ME 598
Introduction to Robotics

Fall 2022

Image Processing for Vision-Based Tasks

Group # R3

28 November 2022

Graduate students:

"I pledge that I have abided by the Graduate Student Code of Academic Integrity."

The report has been prepared by:

1. Lab leader: Peera Tienthong
2. Sakshi Ranjeet Pol
3. Krupansh Manish Shah
4. Shanmukha Sampath Kumar Tiriveedhi

Table of Contents

Abstract

Image processing is a method to perform some specific operations on an image to get an enhanced image or extract some useful information from it. It is a type of signal processing in which the input is an image and the output may be either an image or characteristics associated with that image. This lab experiment expatiates the process, computation, simulation, and examples which was theoretically learned in the class and later practically applied in this lab. In the lab, we have performed different tasks of image processing and learned to have our robot visualize. This task helped us in implementing and analyzing different challenges of Image Processing for Vision-Based Tasks.

Introduction

Image processing is performing various operations on an image in an attempt to optimize it and extract relevant information. In this input is an image and the output is either that image or its characteristics. Image Processing is a critical research field in engineering. In this lab, we are using image processing to locate specific landmarks within a mobile robot's field of view using real-world images. Here we are using "blob analysis". Blob analysis is a technique for examining an image that has undergone binarization processing. Binarization processing is the process of turning a gray image into 0 and 1 values by using a chosen threshold value as a standard. This method allows you to concentrate fully on the object of inspection while conducting numerous analyses. Blob analysis is the simplest fundamental approach of image processing for assessing an object's shape properties, such as the presence, number, area, position, length, and orientation of lumps.

Theory & Experimental Procedure

Part 1: Camera Setup

MATLAB on the system was set up with support for webcam. As indicated in the tutorial, a variable is created to load the feed of the webcam into runtime. A snapshot is captured at an instance using the *snapshot* method. Although the Resolution of the captured snapshot be preset, the default camera resolution is used for this lab purpose. The provided method first displays the snapshot in a *figure* window of MATLAB which is then saved as a *jpg* image. These images can be used for processing in the later parts. Please note once the webcam is no longer in use, it is advisable to delete the instance from the runtime as it holds the connection and prevents other programs from accessing the webcam. *Delete* can be used to delete the instance from the runtime instead of using *clear* which clears all the runtime variables.

Part 2: Configure Color Detector

The *Color Thresholder* app of MATLAB is generally used to play with the color spaces of an image and observe the changes in values visually when changing them. The use case is to determine the Hue, Saturation and Value(HSV) of the image to identify the traffic cones in the image provided. Among the provided set of training images, one image is loaded into the color thresholder app and its HSV color space is selected. After varying the values of each channel, the traffic cone(s) in the image are isolated. It is possible that there can be some gaps in the cones due to the selected channel values, but these can be filled using MATLAB functions which are explained in the later sections.

Once the HSV values are determined to isolate the cones in an image, a function named *coneThreshold* is exported from the Color Thresholder app. This function can be used to apply the channel filters to many images. Using another application of MATLAB, Image Batch Processor, the captured HSV values are applied to other images in the training set and the final thresholds for each channel are determined as provided in table 1.

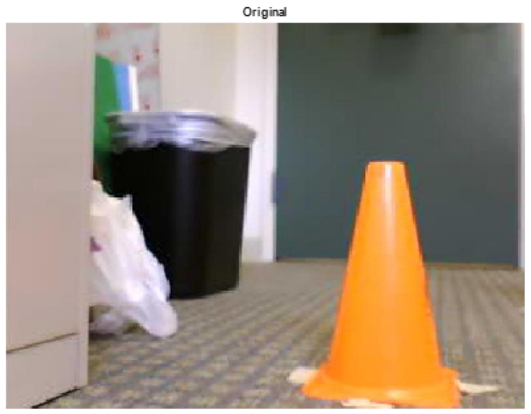
The values captured are good only for the training image set. These might need to be adjusted for the test set and also the new set of images captured.

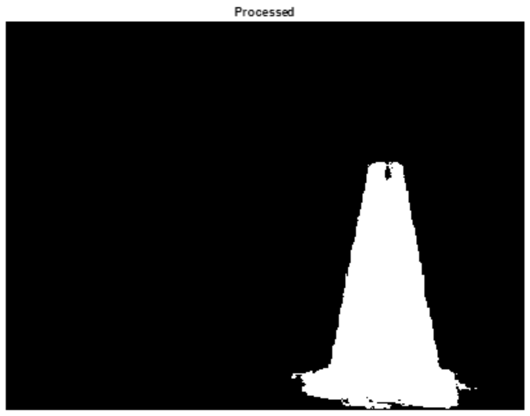
Channel	Minimum	Maximum
Hue (H)	0.031	0.129
Saturation (S)	0.174	1
Value (V)	0.956	1

Table 1: Captured HSV Values for Train Image Set

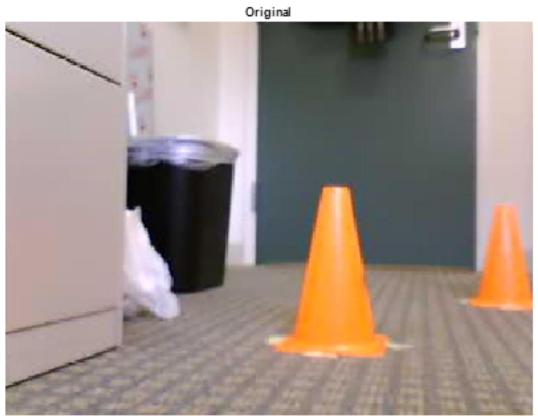
Once the thresholds are determined, the processed images are stored so that they can be used for further processing. MATLAB provides an option to export the processed images into the workspace and also as files. The output images are first exported into the workspace first to generate the side by side format of the original and processed images for a comparison. These comparisons are shown below. The output images are also exported as files to be later used for blob analysis and target detection.

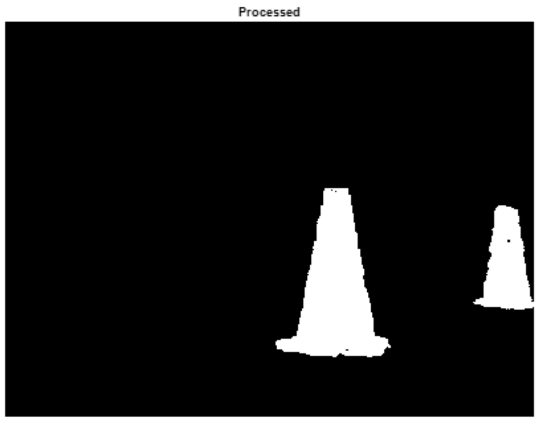
Example 1:



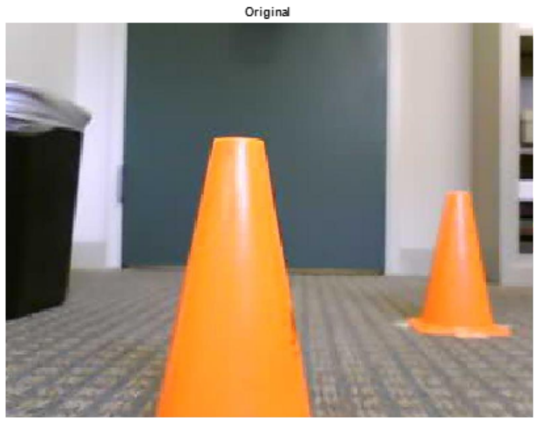


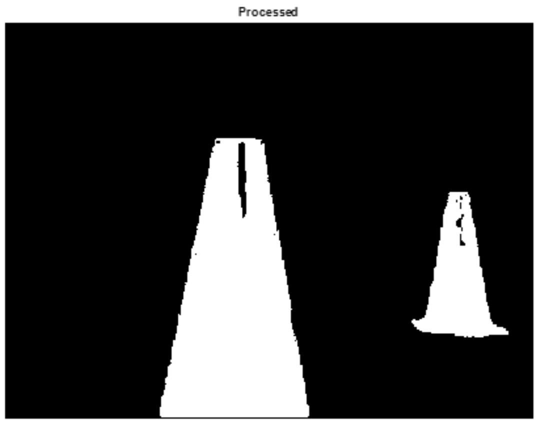
Example 2:



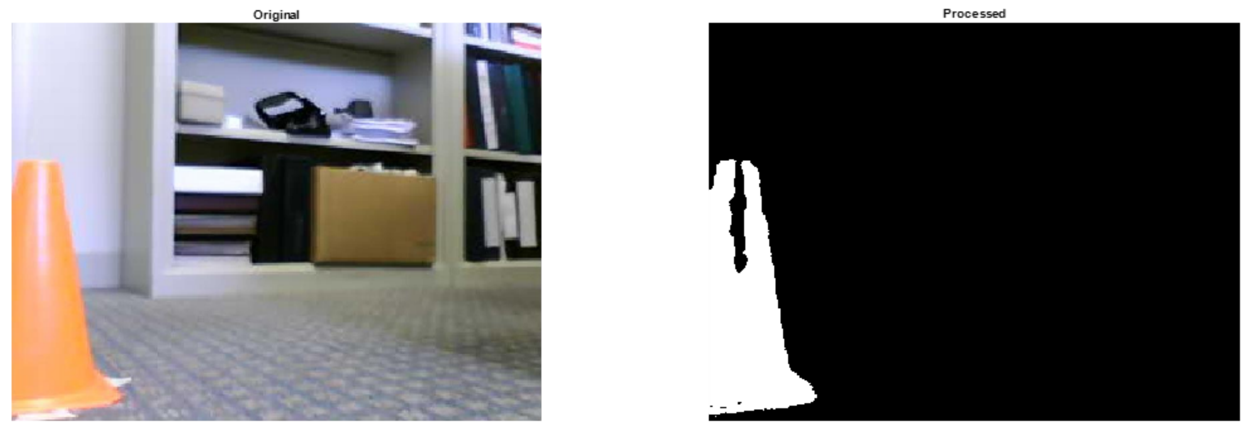


Example 3:

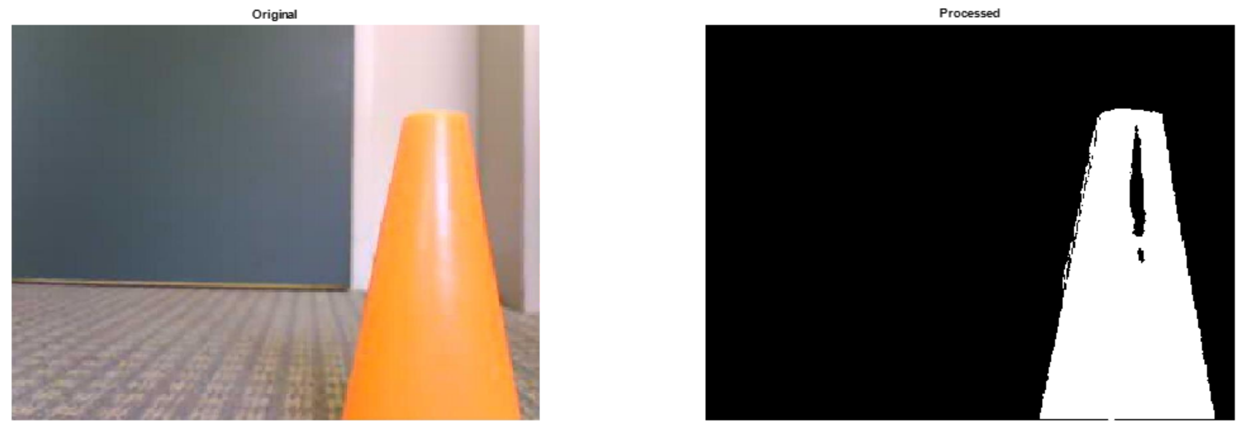




Example 4:



Example 5:



It can be observed that there are visible gaps in the cones that are blacked out which means that the color thresholder detects those areas to be out of bounds as the color captured doesn't match with the other surface area of the cone. Below are 5 set of train images with original and processed. The code used for this part is provided in the appendix of this report. (MATLAB Function - coneThreshold.m, ImageExporting.m)

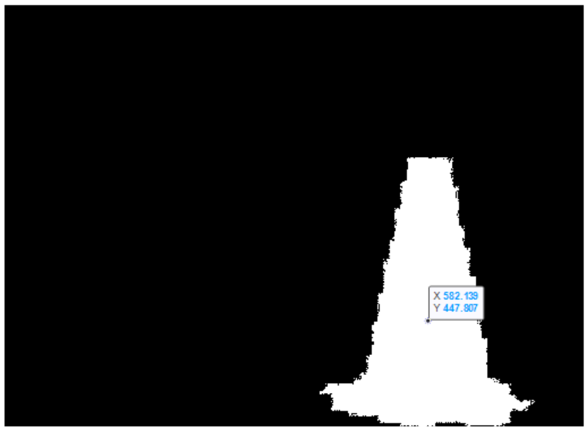
Part 3: Configuring Blob Analysis & Target Detection

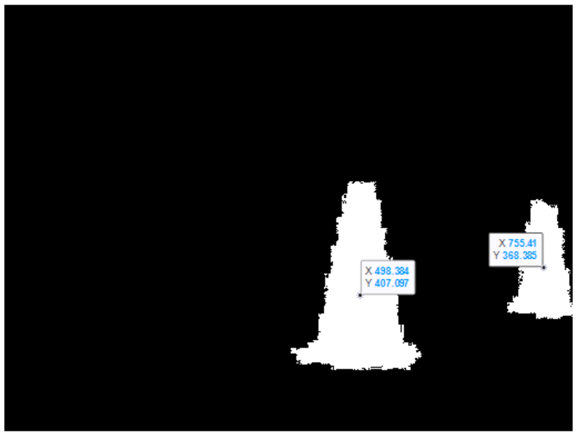
The goal of this part is to find the areas and the positions of the cones in the processed image output of part 2. It can be observed from the sample outputs that the cones in the image are not always complete and might have some gaps. This can occur due to the glare effect on the cones if the cones have shiny surfaces or the light is too much in the scene. There can be many other such factors like camera properties, light and surface properties in the scene. MATLAB's image processing library has many functions which can be used to tackle such problems. For our use case, we go with the below steps.

3a. First step is to filter the smaller regions from the image and only consider pixel areas that are larger than a certain threshold. This threshold is determined to be at 700 for this use case and is arrived at after observing the train, test and new test sets of images with cones. If a cone or any area is less than 700 pixels in size, then it is ignored. *bwareaopen* function in MATLAB is used for performing this step.

The next step is to fill the gaps in the continuous areas that are identified in the first step. As it can be seen from the example provided in the assignment, there can be some gaps on the object surface caused due to overlap of smaller objects or glare or other such effects. To mitigate these, *the imfill* function of MATLAB is used. This step is necessary to calculate the correct areas of the surfaces identified which are key in processing going forward.

After the gaps are filled, properties of the individual objects are identified by labeling them. The *bwlabel* function of MATLAB is used for this. The output of this function is then supplied to the *regionprops* function which to identify the centroids and areas of the regions identified in the image. Among the areas identified, only the ones beyond a threshold are considered. This threshold is set to be at 7000 for the train test and new test set of images. The custom code written for this part uses all these functions and iterates over the list of areas identified from largest to smallest of the cones. Once the cones are identified, they are plotted in a figure with their centroids. Fig. 3.1-5 are a few examples of images with cone centroids plotted. Fig 3.6 is an example where there are no cones detected in an image.





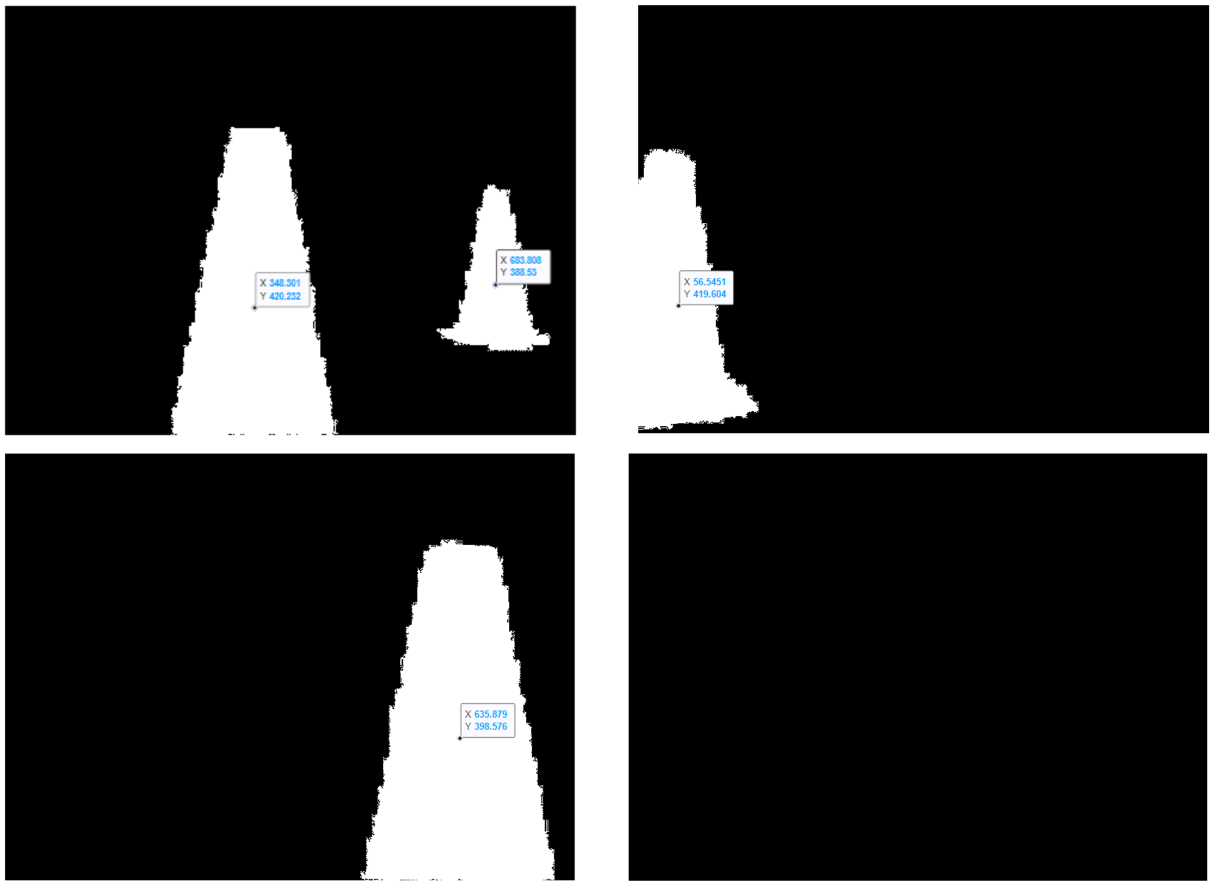


Fig 3.1-6: Centroid of cones plotted on the color detected images

3b. The next goal of this part is to plot the overhead view of the image Field of view to identify where the cones are with respect to the point of observation. To find the orientation of an object with respect to the robot FOV, first calculate the center of the image in x direction. The object's centroid is obtained by part 3a. Here the orientation of the centroid is based on the difference between its centroid and the image center in x direction as in the figure 3.7.

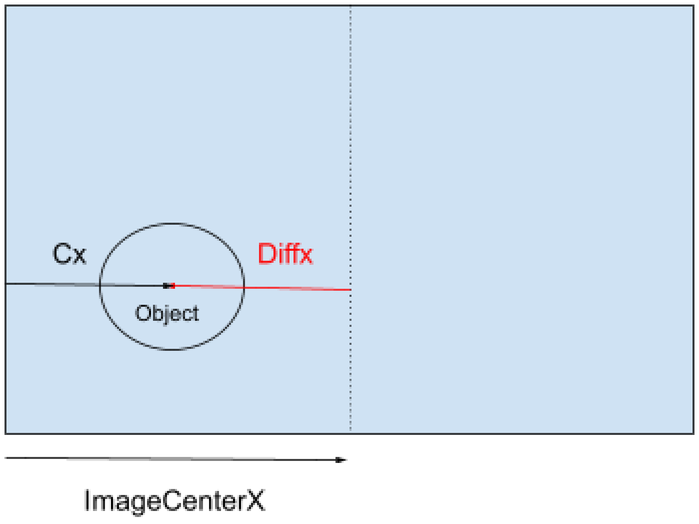


Fig 3.7: Centroid and Image center

According to the normal robot's FOV, the left-most and the right-most pixels are separated by 60 degrees referred to the image center as illustrated in the figure 3.8. The angle of Diffx is then compared with the half angle of the FOV, which is 30 degrees. By taking triangular similarity into account, the angle of Diffx, θ, can be found by.

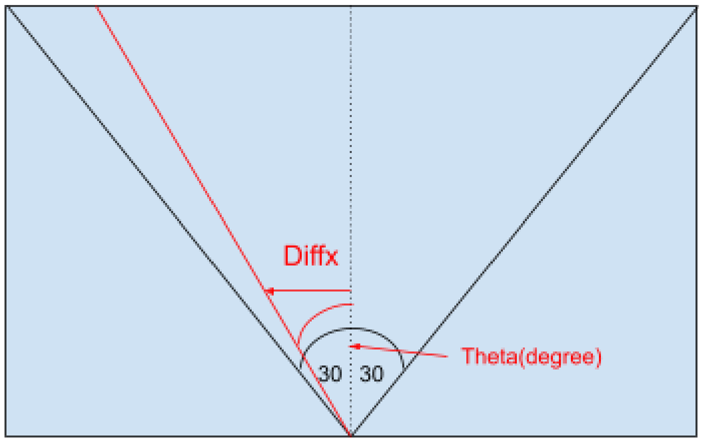


Fig 3.8: Overhead view - Angle between the center and position vector of the area centroid

$$Diff_x = ImageCenterX - C_x$$

$$Diff_x = \frac{sin(\theta) \times ImageCenterX}{sin(30)}$$

$$\theta = sin^{-1}(Diff_x \times sin(30)/ImageCenterX)$$

Here $Diff_x$ is the difference between the x-coordinates of the image center and the cone centroid. The θ value in the equation above can be considered as the orientation of the cones in the image with respect to the observation point i.e. camera. If the value of $Diff_x$ is negative then θ is also negative indicating the cone is on the right half of the image. If it is positive then the cone is in the left half.

The next step is to identify the proximity of the cone. This proximity is determined as a categorical value 1 being the cone is near and 3 being far from the camera and 2 is somewhere in between. From the observations in the entire test set, the pixel area of the closest cone is around 43000 and the farthest is at around 8500 pixels. The difference between these values calculated is used to determine the proximity of the cone. If the area is about one third of the difference or less then it is considered to be far, if it is more than 2/3rds of the difference then it is near and it is somewhere in between if the value is anything else.

Once the angles and proximities are calculated, they are given as input to the *overheadview* function provided in the assignment which generates an overhead view plot for the scene in the image. At the end of this part, traffic cones are identified in the image and also their positions. These can be very important for robot movement in its operational field. The overhead views calibrated for the input images of examples considered are shown in fig. 3.9-13 where the left side images are the color detected images and the right side images are the overhead view plots. Fig 3.14 is an example of an image with no cones. The code used for this is provided in the appendix of this report. (MATLAB Functions - blobAnalysis.m, get_3b.m, ME598_Grp3_Lab03_Part3.m and ImageExporting.m)

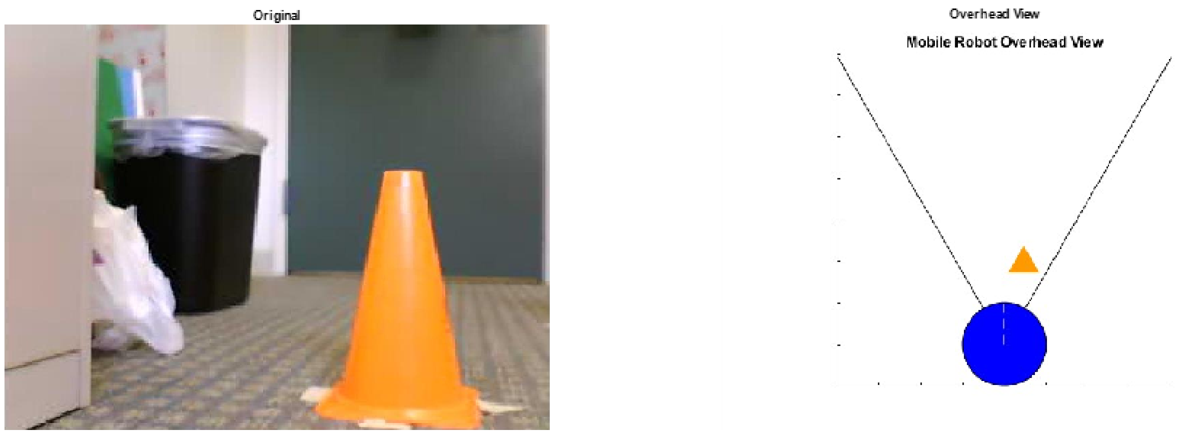


Fig 3.9: Example 1 - Original and Overhead view

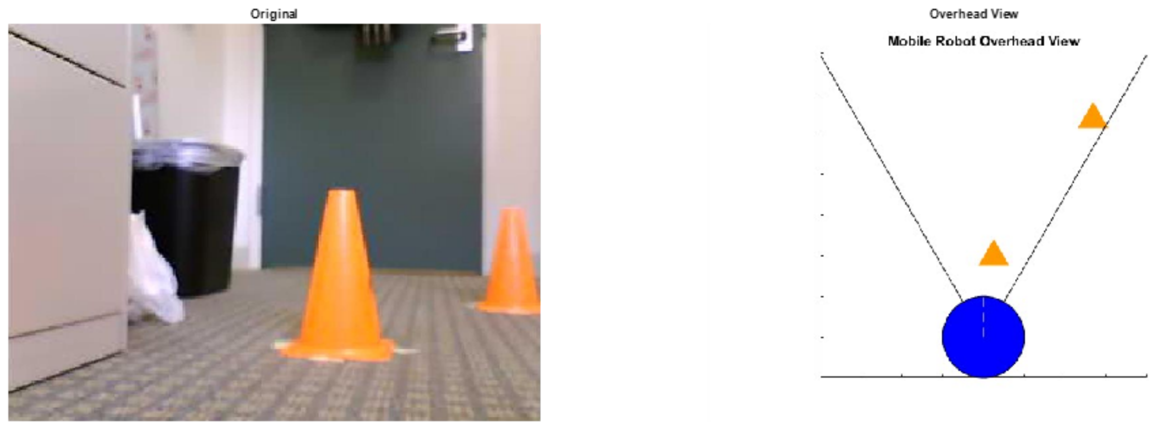


Fig 3.10: Example 2 - Original and Overhead view

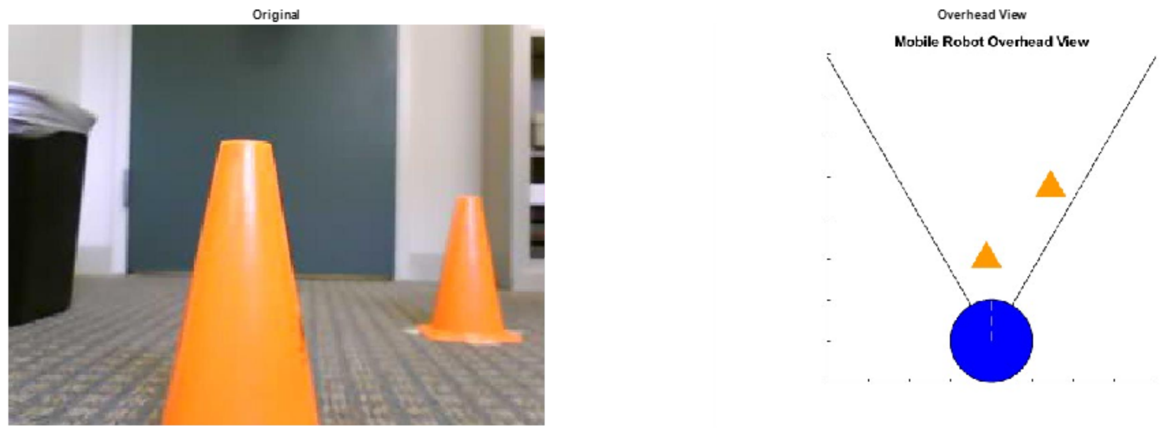


Fig 3.11: Example 3 - Original and Overhead view

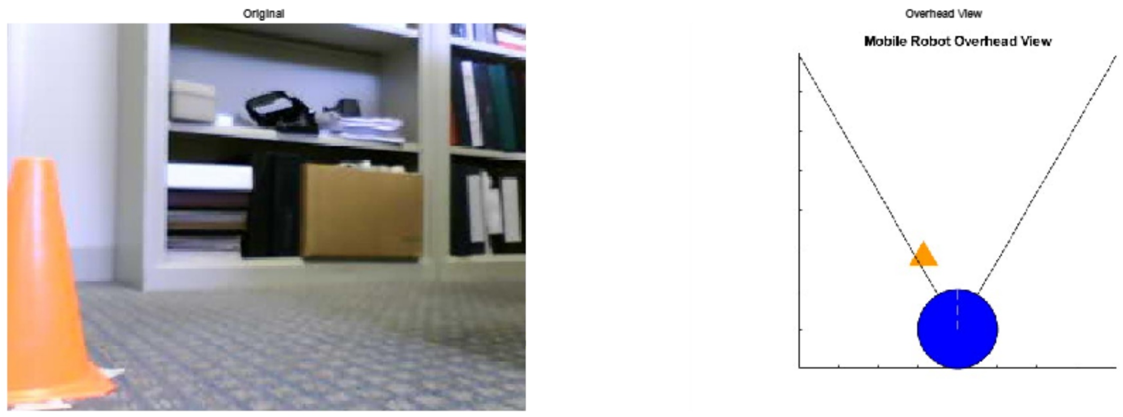
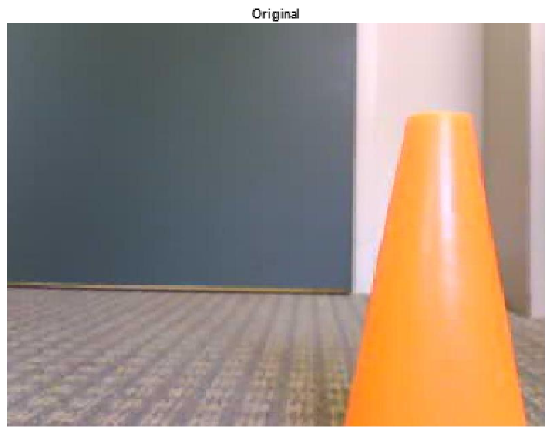


Fig 3.12: Example 4 - Original and Overhead view



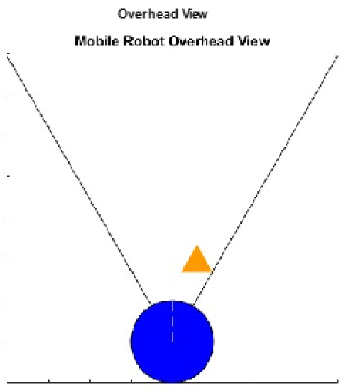
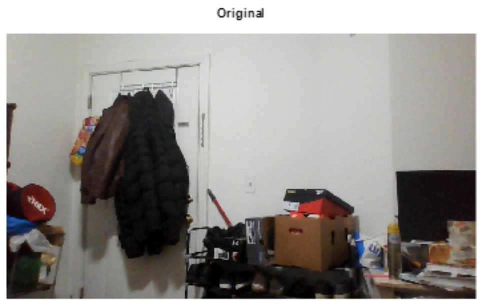


Fig 3.13: Example 5 - Original and Overhead view



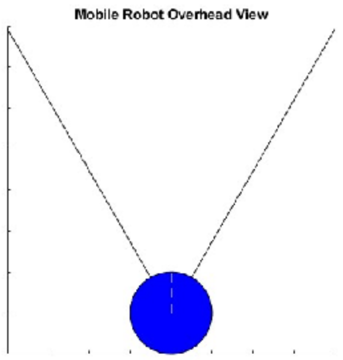


Fig 3.14: Example 6 - Original and Overhead view

Part 4: Testing Blob Analysis & Target Detection

The process of identifying the traffic cones in an image, as discussed in parts 2 and 3 is repeated for a different set of images known as the test set. These images are provided as part of the assignment. The coneThreshold function is used to identify the cones in the images and the output is passed on to the next functions of blob analysis and target detection where the positions and orientations of the cones are identified. No changes were made to the HSV and area thresholds i.e. the values used for the training set held good for the test set as well. The overhead view of the image is then plotted using the function provided. MATLAB code of parts 2 & 3 are used for accomplishing this task which are provided in the appendix.

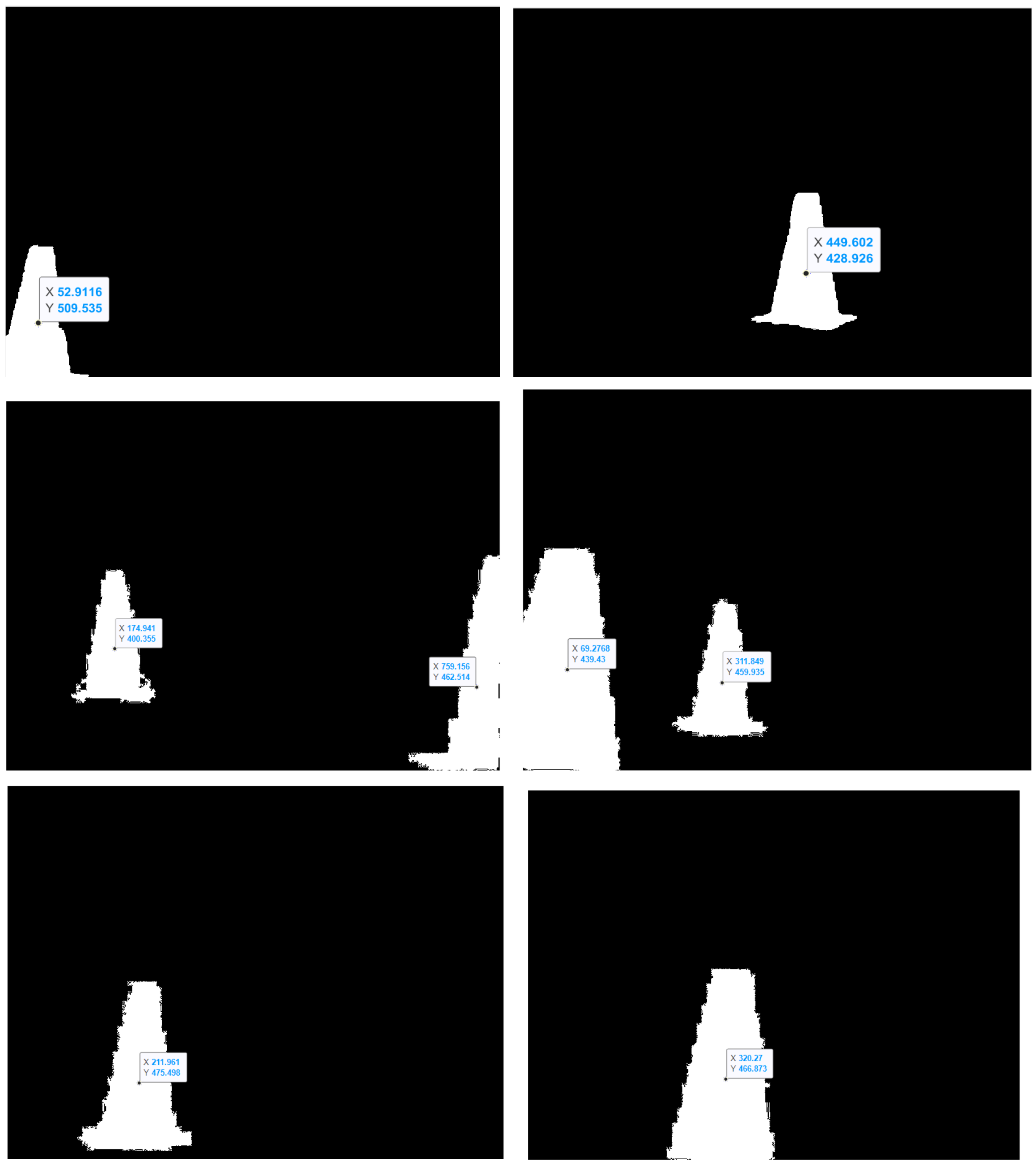


Figure 4.1-6: Centroid of cones plotted on the color detected images

The original and overhead view for the above examples are shown in figs 4.7-12.

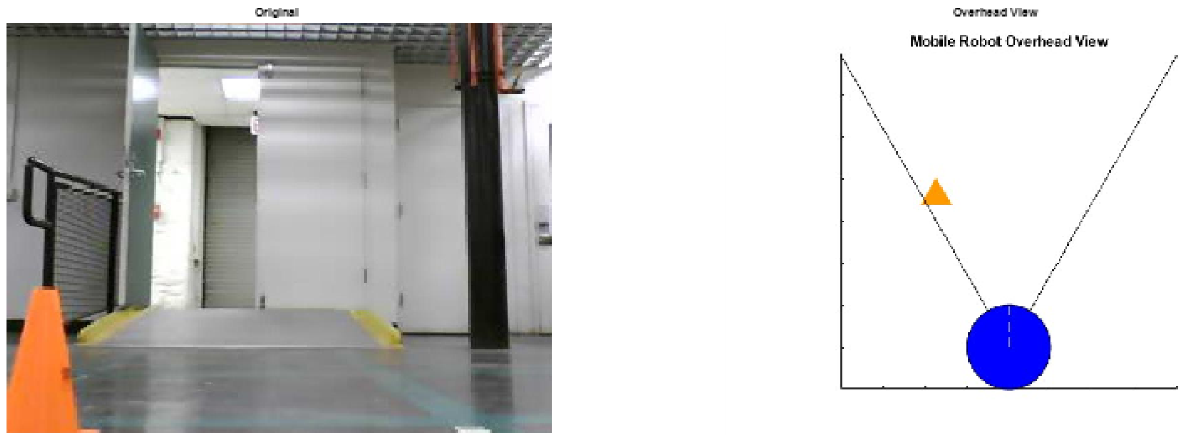


Fig 4.7: Test Image Example 1 - Original and Overhead view

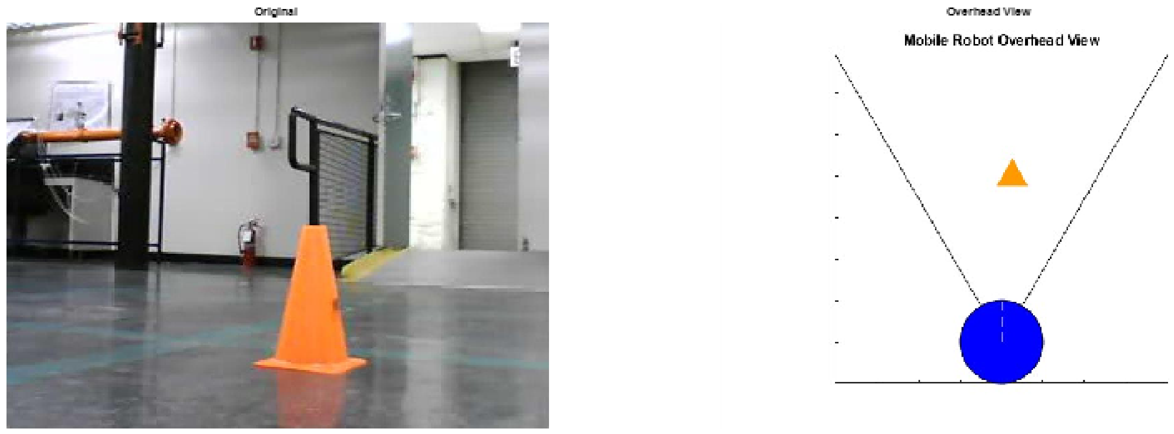


Fig 4.8: Test Image Example 2 - Original and Overhead view

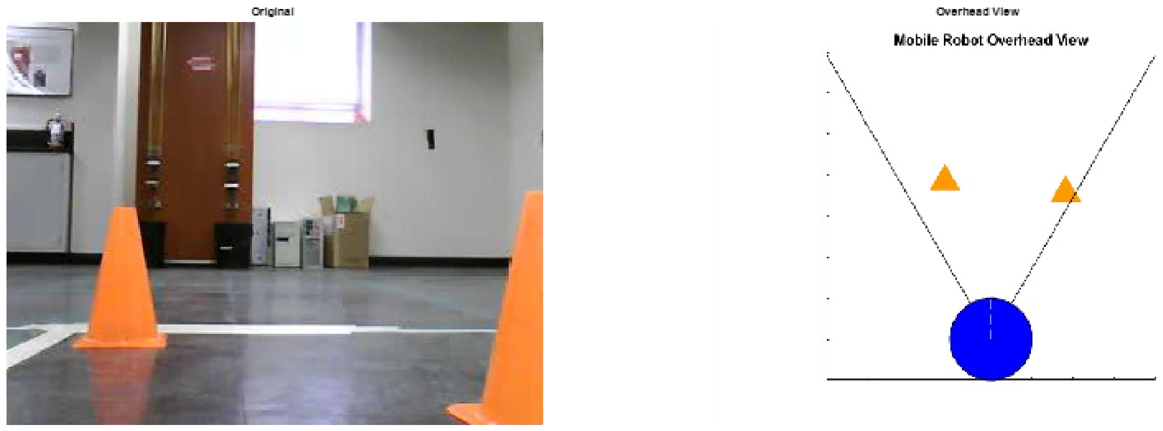
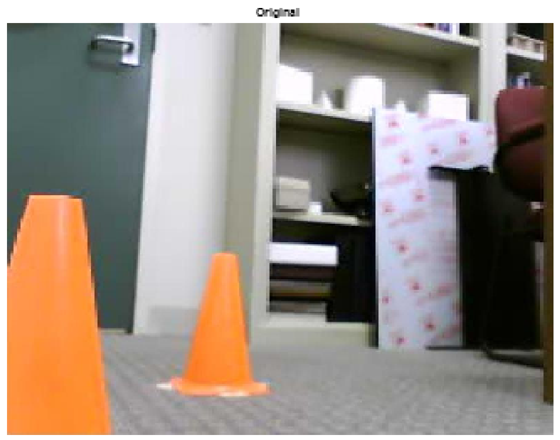


Fig 4.9: Test Image Example 3 - Original and Overhead view



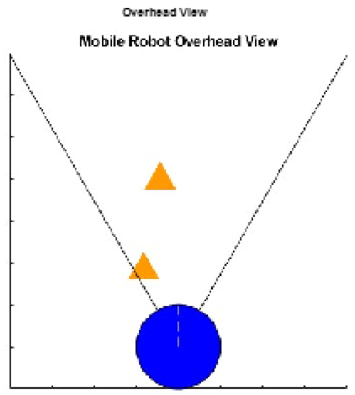
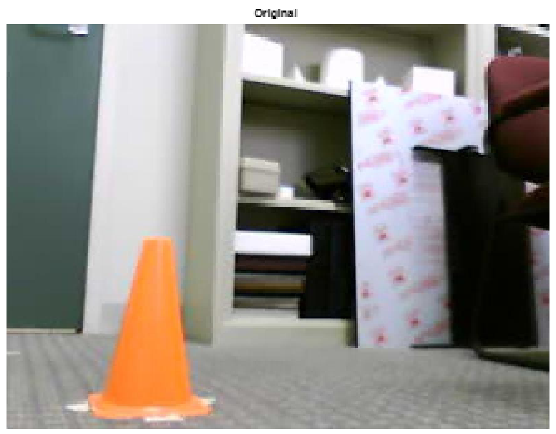


Fig 4.10: Test Image Example 4 - Original and Overhead view



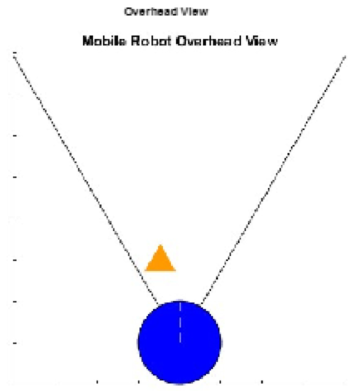
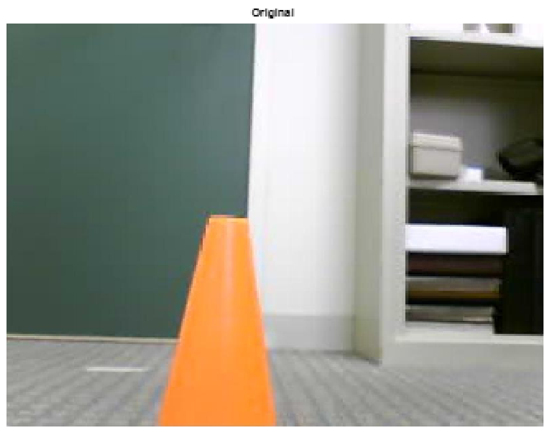


Fig 4.11: Test Image Example 5 - Original and Overhead view



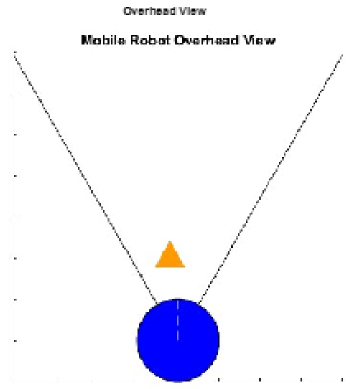


Fig 4.12: Test Image Example 6 - Original and Overhead view

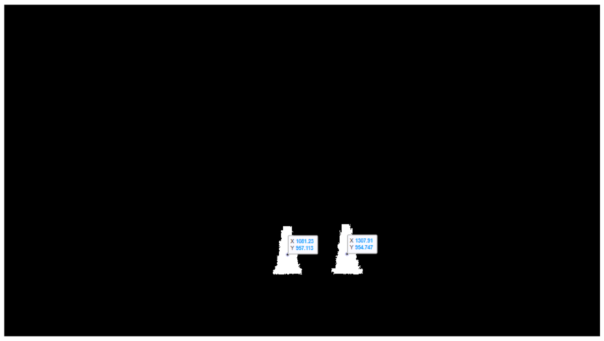
Part 5: Vision-Based Detection with Newly-Captured Images

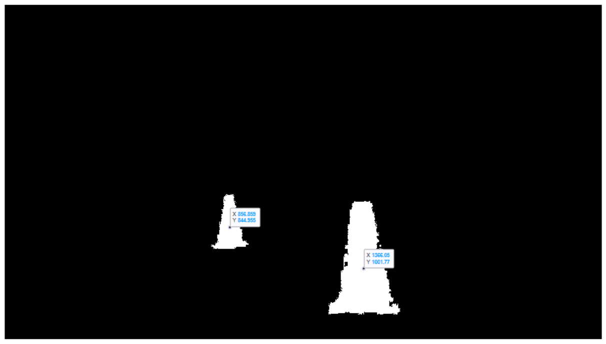
For this part, a new set of images are captured using a webcam on one of the laptop computers of a team member. The coneThreshold function had to be modified as the HSV values to identify the cones are different for this set of images. The values are as mentioned in table 2. Other thresholds of areas hold good for these images as well.

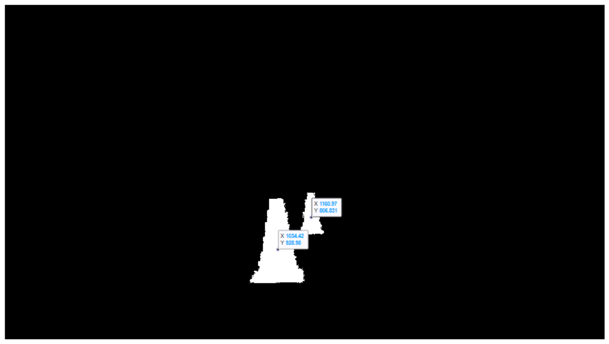
Channel	Minimum	Maximum
Hue (H)	0.005	0.077
Saturation (S)	0.405	0.903
Value (V)	0.756	1

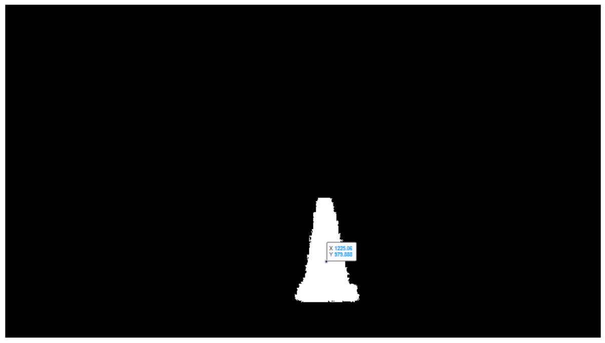
Table 2: Captured HSV Values for New Test Image Set

Once the new thresholds are determined for the image set, image batch processor is used to apply it for some images and the output can be observed in the figs. 5.1-6. Figure 5.7 & 8 show special cases where the cones were overlapping with one another, but the blob analysis detects it as a single object as it is a continuous region.









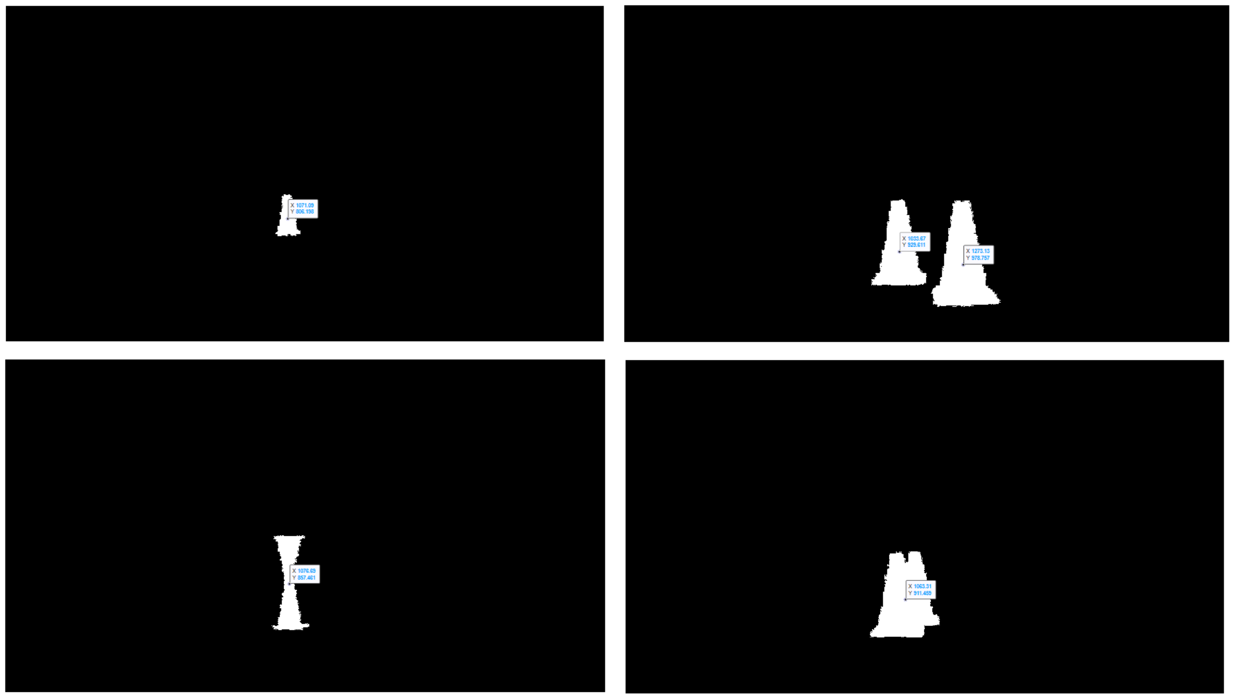


Figure 5.1-8: Centroid of cones plotted on the color detected images

The original and overhead view for the above examples are shown in figs 5.9-16.

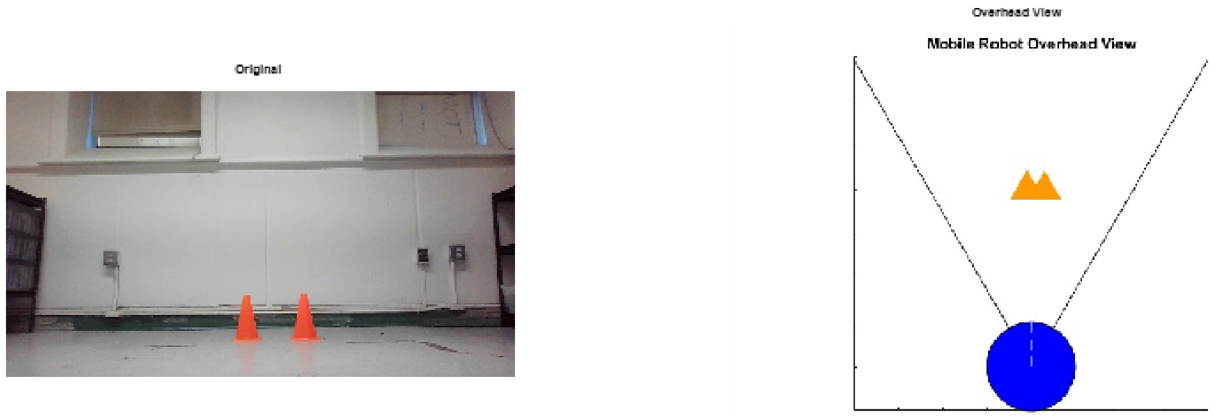
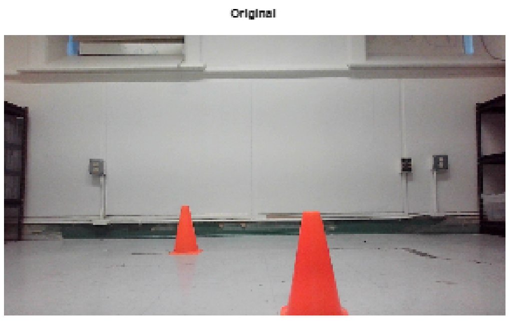


Fig 5.9: New Test Image Example 1 - Original and Overhead view



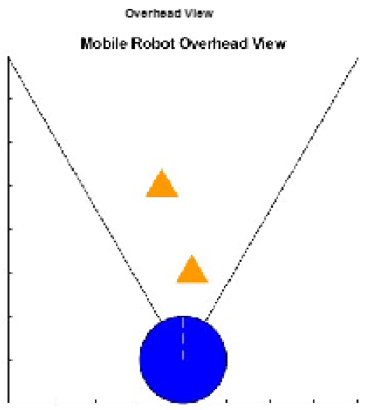
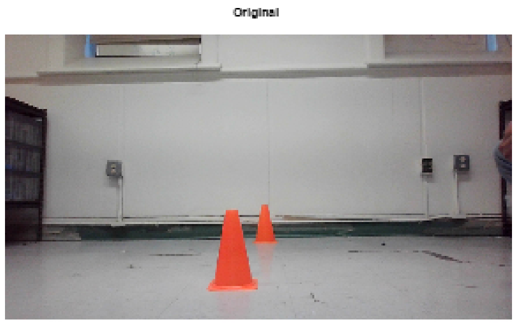


Fig 5.10: New Test Image Example 2 - Original and Overhead view



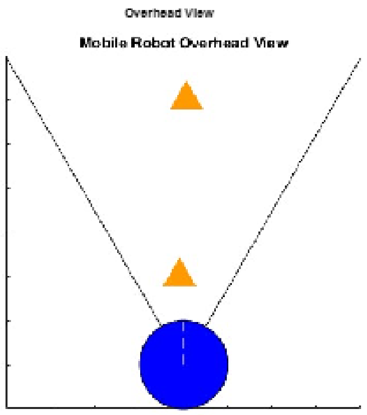
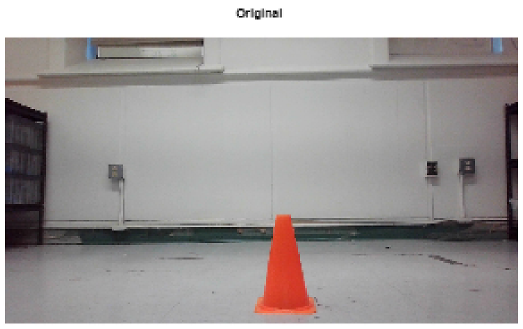


Fig 5.11: New Test Image Example 3 - Original and Overhead view



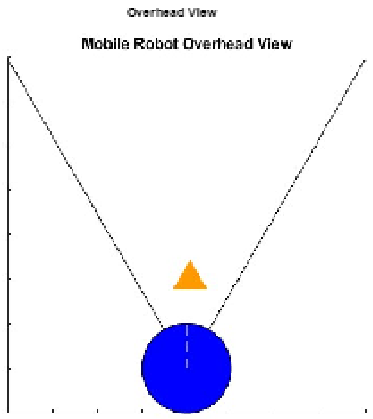
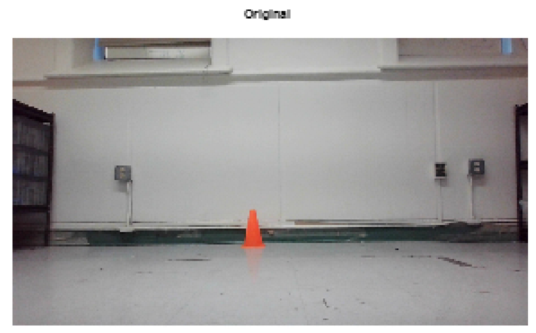


Fig 5.12: New Test Image Example 4 - Original and Overhead view



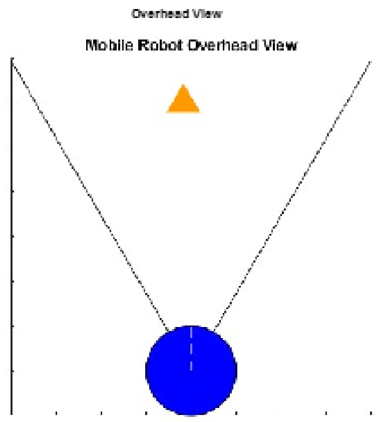
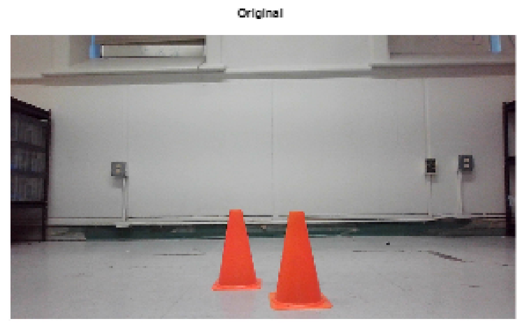


Fig 5.13: New Test Image Example 5 - Original and Overhead view



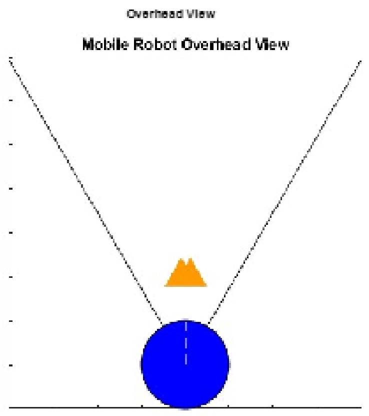
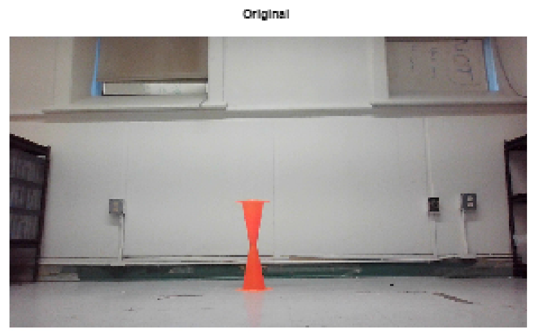


Fig 5.14: New Test Image Example 6 - Original and Overhead view



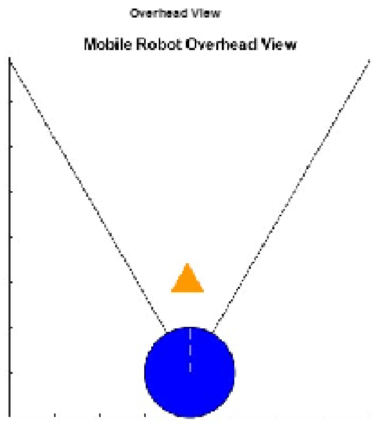
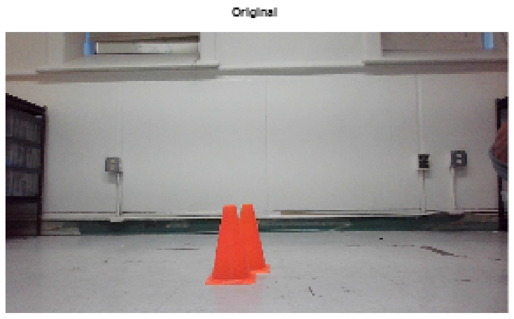


Fig 5.15: New Test Image Example 7 - Original and Overhead view



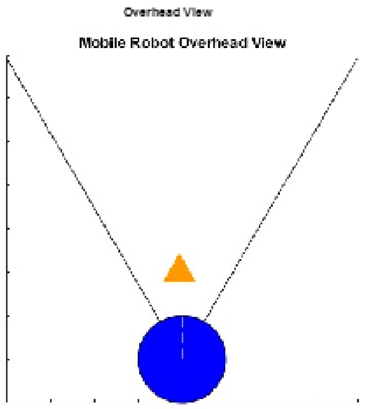


Fig 5.16: New Test Image Example 8 - Original and Overhead view

Examples 7 and 8 are special cases which identify one of the drawbacks of this approach. The code detects the 2 cones (overlapping) as a single object and places it right in front of the reference point as observed in the overhead view plot due to the large surface area in color thresholding when in the real scene they are not in similar positions. So, this model cannot be used as a one size fits all and hence the inspiration to look for better approaches in image processing. MATLAB code of parts 2 & 3 are used for accomplishing this task which are provided in the appendix.

Part 6: Student Feedback

This lab was completely new for most of us. Image processing is one of the important areas of robotics and it is interesting to see such simple approaches to identify the objects in the operating environment of a robot. This was a neat introduction and can be explored further. No major difficulties were encountered as everything was done on a laptop computer. Once the necessary data was collected, it was easy to come up with and execute the code required to perform image processing tasks. Though the execution of the code took relatively less time, the concepts involved needed to be explored and understood.

Discussion

Part 3. The color space used to identify the cones is only good for the use case in this assignment, because the color of the traffic cones can differ depending on the camera and environment.

As observed, if there is an overlap of other objects on the traffic cones, then they might not be detected and if there is too much glare, the cone can be identified completely differently. For the blob analysis and target detection, the camera is assumed to have the angle in the field of view as 60 degrees. But this can't be assumed for all the cameras. Some have a wider field of view and some have smaller ones. But for the use case of this lab, it is good to go with the default case of 60 degs. The law of proportions between the distance from the center to the object centroid and half of image length is used to determine the angle at the center. This angle is bipartisan and the positive indicates the left side whereas the right is given by the negative of the difference. For determining the proximity, the thresholds of the cone areas had to be determined manually. Though the same ones worked for train, test and new test sets this time, this might not be the case when different cameras are used. One example is also provided to see if the algorithm identifies any cones when there are none in the image captured and it can be seen that the algorithm behaves correctly.

Part 4. Testing the blob analysis and target detection on the test set of images was done without changing/tuning any parameters. It is observed in the examples that none of the cones have been identified as far from the reference point and they don't seem to be by looking at the images. It can be argued that the thresholds had to be set differently to gauge the proximity and multiple factors can influence them but they are kept the same as used for the training set. Once all the examples were processed, the output, overhead view, was compared to the original scene image.

Part 5. For the new test set, the thresholds have to be adjusted to extract the cones from the images. The area thresholds were kept the same as the new areas also fell in similar brackets and the images were processed. This approach has some drawbacks as identified in the last 2 examples where the code detects 2 cones (overlapping) as a single object and places it right in front of the reference point as observed in the overhead view plot due to the large surface area in color thresholding when in the real scene they are not in similar positions. So, this model cannot be used as a one size fits all and hence can be an inspiration to look for better approaches in image processing.

All in all, this was an interesting experiment to see how the simple techniques can be used to approach solutions to complex problems in image processing.

References

[1] Introduction to Autonomous Mobile Robots; Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza

[2] MATLAB Examples and API Reference: https://www.mathworks.com/help/matlab/examples.html?s_tid=CRUX_topnav

Appendix

Code used for this lab assignment.

Part 2:

- coneThreshold.m - Function to apply color thresholding to images and detect the cones.

```
% function [BW,maskedRGBImage] = coneThreshold(RGB)
% %createMask  Threshold RGB image using auto-generated code from
colorThresholder app.
% %  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
% %  auto-generated code from the colorThresholder app. The colorspace
and
% %  range for each channel of the colorspace were set within the app.
The
% %  segmentation mask is returned in BW, and a composite of the mask and
% %  original RGB images is returned in maskedRGBImage.
%
% % Auto-generated by colorThresholder app on 14-Nov-2022
% %-------------------------------------------------------
%
%
% % Convert RGB image to chosen color space
% I = rgb2hsv(RGB);
%
% % Define thresholds for channel 1 based on histogram settings
% channel1Min = 0.031;
% channel1Max = 0.129;
%
% % Define thresholds for channel 2 based on histogram settings
% channel2Min = 0.174;
% channel2Max = 1.000;
%
% % Define thresholds for channel 3 based on histogram settings
% channel3Min = 0.956;
% channel3Max = 1.000;
%
% % Create mask based on chosen histogram thresholds
% sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
%     (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
%     (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
% BW = sliderBW;
%
% % Initialize output masked image based on input image.
% maskedRGBImage = RGB;
%
% % Set background pixels where BW is false to zero.
% maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
%
% end
```

Part 3:

- blobAnalysis.m - Identifying cone areas and centroids of these areas from the color detected images.

```
% function [areasP, centroidsP] = blobAnalysis(imageFile)
%
% % Read image into MATLAB
% CD = imread(imageFile);
% figure(1)
% imshow(CD)
% title('Original BW')
%
% threshold = 700;
%
% % Open image by removing connected pixel regions less than 200 pixels
in
% % size
% CDfiltered = bwareaopen(CD, threshold);
% figure(2)
% imshow(CDfiltered)
% title('Opened')
% %%
% % Fill image
% CDfilled = imfill(CDfiltered, 'holes');
% figure(3)
% imshow(CDfilled)
% title('Opened + Filled')
% %%
% % Label connected components
% L = bwlabel(CDfilled);
%
% % Calculate region properties for connected components
% s = regionprops(L);
%
% % Concatenate an array of all the region's 'area' values
% areas = cat(1, s.Area);
%
% % Concatenate an array of all the region's 'centroid' values
% centroids = cat(1, s.Centroid);
%
% areasP = [];
% centroidsP = [];
%
% BW2 = zeros(size(L));
% threshold = 7000;
%
% while ~isempty(centroids)
%     max_area = max(areas);
%     idx = find(areas == max_area);
%     centX = centroids(idx, 1);
%     centY = centroids(idx, 2);
%     if max_area > threshold
%         BW2 = BW2 + ismember(L, idx);
%         areasP = [areasP max_area];
%         centroidsP = [centroidsP; centX centY];
%         areas(idx) = 0;
%     else
%         break;
%     end
% end
%
% % Plot the image of the largest connected region
% figure(4)
% imshow(BW2)
% hold on
%
% l = size(centroidsP);
%
% for i=1:l(1)
%     x = centroidsP(i, 1);
%     y = centroidsP(i, 2);
%     plot(x, y, 'b*');
% end
%
% hold off
%
% end
```

- get_3b.m

```
% function [angles, proximities] = get_3b(imgBW, centroids, areas)
%     %input: image with centroid(s) and BW image
%     %orientation of the centroid(s) compared with the robot's FOV
%
%     %dimension:
%         %imgBW = MxN
%         %centroids = axa (square)
%         %areas = bx1 (vector)
%
%     %check if areas is empty
%     if isempty(areas)
%         angles = [];
%         proximities = [];
%         return
%     end
%
%     %angle reference(degree)
%     refAngle = zeros(length(centroids(:,1)),1);
%     %proximity reference
%     maxThreshold = 43000;
%     minThreshold = 8500;
%     refProx = maxThreshold - minThreshold;
%
%     %get number of rows and columns of the image
%     [nr, nc] = size(imgBW);
%     %image center X
%     imageCenterX = nc/2;
%     %compare x distance with the image center x
%     xDiff = centroids(:,1) - imageCenterX;
%
%     %angles(output)
%     angles = zeros(size(refAngle));
%     proximities = zeros(size(refAngle));
%
%     %find orientation
%     for idx = 1:length(refAngle)
%         if xDiff(idx) <= 0
%             %refAngle = +30
%             refAngle(idx) = 30;
%             %angle(idx) in radian
%             angles(idx) =
asin(sin(deg2rad(refAngle(idx)))*(abs(xDiff(idx))/imageCenterX));
%             %angle(idx) in degree
%             angles(idx) = rad2deg(angles(idx));
%         else
%             %refAngle = -30
%             refAngle(idx) = -30;
%             %angle(idx) in radian
%             angles(idx) =
asin(sin(deg2rad(refAngle(idx)))*(abs(xDiff(idx))/imageCenterX));
%             %angle(idx) in degree
%             angles(idx) = rad2deg(angles(idx));
%         end
%     end
%     angles = angles';
%
%     %find proximity
%     refProx = refProx/3; %3 proximities: 1(near) 2(middle) 3(far)
%     for idx = 1:length(refAngle)
%         if areas(idx) <= 1*refProx %far
%             proximities(idx) = 3;
%         elseif areas(idx) > 1*refProx && areas(idx) <= 2*refProx
%(middle)
%             proximities(idx) = 2;
%         else
%             proximities(idx) = 1; %near
%         end
%     end
%     proximities = proximities';
%
% end
```

- ME598_Grp3_Lab03_Part3.m - Function to apply blobAnalysis and target detection and overhead view functions for Part 3.

```
% function ME598_GrpR3_Lab03_Part3(imageName)
%
% [areas, centroids] = blobAnalysis(imageName);
% image = imread(imageName);
% [angles, proximities] = get_3b(image, centroids, areas);
% overheadView(angles, proximities, 5);
%
% end
```

- ImageExporting.m (For part 2) - Exporting side by side comparison of original and color detected images.

```
% % Left / right
% for iter = 2: size(allresults, 2)
% figure
% subplot(1,2,1);
% imshow(allresults(iter).fileName);
% title ('Original')
% subplot(1,2,2);
% imshow(allresults(iter).output)
% title('Processed')
% end
```

- ImageExporting.m (For parts 3, 4 & 5) - Exporting side by side comparison of original and overhead view of scenes captured.

```
% figure
% subplot(1, 2, 1);
% org = imread('TestConeImagesNew/TestImage7.jpg');
% % org = imread('ME598_ImgProcLab_MATLAB/TestConeImages/ConesN7.jpg');
% % org = imread('TestImageNoCones1.jpg');
% imshow(org);
% title('Original');
% subplot(1, 2, 2);
% over = imread('TestConeImagesNewOutput/TestImage_Overhead.jpg');
% % over = imread('TestImageNoCones1_Overhead.jpg');
% imshow(over);
% title('Overhead View');
```

- captureImage.m - Function to capture snapshots of cones for in lab exercise of Part 5.

```
% function captureImage(name)
% mycam = webcam;
% snap = snapshot(mycam);
% figure(1);
% imshow(snap);
% saveas(gcf, name, 'jpg');
% delete(mycam);
% end
```