Project 1 Report: Autonomous Lane Tracking

1. Lane Recognition

1.1. Design

In this lane recognition part, the goal is to recognize the lane in the picture obtained by camera. Then use the data for the lane tracking control part. First, we would get pictures from the camera, then extract lanes from pictures, next do the Hough transformation and finally fit the lines and select reference points used for the controller.

In the image acquisition part, we use ‘snapshot’ in MATLAB to obtain pictures from a webcam camera. We use ‘size’ function to get the size of the image, which is used for mask and coordinate transformation later. Here, we created an endless loop for obtaining the pictures continuously. In our early test, we used a higher resolution in order to get the best images from the camera, but we found that a higher resolution would cause a long process time. In other words, a high resolution of image would make the code run slowly and can’t control the vehicle precisely. Thus, we choose the resolution of ‘320x240’ at last.

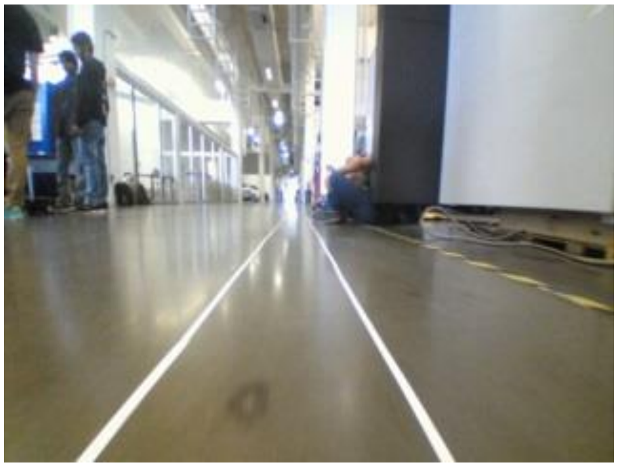


Figure 1 Image acquisition

In the feature extraction part, the goal is to extract steady left and right markers from the images. First, we used ‘rgb2gray’ in MATLAB to convert the image to grayscale. Then we implemented an edge detection function using ‘canny’ algorithm. And we adjusted the thresholds based on our test on the sample videos. However, in our real test on the track, we found that there are some narrow cracks along with the track and they were sometimes detected as the lanes. Because the track is white, we used a color selection part before the edge detection part in order to filter those non-white lines which were detected as lanes by mistake. We chose 210 as the threshold, which is based on the real test on the track. This process would not increase the total process time much and the result was acceptable.

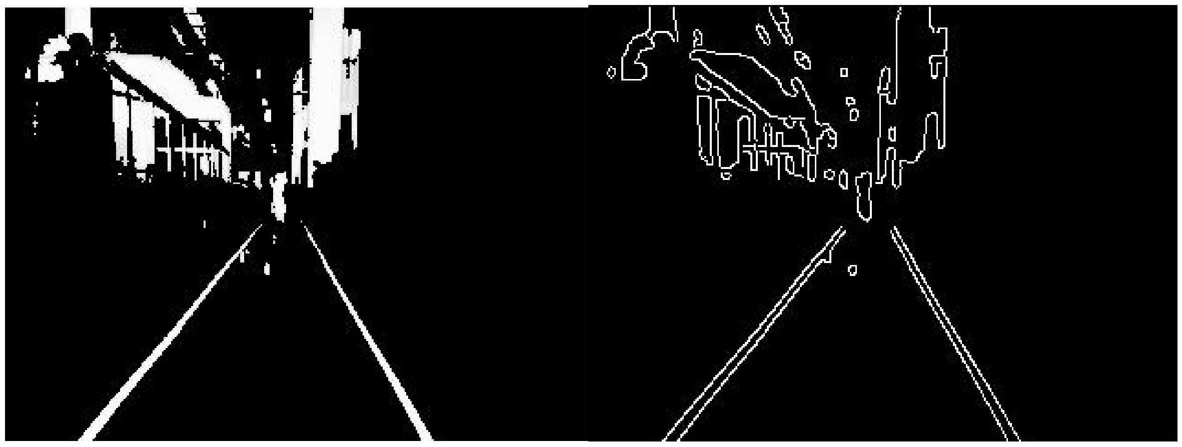


Figure 2 Color selection (left) and edge detection (right)

Next, we could see from the Figure 2 that there are still some other edges in the image besides the lane lines. We need to mask everything out except the lanes. At first, we considered to do the mask in both x- direction and y-direction, but we found that sometimes the lanes would show at the margin of the images in the curve due to our position of the camera. Thus, we just did the mask in y-direction.

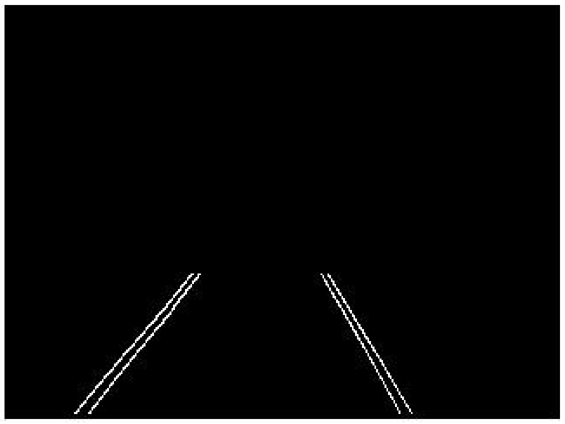


Figure 3 Region masking

Then we converted image space to Hough space using Hough transformation. We used ‘houghpeaks’ to pick the top 3 lines that have the highest number of points and ‘houghlines’ with ‘FillGap’ and ‘MinLength’ to output the lines. If there is a higher value of houghpeaks, there will be more houghlines but the latter fitted results would be worse due to the aliasing of the left and right lines. Thus, we set it at 3. The track is also not very complete at some curves, so we adjusted the FillGap and MinLength in order to be able to obtain houghlines in almost all the conditions.

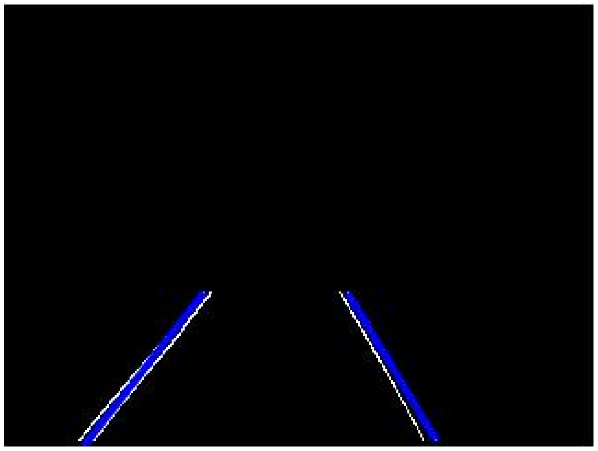


Figure 3 Hough transformation

Finally, we should do the post processing to get the points used for the controller. In this part, we separated the left lines from right lines based on the rho and theta of the houghlines. We set thresholds for rho and theta, which made the close lines into one same line segment. We set 70 as threshold for rho and 10 for theta. Noticed here that we just separated left lines from right lines but we didn’t define whether it’s a left lane or right lane. In other words, we just divided all the houghlines in the image into two line segments but we don’t tell you which segment is the left or right. Then fit the two line segments into just two single lines.

The thought is that we need to make the code more efficiently and laconic. At start, we did the discrimination of the left lines and right lines and we created a lot of conditions and made the codes verbose. Because sometimes one of the two lanes can’t be detected rightly, we need also consider that situation and create corresponding logic to define the left lines and right lines. However, we came out a simpler idea. We found that we just need the central reference point of the track for the pure-pursuit controller rather than the left and right points, which means we just input the coordinates of the central point rather than those of left points or right points. In other words, we could get the central coordinates as long as there are two points but we don’t need to define which is left or which is right. Then the goal is obtaining two points all the time. As the image frame is continuous, when there is one lane missing due to the poor detection, we could just abandon all the results in this frame and use the results obtained in the last frame (last loop). Thus, we could get the two points all the time and this would work because there are only few frames that we couldn’t detect one lane and there are no frames that we couldn’t get both lanes. This simple idea doesn’t need to create conditions to decide how the vehicle should steer when one lane is missing because there is always a right input for the controller. Another former idea is also using the results obtained in the last frame when one lane is missing. In that idea, we also need to discriminate the left lane from right lane and judge which lane is missing then use the results of that lane in the last frame. We spent lots time creating the logic of how to judge which lane is missing until we realized that we didn’t even need to do that. Just abandon all the results for both lanes and use the lanes in the last frame no matter whether the missing lane is left or not.

Here, we also use a simple time domain filter for the slope of fitted lines to increase the stability of the results.

1.2. Results

The final result for this part is one central reference point for the track in the pixel frame and the vehicle central line showed in the image. The code works fine through the test.

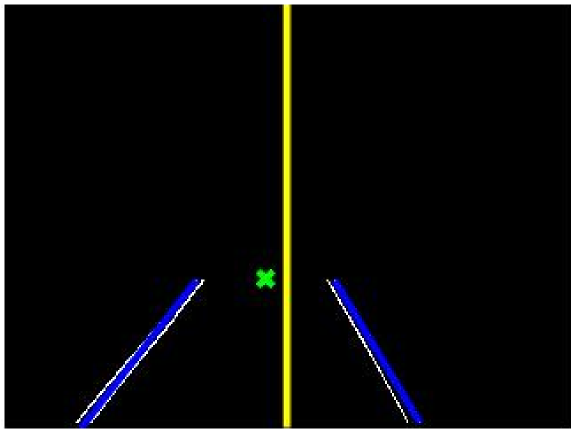


Figure 4 Final results for lane detection

2. Camera Calibration

2.1. Design

The coordinates in pixel frame can't be used in the pure-pursuit controller directly, so we need to do the calibration to build the correlation between what is measured by camera and what we want in the controller.

In the camera calibration part, our goal is to transfer coordinates from image frame to vehicle frame. For the intrinsic parameters, we used Computer Vision System Toolbox in MATLAB to do the calibration and obtain them. Because we choose '320x240' as our resolution, the calibration results would differ according to the change of the resolution.

For the extrinsic parameters, we calculated it based on our measurement and the selection of the origin.

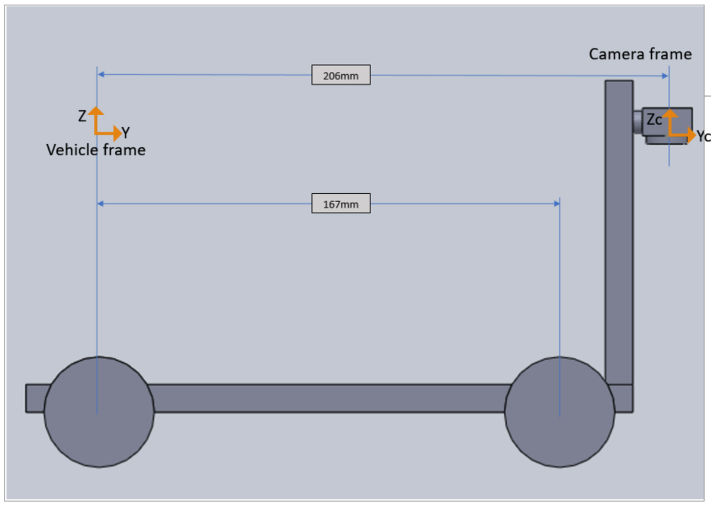


Figure 5 Extrinsic parameters calculation

2.2. Results

For the intrinsic parameters, the results are:

$$\begin{bmatrix} 86.2872 & 0 & 0 \\ 0 & 86.4089 & 0 \\ 76.1311 & 62.2382 & 1 \end{bmatrix}$$

For the extrinsic parameters, as the orientation of the coordinates are same, the rotational matrix reduces to identity matrix. And the results are:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} + \begin{bmatrix} 0 \\ 206 \\ 0 \end{bmatrix}$$

3. Lane Tracking Control

3.1. Design

In lane tracking control, we choose pure-pursuit controller because we don't need to tune the parameters as the controller introduced in the slides. We just need to build a correlation between vehicle actual steering angle and steering control signal. This could be linear and easy to fit using a linear equation.

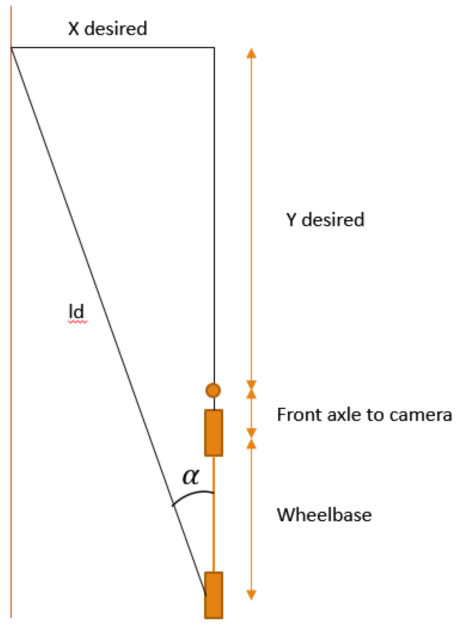


Figure 6 Pure-pursuit controller

$$\alpha = \tan^{-1}\left(\frac{X\ desired}{rear\ axle\ to\ camera + Y\ desired}\right)$$

$$l_d = \frac{X\ desired}{\sin\alpha}$$

$$\delta = \tan^{-1}\left(\frac{2L\sin(\alpha)}{l_d}\right)$$

After these steps, we could get the actual steering angle φ. Then we measured the maximum actual steering angle and its control signal and minimum actual steering angle and its control signal. We use a linear equation to build the correlation.

For left, it is:

$$\varphi = -0.635 * \delta + 0.33$$

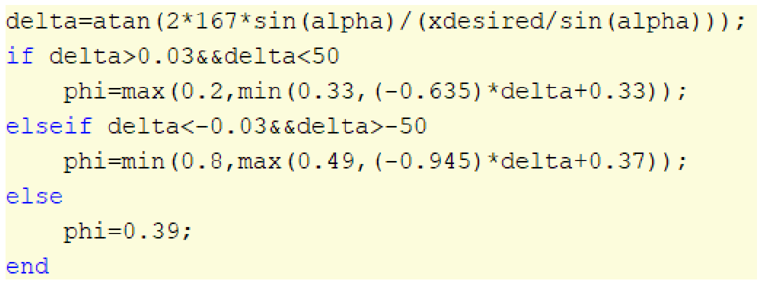
For right, it is:

$$\varphi = -0.945 * \delta + 0.37$$

During the real test on the track, we found that sometimes the vehicle would mess up tracking the straight line but tracked the curve lane well. This means the equation of the controller is right but the controller is too sensitive to the input δ. Thus, we set a dead zone for the neutral point. When the steering input δ is less than 0.03 or greater than -0.03, we make the vehicle go straight. Also, we limit the output of the controller to avoid dramatic control signal to make the vehicle run smoothly.

3.2. Results

The result of this controller is acceptable during the whole test.



```
delta=atan(2*167*sin(alpha)/(xdesired/sin(alpha)));
if delta>0.03&&delta<50
    phi=max(0.2,min(0.33,(-0.635)*delta+0.33));
elseif delta<-0.03&&delta>-50
    phi=min(0.8,max(0.49,(-0.945)*delta+0.37));
else
    phi=0.39;
end
```

Figure 7 Pure-pursuit implementation in MATLAB

4. Conclusion and Discussion

In project 1, we could track the lane well finally but we met some problems in the design process. For the lane recognition, the resolution would affect the process time much so it's important to reduce the resolution. The result of houghlines is quite sensitive to the value of houghpeaks. Neither higher or lower is good and we should find a tradeoff between precise detection and aliasing of different line segments. Abandoning discrimination of left lane and right lane is a good idea and this really make the code laconic. For the extrinsic parameters, we really need to tune it based on the actual test. Sometimes the calculation result of extrinsic would not be precise but the control quite need a precise input, thus some adjustments on extrinsic parameters is important. The dead zone is really convenient and useful in the controller design. Because the whole code is based on an endless loop, we need to notice the overlap of the variable values in two contiguous loops. The solution is clearing all the unnecessary variables at the end of every loop, or it will cause error in the next loop. The whole process time is also worth to be considered in the code writing.