

# Pedestrian detection and recognition around Durham

## Abstract

In this study we attempted to derive an optimal object detection and recognition system for pedestrians in general based on the HOG model. We found various ways to improve performance and efficiency of this method, however we did not manage to derive a good enough solution which filters out FP detections but leaves TP ones intact.

## 1 Introduction

Today, modern technology goes in the direction of automatisisation by using machine learning techniques and AI. Self driving cars are one of the leading fields of this area. After the machine learning "about" a certain type of object, our task is to successfully detect, recognise, and measure its distance to the viewer, whether it is a pedestrian for a car, or face for a phone. In this study, the focus is on pedestrians in the view of a car.

## 2 Our approach

Our approach is based on the HOG. After unsuccessful attempts to improvement the training method, we focused on implementing an optimal detector. First of all, we chose a sensible ROI, that is, on our detection region we exclude the bottom half of the given dataset due to the presence of the car, also we cropped the top half a little bit, because pedestrians are unlikely to be that high up, and if so, they would be too small for the svm to detect. Then by applying a gamma filter and by equalising the histogram of grayscale and RGB images, we managed to do some positive image preprocessing which improved the detection later on. After, our selective search algorithm gave possible regions of objects. We restricted our interest to those which have similar shape ratio as the training datasets:  $\text{height/width} = 2/1$ . After running these regions through the trained classifier, we recorded those regions which produces positive outcome, however, these regions were still relatively large. Thus for each of these rectangles, we applied the sliding window search algorithm to further restrict the area of possible pedestrians: we ran the sliding and scaling window through the svm classifier again, successfully reducing the number of detections, and recording the *new detections*. Still, these still included a good number of FP detections, hence we examined the disparity of the given detections. For each *new detection*, we only inspected the top 2/3 as the disparity values are more likely to differ between the foreground and background on this region. Our aim was to compare the shape of the object to sample picture[Figure 1]. To do this, we needed to get a binary image of the detection by applying threshold. For the threshold value we chose the *most common grayscale* value of the center of the region: we split the region into 16 rectangles and took the middle 4 to calculate our threshold. For the remaining part (outskirt), we decreased the grayscale of each pixel to the *most common grayscale* for those which were higher than this, hence we got an overall region where the highest grayscale value must be in the center. Then we applied the binary threshold to compare the outcome image to the sample one. To do this, we applied pixel by pixel comparison, and examined to what ratio they are the same. We got concluded that a sensible number to let a detection process further is around 0.65. Next we calculated the distance to the given object detections from the disparity map, and we further filtered these by omitting detections which are further than 50 meters. Our last filtering approach was to find those detections where the disparity map was clearly bimodal and not a uniform transition from a lighter to a darker colour, by creating a transition image between the highest and lowest detected grayscale value in the detected region, and compare the resulting image to the detected region, and if they were similar to a certain extent, then conclude it is an FP detection. Ranging for the found object then was computed from the disparity map using the disparity of the *most common grayscale*

### 3 Further improvements

- The uniform transitions filter successfully found several FP detections however it also filtered out a few TP ones as well. This could be improved by setting the parameters more carefully.
- Our method includes a lot of parameter settings. We found an optimal setting, however finding better ones could improve performance.
- Further improvements: investigating the svm score, increasing the rectangle sizes of the selective search detections, adjusting rectangles appropriately around the center of white patches after threshold and thus check for similarity, and scan disparity in the first place for objects.



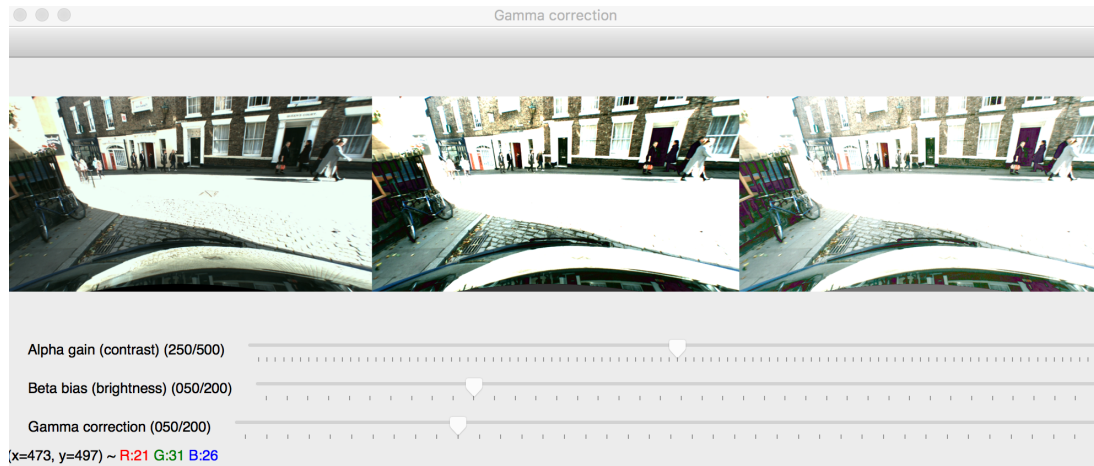
**Figure 1:** Human shaped sample for thresh check.

```

Computing HOG descriptors for hogs/svm_hog_maxit_500.xml
Successfully trained SVM with 27.67% testing set error
-- meaning the SVM got 72.33% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_maxit_700.xml
Successfully trained SVM with 29.55% testing set error
-- meaning the SVM got 70.45% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_maxit_900.xml
Successfully trained SVM with 31.44% testing set error
-- meaning the SVM got 68.56% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_neg_5.xml
Successfully trained SVM with 28.88% testing set error
-- meaning the SVM got 71.12% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_neg_10.xml
Successfully trained SVM with 30.63% testing set error
-- meaning the SVM got 69.37% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_neg_20.xml
Successfully trained SVM with 32.39% testing set error
-- meaning the SVM got 67.61% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_offset_2.xml
Successfully trained SVM with 33.2% testing set error
-- meaning the SVM got 66.8% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_offset_3.xml
Successfully trained SVM with 26.59% testing set error
-- meaning the SVM got 73.41% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_offset_4.xml
Successfully trained SVM with 29.28% testing set error
-- meaning the SVM got 70.72% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_pos_3.xml
Successfully trained SVM with 29.01% testing set error
-- meaning the SVM got 70.99% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_pos_5.xml
Successfully trained SVM with 30.5% testing set error
-- meaning the SVM got 69.5% of the testing examples correct!
Computing HOG descriptors for hogs/svm_hog_pos_10.xml
Successfully trained SVM with 32.66% testing set error
-- meaning the SVM got 67.34% of the testing examples correct!
Computing HOG descriptors for svm_hog_optimal.xml
Successfully trained SVM with 31.85% testing set error
-- meaning the SVM got 68.15% of the testing examples correct!

```

**Figure 2:** Results of trainings for various parameters.



**Figure 3:** Finding the best preprocessing filter.



**Figure 4:** Performance of uniform transition filter on the detected regions. Top half shows the actual detected pedestrians, bottom half shows the regions which were eliminated due to uniform transition test.