Robotics Practical 1 Assignment 2

Laura M Quirós (s4776380)
Adriana A. Haralambieva (s4286103)

February 25, 2024

Include an original image from the camera from the robot starting position and a second image that is the output of the *warp_perspective* function.

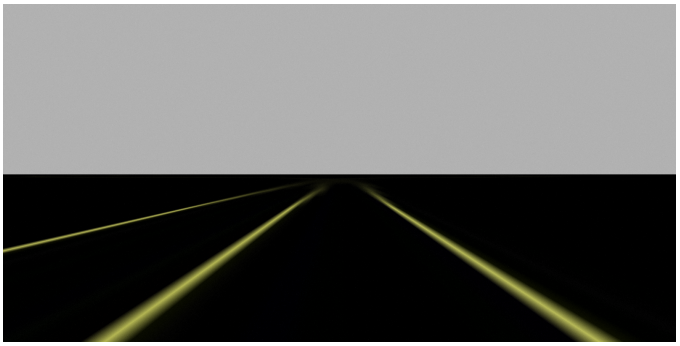


Figure 1: Original camera from the robot starting position

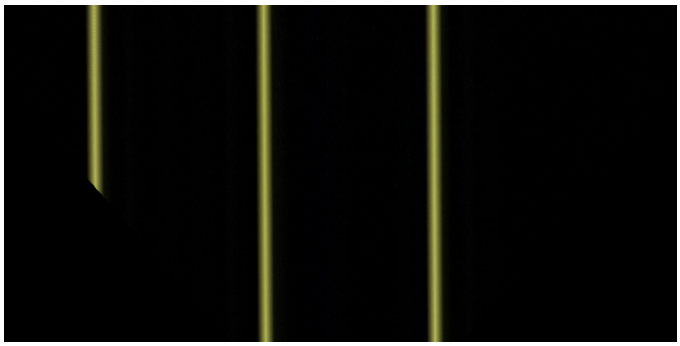


Figure 2: Warp perspective image from the starting position

Include the output of *filter_hsv* with the warped image from the robot starting position as input. Also write down the parameter values you hard coded.

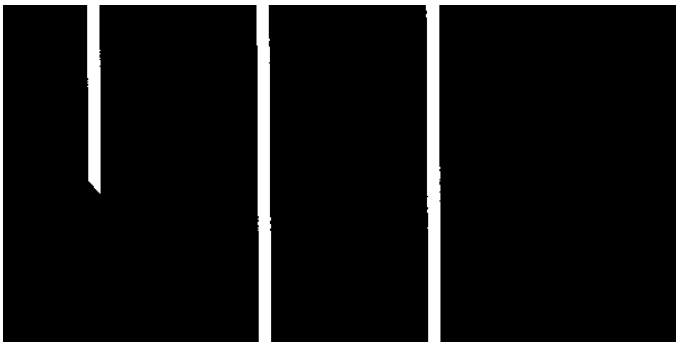
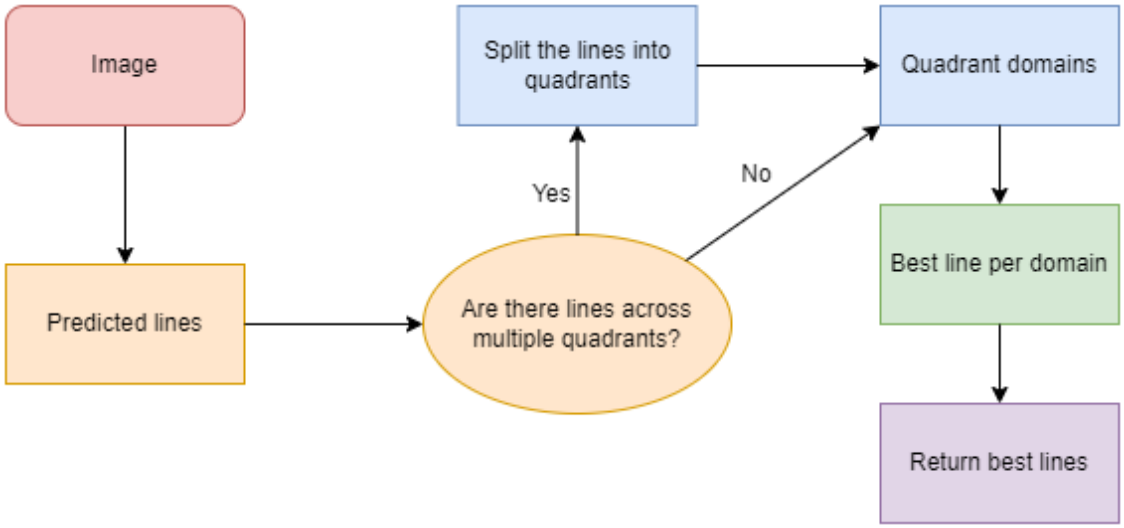


Figure 3: Filtered by HSV image

Channel	HSV Lower	HSV Upper
H	4	44
S	61	194
V	55	191

Include a description of the line-equation-finding method you chose to implement, with a (visual) description of the pipeline. As with any academic reporting, write this part in such a way that a student would be able to recreate the code from your description.



As seen in the flowchart, the main processes happening are:

1. predict lines: We use the function `pHoughLines()` which takes as arguments the filtered HSV image, the rho and theta resolution by which the image clusters are defined, and lastly the vote threshold (only if there are enough votes for a given line, it will be claimed as a straight line). The function will then return many predicted lines.
2. quadrants: The next step is to split all those lines into 4 different quadrants. We set the quadrants in relation to the x coordinates given by the shape of the cropped figure. The first step is to check whether some of the predicted lines are in multiple quadrants. If that is the case, we split this line into as many lines as quadrants it goes through. For example, if the line is in quadrants left and far left, we split it into two lines - one in the left and one in the far left quadrants. When there are no more lines in multiple quadrants, we put each line to the respective quadrant it belongs to, depending on its coordinates.
3. best line: Now we end up with many lines in each quadrant. However, we want to have only 1 per quadrant. That is why for each line we check the y-coordinates of its two ends. We search for the furthest top and bottom end of the lines (We treat each end as a separate dot. This means that we separately search for the two best ends, and not the line that is the best. In this case we can end up with two ends of two different lines). After finding the best ends, we connect them to make the best line. We do this for each quadrant.
4. return: For each quadrant, we return 1 best line (or none).

Write down the found line equations from the image of the previous part.

For each quadrant, we have example coordinates (in the format: $[x_1, y_1, x_2, y_2]$). For quadrants where no lines were found, we returned an empty []. Equations: best far left: [78, 0, 86, 167]; best left: [231, 0, 236, 299]; best right: [384, 0, 389, 299]; best far right: []

Additionally, include a figure that overlays the found line equations on top of the output of the *warp_perspective* function.

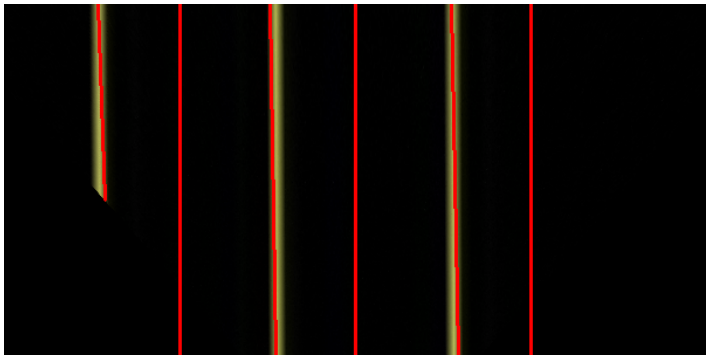


Figure 4: Figure overlaying the found lines in top of Figure 2