where the first value is for the color with 0 being red and 255 blue. The second value represents the saturation and the third describes the brightness of the color.

With knowing how the HSV color space works it is possible to find the right values which will describe the color of the cones. It will be necessary to try the colorfilter for images in bright sunlight as well as cloudy conditions the make sure to detect cones in every possible environment.

Next step is comparing every Pixel with our given value. With the OpenCV function "inRange" one image will be read and every pixel compared to the set boundaries. It returns a binary image which means every pixel that was in range of the colorspectrum will be saved as a one/white and everything else will be a zero or black.

2.2 Contours

Searching for Contours on the black and white image is a easy task and delivers the same results every time.

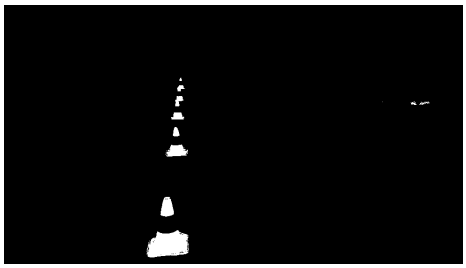

Figure 2: binary-image

But as seen in the top right hand corner of the image some noise will be left inside the picture. Depending on the scenery the colorfilter will always detect areas that are not of any interest. Therefore another filter is needed. A good difference between important and noisy areas, is the size of the contour. But you have to keep in mind, that deleting all small contours also deletes cones far away. Keeping the rules in mind, deleting the ones further away than 10m should be fine.

2.3 Hierarchy

The color-filter left some pixel inside of the designated areas out and will show up as contours. Luckily the contour-function gives us back the hierarchy of each contour. By filtering out the parts lying inside of bigger areas leaves possible cone-areas with less noise.

2.4 Convex-Hull

Now only a handful of areas should be left to analyze, so executing intense operations on these contours will not have a big impact on the runtime of the code. The next step is finding the convex hulls. A more precise description will be given in the section 3. The convex-hull leaves a very good approximation of the cones edges as seen in Figure 3.

Figure 3: convex-hull

2.5 Aspect Ratio

A nice feature of the cones is the round top part. So no matter at what angle or distance the convex hull is detected the ratio between the height and width

stays the same. With this feature extracted, now all top parts of cones inside the image should be detected. As seen in the Figure 4 it does not work perfectly as it does not track the cones partially covered. Again, knowing the Rules helps to estimate that this should be a rare occasion and two missed cones are better than a cone detected where there is none.

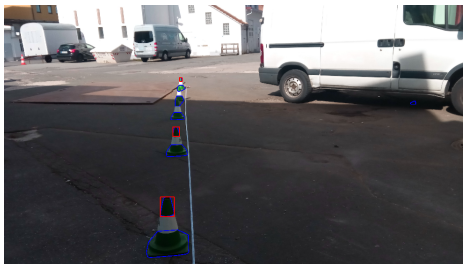

Figure 4: top-part

2.6 Bounding Box

With every top part of the cones found now the bottom parts need to be added. Therefore the distance to every other contour lying below and being bigger in size, is calculated. The nearest neighbor, while being not any further than the y-dimension of the top part away, is most certainly the fitting part. The maxima of both contours give the outlines of a bounding box which is drawn in red on Figure 5. This helps to visualize what the computer detects and is needed for the distance estimation.

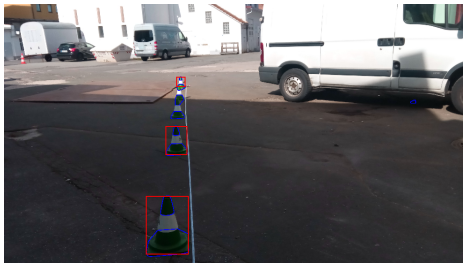

Figure 5: bounding-box

3 Convex-Hull Algorithm by Graham and Yao

The convex hull is a polygon with all points inside the boundaries while having no concave edges. Finding the Conex Hull is a well known problem with different solutions. OpenCV uses a gift-wrapping-algorithm introduced by Sklansky in 1982 [3] which is proved to be wrong on some special cases[4]. To always find the right convex hull in a reasonable time an algorithm by Graham and Yao looks promising. Sadly the time for this project run out, so there were no resources left to implement the algorithm.

4 Distance Estimation

To find out where the cones are on the track the distance to each cone must be calculated. With only one camera the number of pixel and a known height for a cone is the only value that can be used. It is also important to know the resolution of the used camera. While it is quite simple to take high-resolution pictures and then wait for a computer to analyze them, it is not so easy in a real-time application on mobile Hardware. To make sure the picture can be read and analyzed the input will be limited to an 1920x1080 px image which can be processed by a small computer-module like a Jetson-Nano with a frame-rate of about 30 fps.

4.1 Height

One possible feature is measuring the height by counting the pixel describing the y-Height(h) of the detected cone. The bounding box function of OpenCV already gives back the number of pixels in the y-Direction.

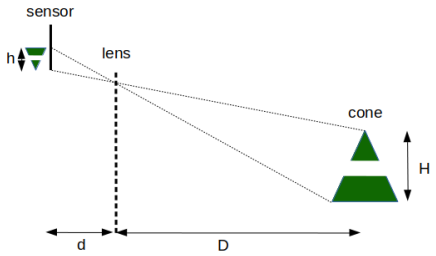

Figure 6: distance-estimation

The thin lens equation is given as

$$\frac{D}{H} = \frac{d}{h}$$

where d is an unknown variable depending on the manufacturer of the camera. But d can then be calculated with $d = \frac{D_1 \cdot h_1}{H}$ and then calibrated by taking a picture of a cone at a known distance. After evaluating the dimensions a good value is $D_1 \cdot h_1 = 440$mpx. So to calculate the distance to each cone the following equation can be used:

$$D = \frac{440}{h}$$

4.2 Width

Similar to the y-Dimension the bounding-Box-function gives back the x-Length. The math and algorithm for this distance estimation would be the same as for the height as it is for the error and possible disruption. Therefore it will not be discussed any further to make a clearer presentation.

4.3 Side-Length

The top part of a cone is round and so no matter at which angle the camera is detecting the cone, the proportions are the same. Being robust against rotation it is possibly a better feature for evaluating the distance. By using the side-length s as described in Figure 7 for the distance estimation it can be valid for all angles of the cone.

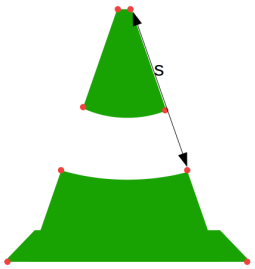

Figure 7: corners of a cone

To measure the side-length the corner points need to be found first. Inside of Figure 7 the important points are marked with red-dots. For all cases it can be estimated that the cone will have a flat point on the top and therefore both parts of the cone are trapezium-shaped.

Input: convex-hull
Output: extrem-points
find bounding Box;
get top-line;
get bottom-line;
while *point in convex hull* **do**
  if $y_{point} = y_{top-line}$ **then**
    if $x_{point} \leq x_{left-top-corner}$ **then**
      left-top-corner = point;
    if $x_{point} \geq x_{right-top-corner}$ **then**
      right-top-corner = point;
  if $y_{point} = y_{bottom-line}$ **then**
    if $x_{point} \leq x_{left-bottom-corner}$ **then**
      left-bottom-corner = point;
    if $x_{point} \geq x_{right-bottom-corner}$ **then**
      right-bottom-corner = point;
end

Algorithm 1: extreme-points

By drawing a bounding box around the contour the y-Dimension of the top and bottom corner-points can be estimated. Next step is to go through the points of the convex hull. If it lies inside a margin of the top or bottom line of the bounding boxes y-coordinates it is maybe a corner point. So the next step is then to look at the x-coordinates and compare if it lies more to the left or to the right then an already scanned point. That way the outermost points will be detected.

The algorithm 1 returns good corner points for cones standing upright. This code has to be run for the top section as well as the bottom section of the cone to be able to find all 8 corners.

By then calculating the distance between the Top-Right-corner of the top part to the Top-Right-corner of the bottom part the side-length of the cone can be calculated and used for a distance estimation similar to the one used in section 4.1 with the calibrated value for $D_1 \cdot h_1 = 280\text{mpx}$.

$$D = \frac{280}{s}$$

4.4 Error

With to much noise inside the image and low resolution the distance can not be accurate within a centimeter. But it is important to know how exactly each cone can be detected to know in which area a cone probably stands for the car to know how fast it can travel.

4.4.1 Rotation

One error can occur due to the angle the cone is facing the car. An example picture (Figure 8) was taken to evaluate the error by rotating one cone by 45°. The measurements for the cone $2m$ apart show a small difference with

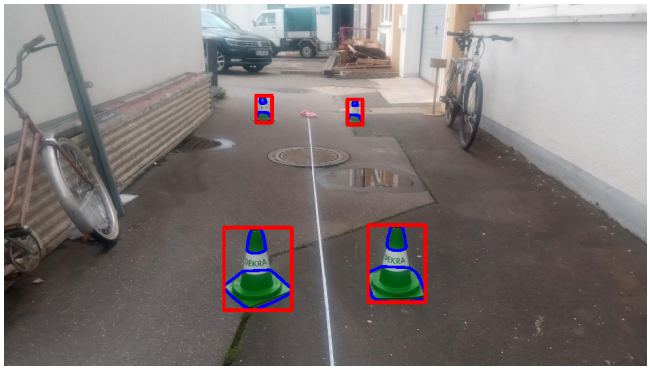

Figure 8: cone 45° rotated

the estimation on the left side measuring $s_{45°} = 2,22m$ and $s_{0°} = 2,23m$ on the right side. So an overall difference between both measurements of just $\Delta_s = 0,01m$. The overall-height estimation gives back $s_{45°} = 1,79m$ and $s_{0°} = 1,90m$ with a delta of $\Delta_h = 0,11m$. For the cone $6m$ apart the error because of noise is already more important than the error occurred due to the rotation. Nevertheless the relative distance for the side-length with $\Delta_s = 0,30m$ is still a little bit better than for the height-measurement $\Delta_h = 0,36m$.

4.4.2 Pixel-Error

To make sure the track can be monitored correctly it is important to know how good the distance estimation will be. By using only the number of pixels for an estimation you need to know how accurate the size can be estimated and how

big the difference is for one pixel. The error for the different detected distances is shown in Figure 9.

$$\Delta D = \frac{\partial D}{\partial h} \cdot \Delta h = \frac{D_1 \cdot h_1}{h^2} \cdot \Delta h$$

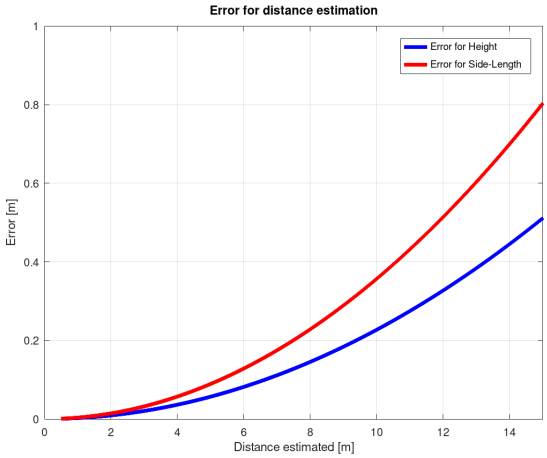

Figure 9: error for one pixel

4.5 Disruption

The most discouraging dial to get right will be the color-filter. It has to accurately detect the right color inside the image but it must not detect any yellow banners or parts of the blue sky as cones, while working in bright sunlight as well as pouring rain.

An often seen "DNF" (Did not Finish) in the driverless class is, when a cone is tipped over. In the next round, the car detects the cone as part of the track-boundary and stops due to a imperfect course. So hopefully by detecting only the upstanding cones the ones not marking the true edge of the track wont disturb an path-finding algorithm.

5 Tests

An important test will be to what happens to a cone that is tipped over and lies somewhere on the track. Either it should not be detected at all or mark it as tipped over to ignore it for calculating the path. As seen in figure10 the computer visualize a green area but does not detect it as a valid cone.

Further tests can be done by using a computer generated image like done for Figure 7 and introducing possible disruption. They can be similar to the points discussed in section 4.5 or be a marks (e.g. Dirt) on the cone in various sizes and forms.

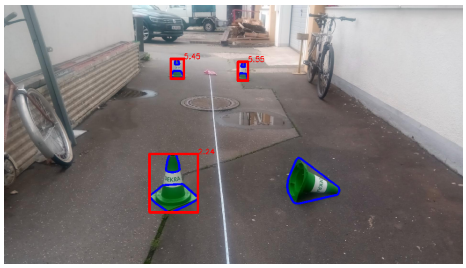

Figure 10: tipped cone

6 Discussion

The best angle of the camera sill needs to be evaluated. For Testing purposes it was neglected but with cones coming closer to the car than $2m$ this can make a noticeable difference.

Visualizing the cones for a computer algorithm works really good. Further tests to evaluate the robustness are needed, but with the distance estimation the next step towards creating a Map of the track can be taken. This is an important preprocess for a path-finding algorithms and to give the car the ability to know where on the track it is located. It will be an exiting future with challenging and cool new projects for our Formula Student team.

References

[1] Finding the Convex Hull of a Simple Polygon, Ronald L. Graham, F.Frances Yao, 1983

[2] FSG Competition Handbook 2020

[3] OpenCV-docs(3.4.9).[Online] available on: https://docs.opencv.org/3.4.9/index.html. accessed on 08. July 2020

[4] G. Aloupis, A History of Linear-time Convex Hull Alorithms for Simple Polygons.[Online] available on: http://cgm.cs.mcgill.ca/ athens/cs601/. accessed on: 08. July 2020