Coursework Report Task 2 (MatLab)

UP902282

January 2021

Contents

1 Task Objectives

The objective of this task was to develop and application in ***MATLAB*** that could serve as a traffic detector that could identify from an image or a set of images, what color each car was, the type of the car, if it's a Fire Engine or not, and the dimensions the car, based on the angle and distance of the camera provided from a diagram.

2 Assumptions

From the start of the program we can assume that the camera is located at an altitude of 7 meters, looking at the car at an angle of 30 degrees. From this we need to do some calculations that can be made to detect the distance from the camera to each part of the car, that, if subtracted from one another can get the size of the car, based on the amount of pixels that the car occupies. The diagram of the camera was provided in the coursework requirements. (1)

We can also assume that in every image that the camera takes, there will **always** be only one car in frame, to be detected, this way the application can prevent calculation errors from occurring and make the car detection easier. The car images are also provided by the coursework requirements.

Finally we can also assume that every pixel in the image corresponds to 0.042 degrees of view angle, and the total FOV of the camera is 30 degrees

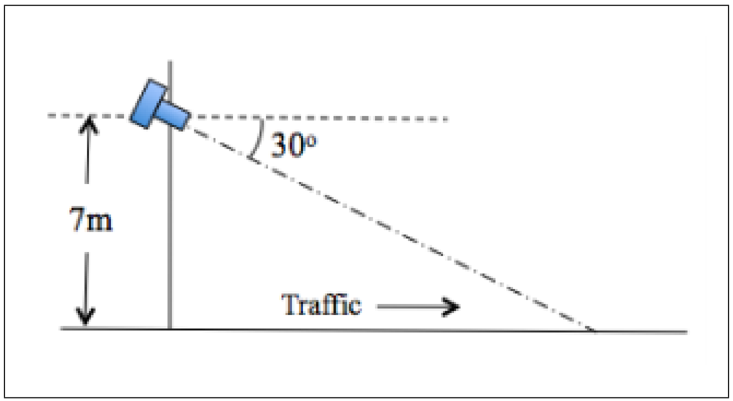


Figure 1: Camera Reference Image provided in the CW.

3 Conditions

There are several conditions that the program must respect, most of these are dependant on the type of the car that is detected.

Normal Car Conditions:
- Only one car will be in frame at the time.
- If the *speed* of the car is above 30mph, the car is speeding.
- If the *width* of the car is above 2.5meters, then the car is considered oversized.
- The car color is ***blue***.

Fire Engine Conditions:
- Only one car will be in frame at the time.
- Speeding and Size of the Fire Engine will be ignored
- Color of the Fire Engine is ***red***.

4 Processing Steps

After we know all the conditions we can start to ”deconstruct” the image, we do this, first by reading the image (2), after we get the image we need to find a way to get the color of the car as well as the dimensions, which means we need to isolate the car from the background, we can do this by getting all the color values of the pixels, not the *RGB* values but instead the HSV values, which mean Hue, Saturation and Value, to get the color blue, we can say the Hue is 0, the Saturation can be any value, and the for the Value we need to define a threshold, we then get the processed image from reference number (3), and this is done with the code on reference (4):

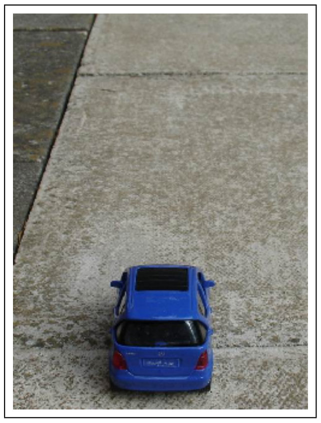


Figure 2: Initial Image of the car

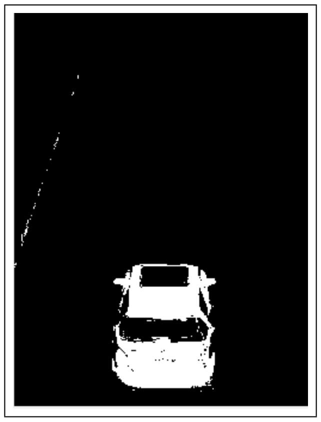
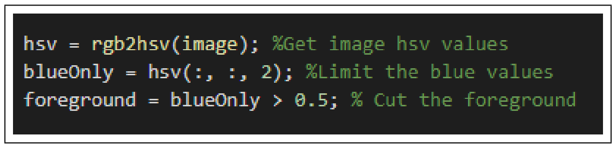


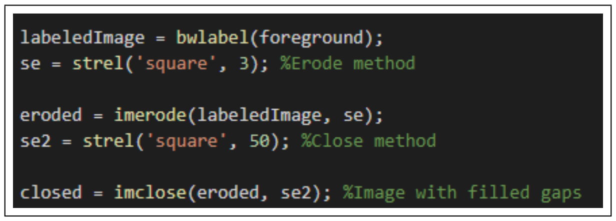
Figure 3: Image after HSV Conversion



```
hsv = rgb2hsv(image); %Get image hsv values
blueOnly = hsv(:, :, 2); %Limit the blue values
foreground = blueOnly > 0.5; % Cut the foreground
```

Figure 4: Code to get blue from HSV

After we isolate the car from the image we can see that there are some small white specs at the left of the image, this can cause problems when we create the BoundingBox as it might detect those specs as something that we want to stay, for simplicity it is better to remove them, so we use the following code (5) that works by ”eroding”/removing the noise from the image we get the following result (6)



```
labeledImage = bwlabel(foreground);
se = strel('square', 3); %Erode method

eroded = imerode(labeledImage, se);
se2 = strel('square', 50); %Close method

closed = imclose(eroded, se2); %Image with filled gaps
```

Figure 5: Code to erode the image

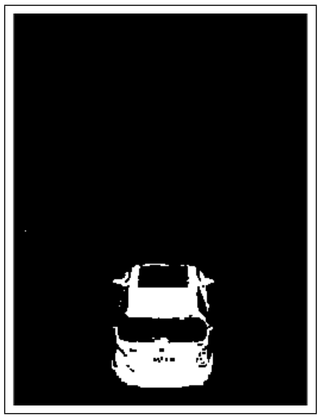
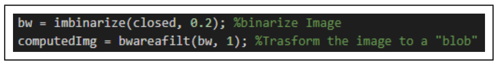


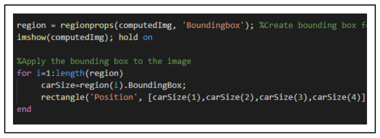
Figure 6: Image after Noise Erosion

Finally we can grab the eroded image and uniform it, by transforming what we got into a "blob" that can basically turn the car into a white square, so that it's easier to add a BoundingBox, which we can make with the following code



```
bw = imbinarize(closed, 0.2); %binarize Image
computedImg = bwareafilt(bw, 1); %Trasform the image to a "blob"
```

Figure 7: Code to create a blob of the car



```
region = regionprops(computedImg, 'Boundingbox'); %Create bounding box f
imshow(computedImg); hold on

%Apply the bounding box to the image
for i=1:length(region)
    carSize=region(i).BoundingBox;
    rectangle('Position', [carSize(1),carSize(2),carSize(3),carSize(4)]
end
```

Figure 8: Code to create a BoundingBox

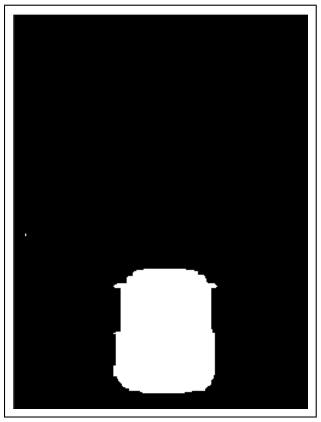


Figure 9: Blob Image after transformation

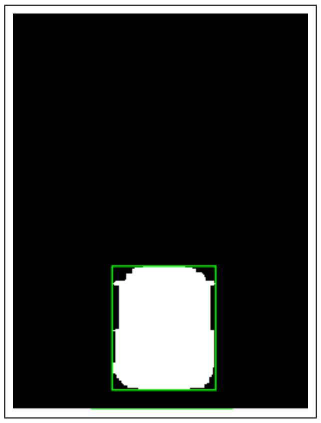


Figure 10: BoundingBox Image after creation

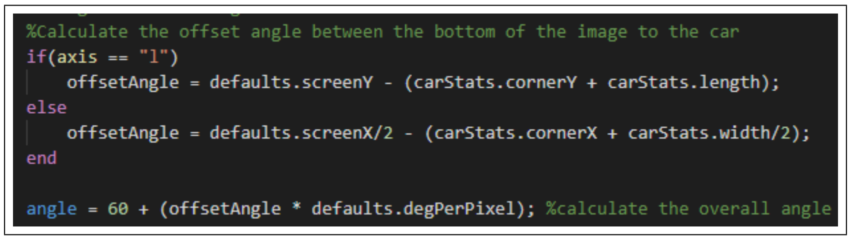
After going through all these steps we are pretty much done, and can use this code to process all the images, which the X and Y values of the top corner, plus the width and length finally these just need to be converted to Real Sizes.

5 Calculations

As previously mentioned, we already have the dimensions of the car, but these need to be converted to a real size, since at the moment, these are measured in pixels. To calculate this we will need to use the diagram of the camera mentioned above (1). With this reference we have enough information to be able to calculate the distance from the camera which is made with basic trigonometry.

Before going into the code, from the camera reference image, we can see that the angle of the camera FOV, plus the angle until it's perpendicular to the floor, is 90 degrees, if we subtract the FOV from these 90 degrees we get 60 degrees which is the value we will use in our calculations.

First we will need to get the angle from the border of the image, to the point we want to calculate, which in our case, is both the top and bottom of the car, for the length, and the left and right border of the car for the width. After we get the values in pixels between the image and the borders of the car, we multiply by 0.042 which is the degree every pixel in the image corresponds.

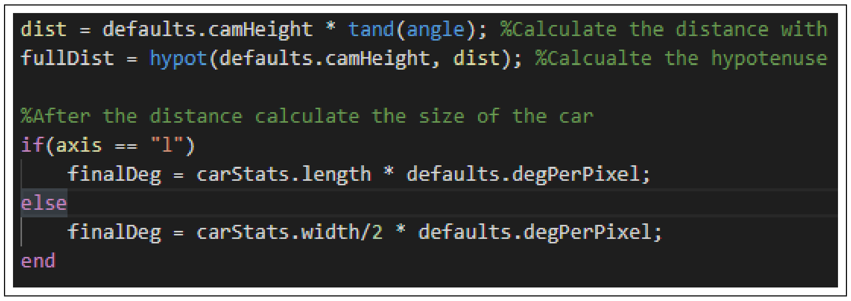


```
%Calculate the offset angle between the bottom of the image to the car
if(axis == "l")
    offsetAngle = defaults.screenY - (carStats.cornerY + carStats.length);
else
    offsetAngle = defaults.screenX/2 - (carStats.cornerX + carStats.width/2);
end

angle = 60 + (offsetAngle * defaults.degPerPixel); %calculate the overall angle
```

Figure 11: Code to calculate the angle

After we get this angle, we need to calculate the distance between the camera and the car, we can do this by multiplying the height of the camera with the tangent of the angle we just got.
After we get the distance we just need to get a new angle which represents the size of the car, but with an angle.

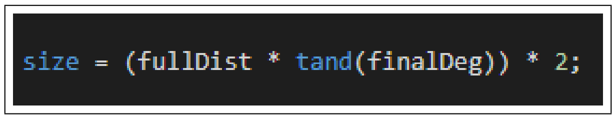


```
dist = defaults.camHeight * tand(angle); %Calculate the distance with
fullDist = hypot(defaults.camHeight, dist); %Calcualte the hypotenuse

%After the distance calculate the size of the car
if(axis == "l")
    finalDeg = carStats.length * defaults.degPerPixel;
else
    finalDeg = carStats.width/2 * defaults.degPerPixel;
end
```

Figure 12: Code to calculate the distance

And finally we just need the last calculation to get the width and height of the car, which multiplies the distance with the tangent of the degree we just got, multiplying this by 2 to get the size in meters.



```
size = (fullDist * tand(finalDeg)) * 2;
```

Figure 13: Code to get the car size

6 How to run

To run the program you just need to run the file ***test.m***, which will then display all the possible test cases. These tests contains all the possible speeds, the size of the car, if it's oversized, speeding or a Fire Engine, and all the other values.