# Task 2: Matlab

## Image Detection

The first step that was undertaken in the image detection test was to convert the original image into an hsv image which uses hue saturation, this is done to separate the image intensity from the colour information. This is achieved through the function carHsv = rgb2hsv(carImage) which converts whatever the image is, in this case its ‘001.jpg’ into an hsv image and the result is shown in figure 2.

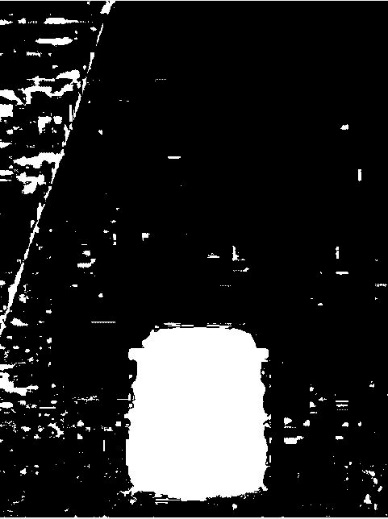


*Figure 1: Original car image* *Figure 2: Hsv car Image*

With this achieved the next step is to convert the hsv image into a grayscale or binary image in order to allow different functions to be carried out in areas of interest later on, this is achieved by using the function carBinary = im2bw(carHsv, 0.2) the result is shown in figure 3.

However in the result we can see that there a large number of holes on the image of the car as well as on the rest of the image and these need to be filled in. This is achieved using the following function carFill = imfill(carBinary, 'holes') which when tested filled in the holes within the image of the car itself not outside the car.





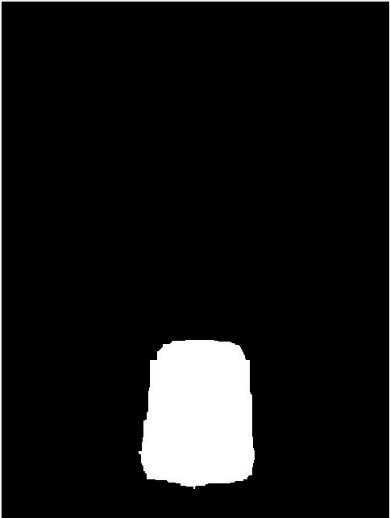
*Figure 3: Binary car image*  *Figure 4: Fill car image*

The next step erode sections of the image, this is achieved through the following functions:

carFlat = strel('square', 30);

carErode = imerode(carFill, carFlat);

carFlat created using the strel function, this allows a square structuring elements with the width of 30 elements. imerode erodes the grayscale image using the structured element of carFlat and returns the eroded image with the function carErode, the results are shown in figure 5.



*Figure 5: eroded car image*

The next step was to dilate the image further to improve the boundary detection:

carFlat2 = strel('square', 60);

carDilate = imdilate(carErode, carFlat2);

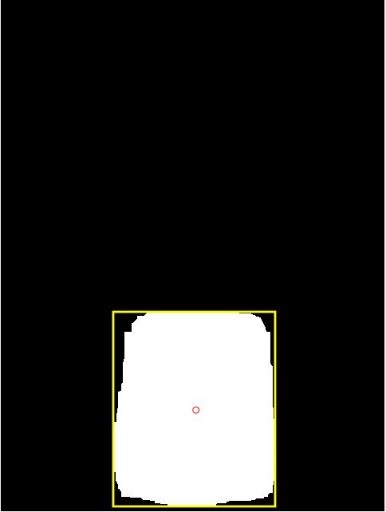
carDilate also uses the strel function as with the eroding except it uses a larger number of elements in this case it uses 60. imdilate will dailate the grayscale image using the structured element of carFlat2 and returns the dilated image with the function carDilate.

Afterwards the boundingbox was created to detect the edge of the image of the car as well as the centre point of the car. The centre point is achieved through the following function:

carMiddle = centroid(1,2).

While the bounding box is drawn using the following function: rectangle('Position', boundingBox, 'EdgeColor', 'yellow', 'Linewidth', 2);

The end result is shown in figure 6.



*Figure 6: dilated car with boundary box*

## Car Speed

The next test to be done is to measure the car speed, the first step that was taken was to calculate the centre point which was done through the following procedure: degreesFromVertical = 90 - 30 - ((640 \* 0.042)/2);

90 degrees is the vertical, 30 is the degrees from the horizon and 0.042 is the view angle.

Afterwards the angle of the image has be worked out which is done from calculating the vertical angle and then add the image and multiply it by the 0.042. Then the horizontal distance is calculated by subtracting the car view angle from the height of 7m. The change in the distance is calculated by subtracting the distance of the first image from the second image.

Then the speed per second is calculated by multiplying the distance by 10 and taken the images at 0.1 intervals.

Afterwards the speed is converted into Miles per hour which is done through the following function:

speedMPH = speedMetersPerSecond \* 2.236936;

Afterwards a loop is created so if the vehicle is travelling above 30 then its speed and below its not.

## Car Width

The next is to test the car width, firstly the coordinates for the boundingbox and then the width of the boundingbox and then work out the size of space between the side of the car and the bounding box.

This is achieved thorough the following code which is for the left car width: leftOfCarWidth = abs((480/2)- carXCoord) \* 0.042;

Both are added together to create the overall width in degrees and then afterwards the square root of the cars position is multiplied by the width, this is achieved using the following code: width = sqrt(distance1.^2) \* tand(widthdegrees);

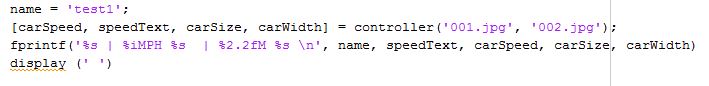
If the width is over 2.5 meters then clearance will not be given and vice versa if it’s under 2.5.

## Controller

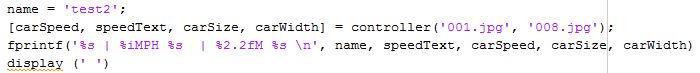
The controller is set up in order to run the functions in imageDetection, carSpeed and carWidth with how they should be written in order to carry out the tests e.g. controller (‘001.jpg’, ‘006.jpg’).

## Tests

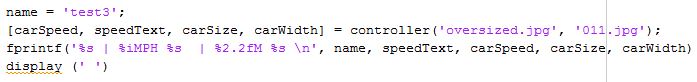
4 tests have been conducted which involved testing the speed and width of the two images together.

The first test tests images 001 and 002

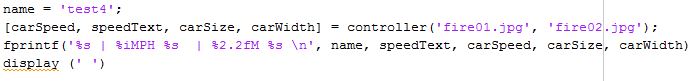
The second test tests images 001 and 008

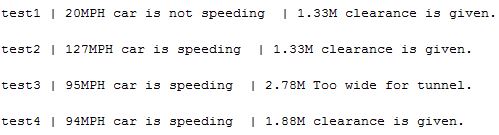


The third test tests image oversized and image 011



The final test tests the fire engine images of fire01 and fire02



The results of each of tests are shown below, they show whether or not the vehicle speeding, how fast it’s going and also the width of the vehicle and whether or not clearance can be given.