Computer Vision Assignment- Software, Systems and Applications

mdsw22

School of Engineering and Computer Science,

Durham University,

Durham, UK

christian.johnston@durham.ac.uk

To fix the issue of varying illumination conditions, I implemented openCV’s Contrast Limited Adaptive Histogram Equalization. I utilised the LAB colour-space (i.e. a colour-space with a luminance channel), then applied adaptive histogram equalisation to the L channel before merging the channels and converting back to RGB.



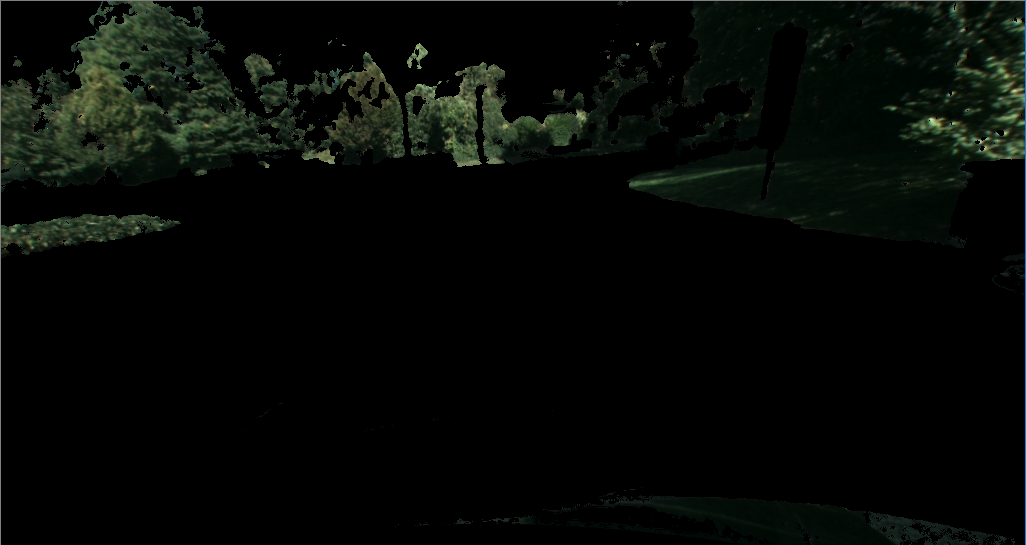


To reduce the search space of points, I filtered pixels by colour; removing green pixels for the trees. Filtering specific colours is easier in the HSV colour-space due to the isolation of the hue channel; selecting green pixels, regardless of whether they are dark or light.

This utilises the morphological operation *opening* which combines erosion followed by dilation.

I utilised Gaussian Blur so neighbouring pixels become more uniform in color, easing brighter and darker spots on the image and keep holes out of the mask.

Image below shows the function picking out the green pixels effectively.



To reduce the search space of points further, I cropped the points at the edge of the image as often the road in front of the car is in the centre of the screen.

I picked lower quartile as 0.2\*width/height and upper quartile as 0.8\*width/height, making pixels outside this range black.

These are examples of region of interest extraction techniques and also heuristics to speed up processing times.

4.

In an attempt to reduce the set of points I utilised Canny edge detector. To get the threshold values I blurred the image and then equalised the histogram of this blurred image. Then I took the mean of this equalised image and then set the lower threshold to 0.66\*[mean value] and the high threshold to 1.33\*[mean value]. This a heuristic to speed up processing times and also improve performance. I ended up not using this method as the disparity image is smooth and didn’t give the best edges.

7.

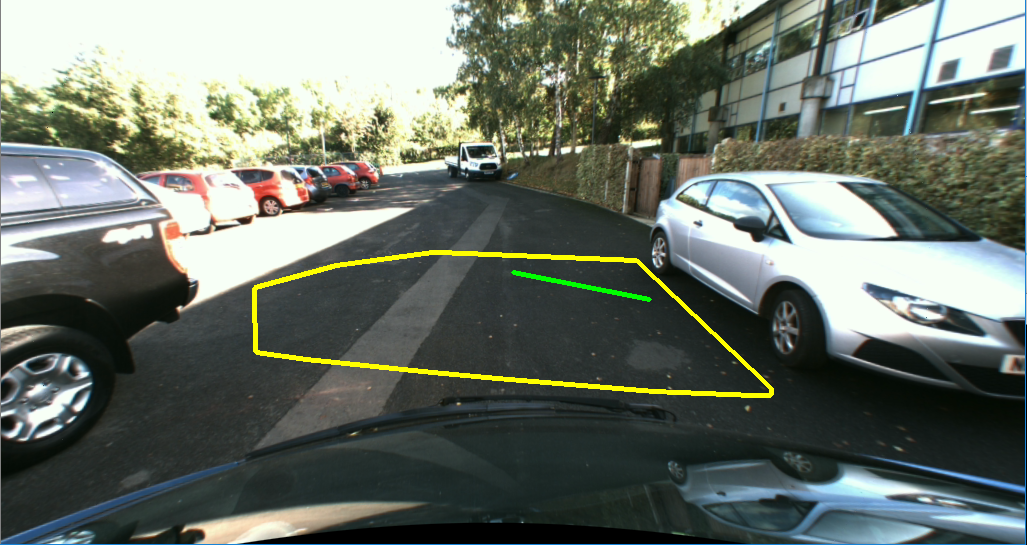
I implemented RANSAC using a very small threshold value (~ 0.01) to determine when a set of points fits the model. The number of iterations of canny dramatically affected performance and efficiency. When I used canny edge detection, there were much less points for it to choose from so the number of iterations made little difference. In my implementation not using canny, the optimal balance between speed and performance was ~40 iterations which displays every image within 5s.

8.

After projecting my 3D points generated via RANSAC to list of 2D points. I created a new image, mapping all these points to the new image and drawing circles around them. I then performed Gaussian blurring, thresholding (which I get very low) and dilation before contouring the image. This helped to get the best image possible before contouring the image. I then sorted the contours by area, selecting the biggest 50. I then drew a convex hull around these contours.

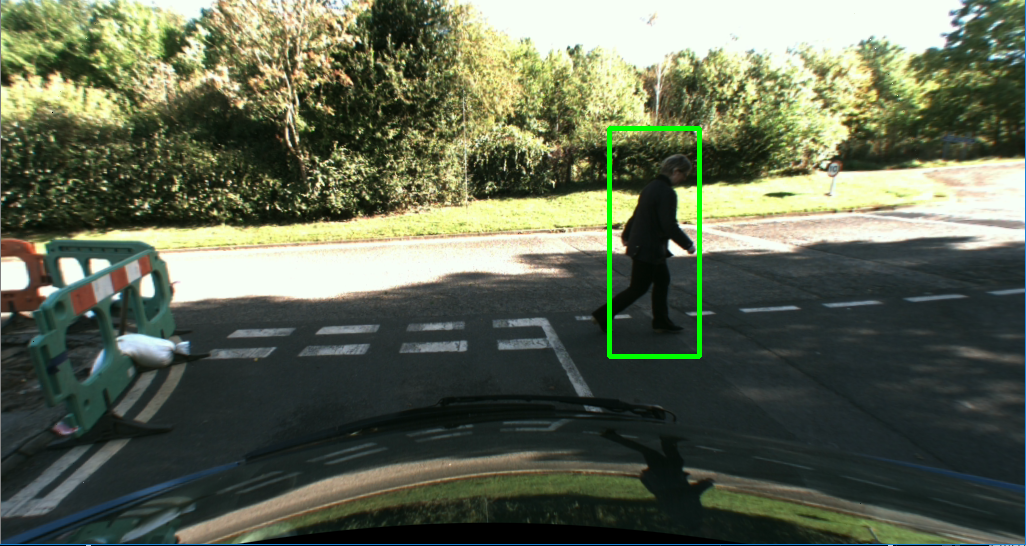
9.

To plot the planar normal direction vector on the image I took the centre of the image and added to this the normal direction vector of the plane to get a 2nd point. I then drew a line between these points. Shown in image below.

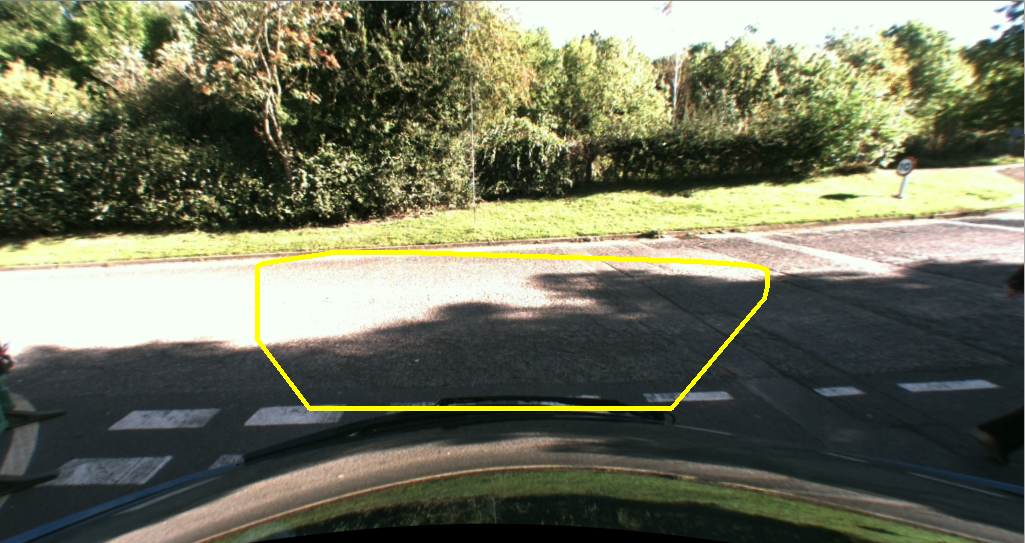


10.

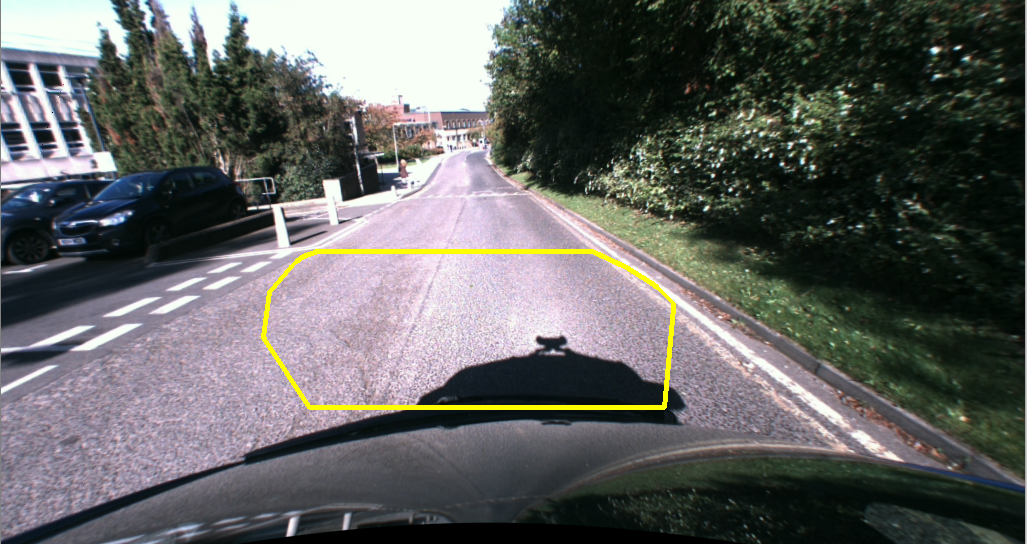
To automatically detect and highlight obstacles rising above the road plane, I implemented HOG pedestrian detection. This detects people walking across the road very well but is sensitive so picks up some objects which it shouldn’t.



**Images illustrating performance**



Images to left shows the edge of the road being identified well.



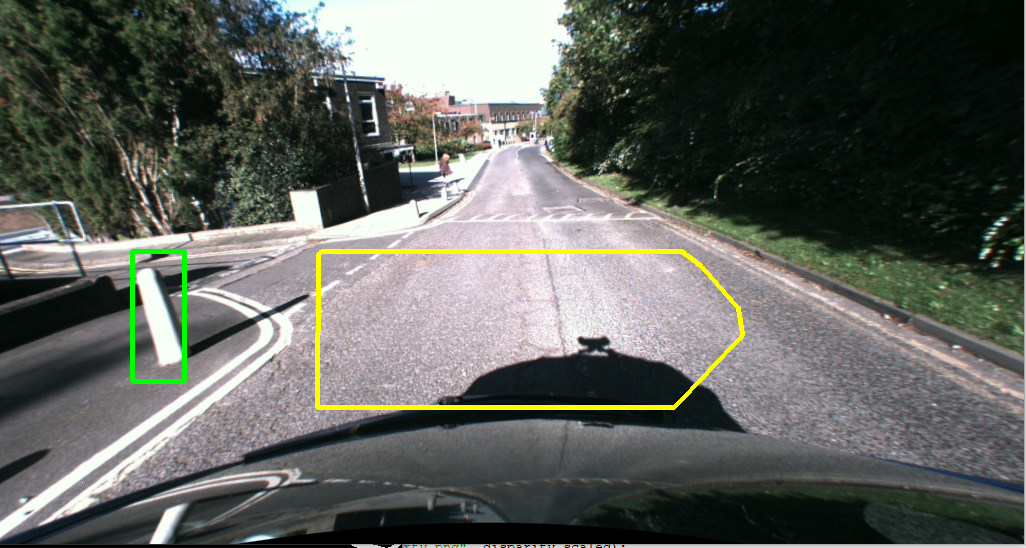


Image to left shows the plane being identified and also HOG identifying a bollard.

**Statistical evidence of success of system**

I am taking as a sample the first 30 frames of the image to evaluate for performance:

* If the system detects a plane and thus is able to plot this plane and the normal direction vector.
* If the plane detected lies only on the road and thus is able to identify the edge of the road
* If a pedestrian rising above the plane is identified by the system.

**Full Table on Next Page, Analysis of Results**

Out of a sample of **30** images:

My system identifies a plane and normal vector 100% of the time.

The plane identified lies completely on the road in front of the vehicle 73% of the time where the 27% it incorrectly determines a pavement, grass or other to be the road.

The HOG pedestrian detection identifies every pedestrian that crosses the road and also identifies evert bollards by the road however 17% of the time it incorrectly detects an object to be an obstacle.

In terms of speed of processing, every image was analysed and the plane was plotted in under 5s.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frame Number | Plane and normal identified | Plane lies fully on road | Pedestrian/obstacle identified | False identification of pedestrian |
| 1 | Y | N | N/A | Y |
| 2 | Y | Y | N/A | N/A |
| 3 | Y | Y | N/A | Y |
| 4 | Y | Y | N/A | N/A |
| 5 | Y | Y | Y | Y |
| 6 | Y | Y | N/A | N/A |
| 7 | Y | Y | N/A | N/A |
| 8 | Y | Y | N/A | N/A |
| 9 | Y | Y | N/A | N/A |
| 10 | Y | Y | N/A | N/A |
| 11 | Y | Y | Y | N/A |
| 12 | Y | N | Y | N/A |
| 13 | Y | Y | N/A | N/A |
| 14 | Y | N | N/A | N/A |
| 15 | Y | Y | Y | N/A |
| 16 | Y | Y | N/A | N/A |
| 17 | Y | Y | Y | N/A |
| 18 | Y | Y | N/A | N/A |
| 19 | Y | Y | N/A | N/A |
| 20 | Y | N | N/A | N/A |
| 21 | Y | N | N/A | N/A |
| 22 | Y | N | N/A | N/A |
| 23 | Y | Y | N/A | N/A |
| 24 | Y | Y | N/A | N/A |
| 25 | Y | Y | Y | N/A |
| 26 | Y | Y | Y | N/A |
| 27 | Y | Y | N/A | Y |
| 28 | Y | N | N/A | N/A |
| 29 | Y | N | N/A | Y |
| 30 | Y | Y | N/A | N/A |