Robotics Lab 3
Group: Afsah,Ailiya, Maleeha

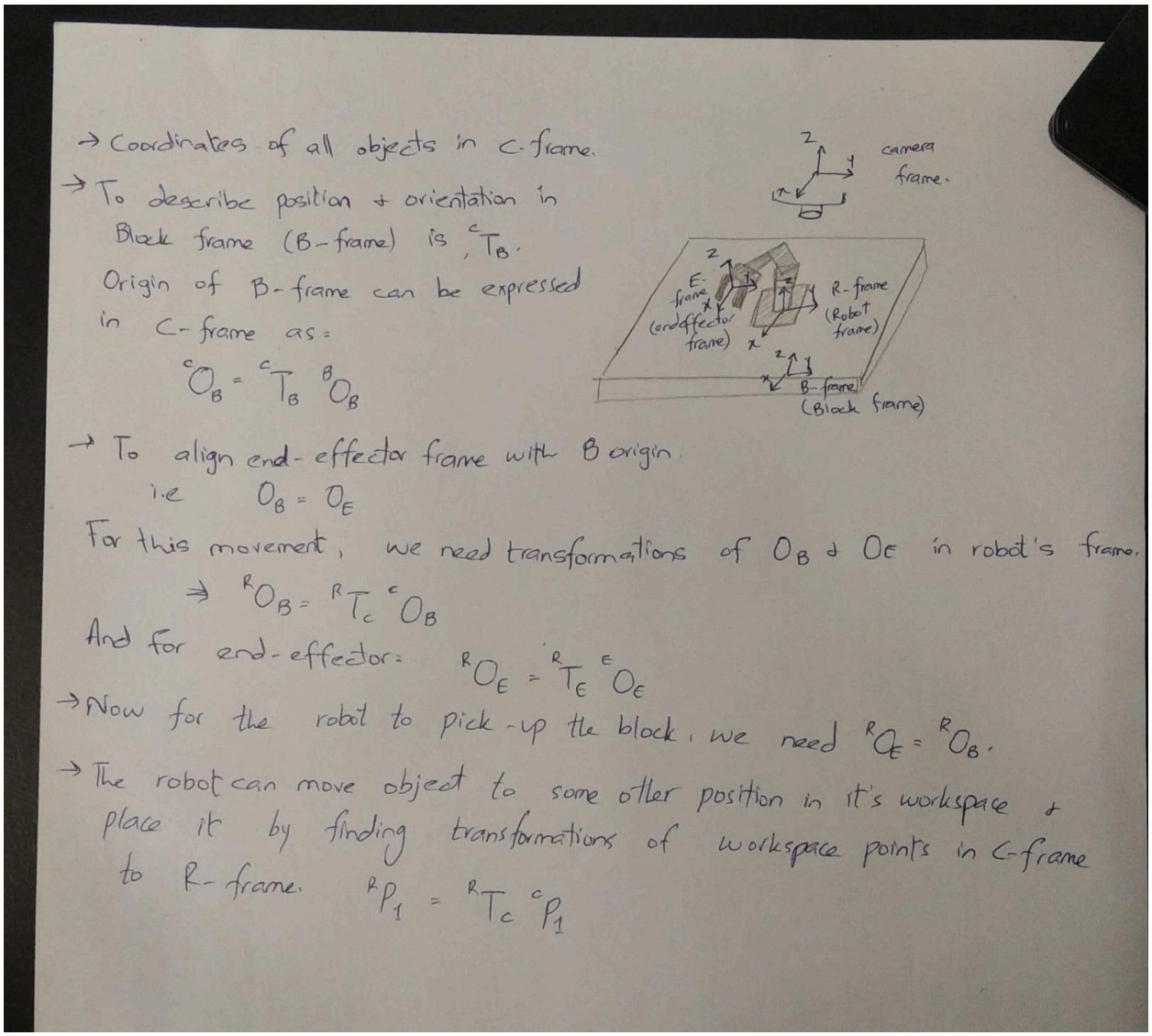
Task 3.1

Task 3.1 Frame Assignment (30 points)

Given the functional blocks of Figure 3.1,

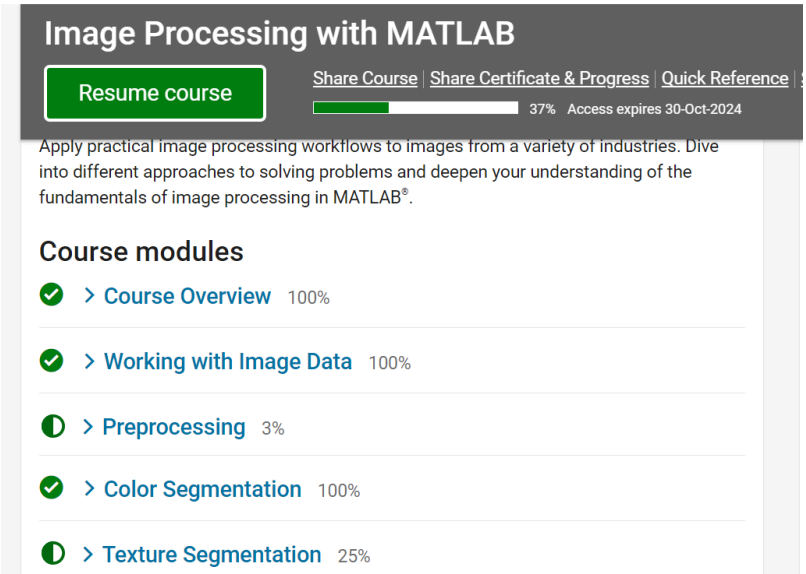
(a) assign frames to rigid bodies of interest in Figure 3.2.

(b) describe the pick and place task in terms of frame descriptions, e.g. determine ${}^{A}T_{B}$, the description of frame B with respect to frame A.

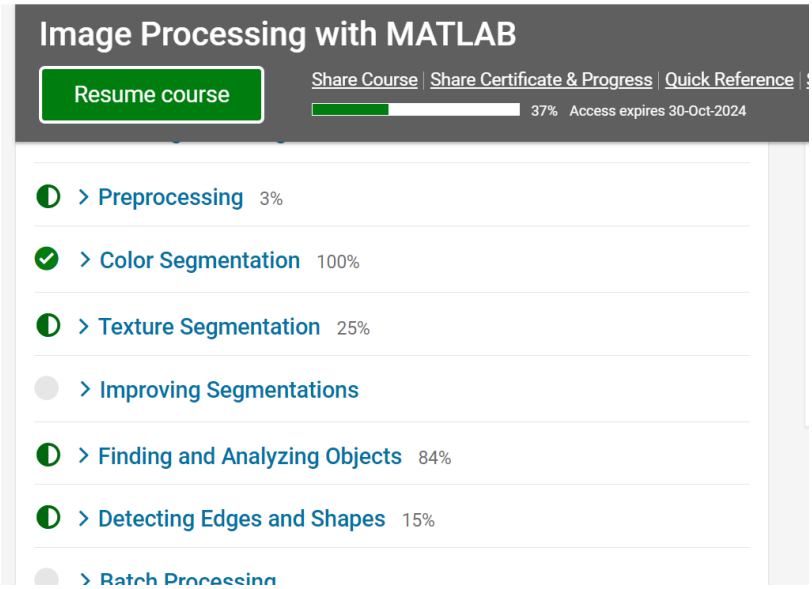


Task 3.2 **Getting to know the camera (0 points)**

Task 3.3 **Image Manipulation in MATLAB (10 points)**



Task 3.4, 3.5, 3.6



Task 3.7 **Find colored objects (60 points)**

Task 3.7 Find colored objects (60 points)

Set up cubes of various colors in the workspace of the robot. It can be assumed that the cubes are of a known size and the workspace is relatively uncluttered.

Tune, determine, and note down the camera parameters, presets, and filters in the RealSense Viewer that will result in an aptly exposed image and accurate depth values. Use [5] and [6] as reference. A MATLAB function `depth_example` is provided on LMS. Use it to obtain a color and a depth image of your setting[a]. The file provides an example of setting the camera parameters in code before acquisition.

Develop an object detection algorithm for determining the pixels corresponding to each object in the workspace. You are required to submit

1. a note describing your object detection strategy and the rationale behind it;
2. a list of the post-processing steps and parameters;
3. aptly commented code for your algorithm;
4. images at various steps of a good test run;
5. statistics across multiple trials, including
   - number of objects of each color in the workspace to be detected
   - number of objects of each color correctly detected
   - accuracy rate for each color.

[a] You will have to add the RealSense library to your MATLAB path. You can do this by navigating to 'Set Path' in the 'Home' ribbon on MATLAB and 'Add with subfolders' the library directory `C:/Program Files (x86)/Intel RealSense SDK 2.0/matlab/`.

1. **Object Detection Strategy:**

Approach: We will use a combination of color segmentation and blob analysis to detect the cubes.
Color segmentation: We'll define color ranges (Lab space) for each cube color to isolate pixels belonging to those objects.
Blob analysis: We'll identify connected regions of segmented pixels (blobs) corresponding to individual cubes.
Rationale: This approach is efficient for well-defined color objects in a controlled environment.

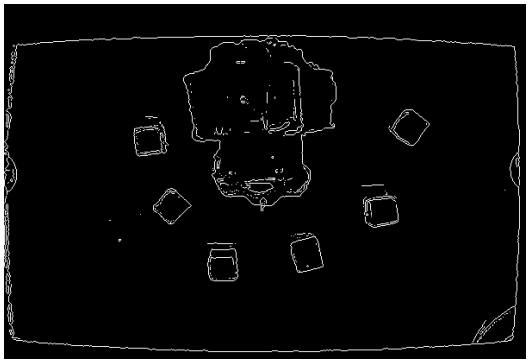
2. **Post-processing Steps and Parameters:**

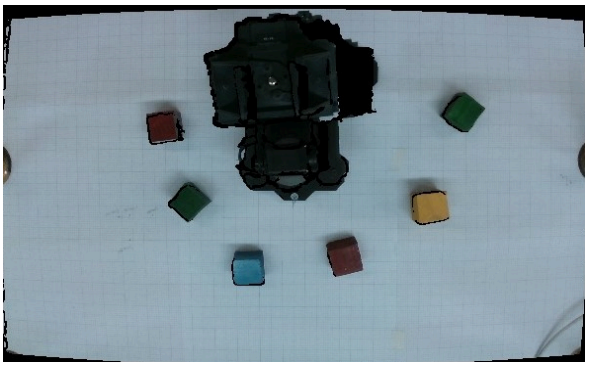
Filtering: Apply morphological operations (e.g., opening, closing) to remove noise and smooth boundaries.
Minimum area threshold: Filter out small blobs that might be noise or artifacts.
Parameter tuning: Adjust thresholds and filter sizes based on observed behavior during testing.

Commented code, image and statistics: We used the built-in color thresholder, and we fine-tuned the code for accurate detection.

Detecting edges:

```
img=imread('C:\Users\computer house\Downloads\robo2.jpg');
bnb=rgb2gray(img)
[defaultEdges,thresh]=edge(bnb)
adjEdges=edge(bnb,thresh/2)
imshow(adjEdges)
lb=bwlabel()
```





Color threshold:
This is an example for green color detection. Similar for other colors.

```
function [green_BW,maskedRGBImage] = create_green_Mask(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 17-Feb-2024
%------------------------------------------------------
% Convert RGB image to chosen color space
I = rgb2lab(RGB);
% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.000;
channel1Max = 79.229;
% Define thresholds for channel 2 based on histogram settings
channel2Min = -23.075;
channel2Max = -7.734;
% Define thresholds for channel 3 based on histogram settings
channel3Min = -3.058;
channel3Max = 31.475;
```

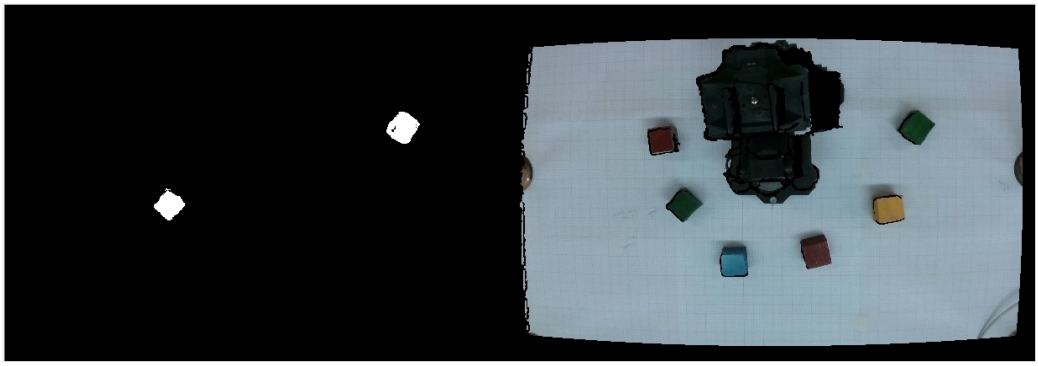
```
% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
green_BW = sliderBW;

% restrict the area to a certain range
green_BW=bwpropfilt(green_BW,"Area",[100 2500]);
% Initialize output masked image based on input image.
maskedRGBImage = RGB;
% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~green_BW,[1 1 3])) = 0;
end
```

This code targets the detection of green color in images. Instead of using RGB, it converts the images to LAB color space. LAB is known to be more intuitive for color manipulation, making green detection easier and more precise. The specific thresholds used to identify green pixels were chosen using a "color thresholder app." These thresholds define the range of acceptable LAB values that represent green in the image. To avoid picking up unwanted green hues like background shades, an additional condition based on area is applied. This ensures that only sufficiently large green regions are detected, potentially representing actual green "blocks" in the image.

Result:

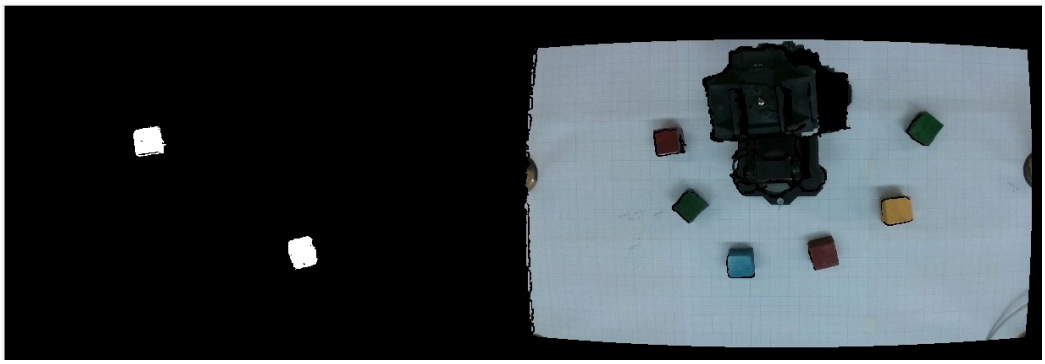
For green color



No. of green objects to be detected: 2

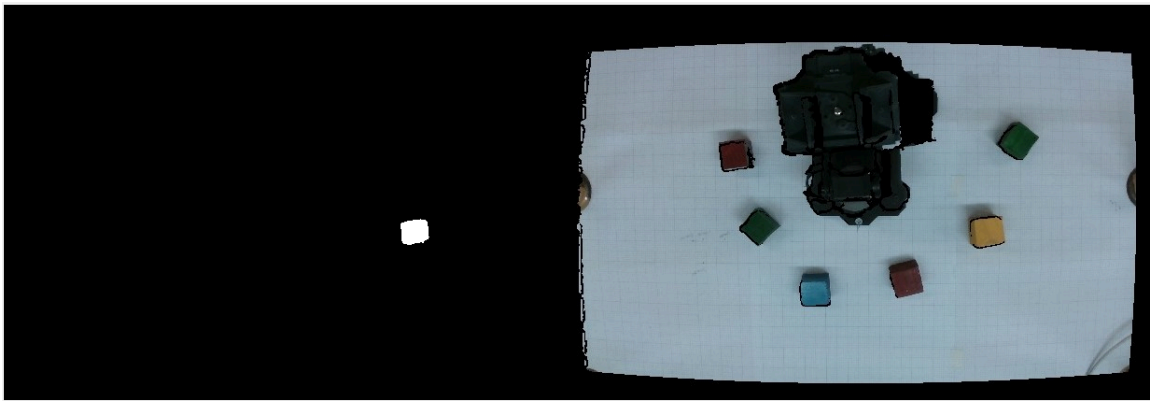
No. of green objects correctly detected: 2

For red color:



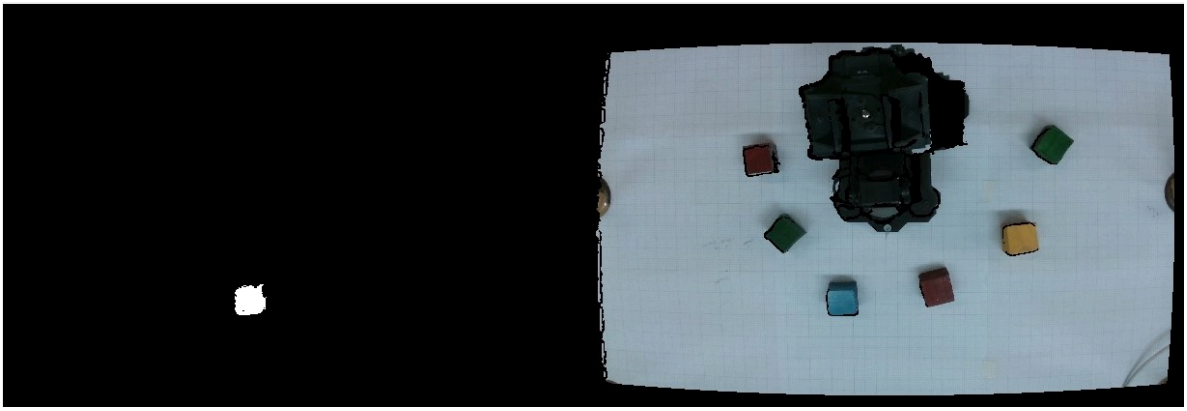
No. of red objects to be detected: 2
No. of red objects correctly detected: 2

For yellow color:



No. of yellow objects to be detected: 1
No. of yellow objects correctly detected: 1

For blue color:



No. of blue objects to be detected: 1
No. of blue objects correctly detected: 1

Task 3.8 Determining geometric features (40 points)

Think about which geometric features could assist you in associating a pose with the object, e.g. you could detect the planes of the object, or the edges, or the corners, etc. You are required to submit

1. your assignment of a frame to an object;
2. description of an algorithm for constructing the pose homogeneour transformation (how do you determine the transformation from your identified geometric features?);
3. description of an algorithm for determining the required geometric features (determining the pixel coordinates of relevant pixels of the blob);
4. aptly commented code for the previous part;
5. images at various steps of a good test run.

After detecting the objects(cubes) present in the image based on color, we could also isolate objects of a single color.
To assign a pose, we need to define the position and orientation of the object with respect to the base frame. One way to do this is to identify first the edges, and then using the edges, we find the corners of the cube.
This method works specifically due the geometry of the cube. Once all corners are identified, we set one of the corners as origin.
We follow these steps in our case:

1. Left - bottom edge is the origin
2. Vector from left bottom to right bottom is the x axis
3. Vector from left bottom to left top is the x axis

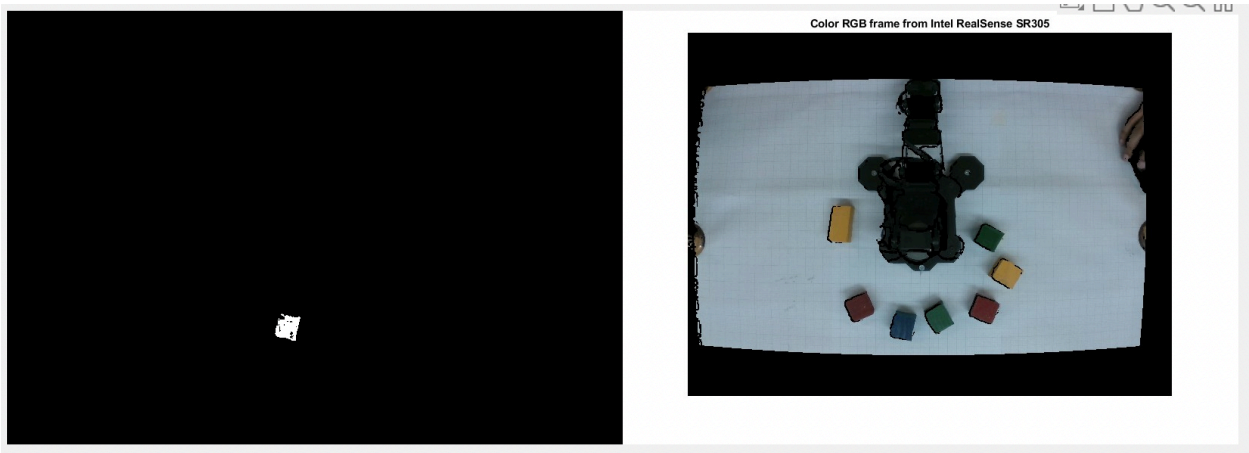
Following is the commented code and output image: (This is for blue color, the same is done for other colors)

```
close all
img=imread('/Users/maleehahussain/Desktop/HU maleeha/Sem
6/robo labs/rgb2.jpeg');
% colorThresholder(img)
% maskg=create_green_Mask(img)
% imshowpair(maskg,img,"montage")
% figure
```

```
maskb= create_blue_Mask(img);
imshowpair(maskb,img,"montage");
% Apply dilation to fill small gaps
se = strel('disk', 5); % Adjust the disk size as needed
maskb_filled = imdilate(maskb, se);
figure
imshow(maskb_filled)
% to detect the edges
% detect edges in the masked color
% gray = rgb2gray(maskb)
figure
lines= edge(maskb_filled,'Canny');
imshowpair(lines,img,"montage");
figure;
imshow(img);
% hold on;
% plot(squareCorners(:,1), squareCorners(:,2), 'r-',
'LineWidth', 2);
% hold off;
figure;
corners = detectHarrisFeatures(lines,'MinQuality',
0.4,'FilterSize', 9)
imshow(lines); hold on;
plot(corners.selectStrongest(4));
corner_matrix= corners.selectStrongest(4)
cube_corners= corner_matrix.Location
% Find the minimum and maximum x and y coordinates
min_x = min(cube_corners(:, 1));
max_x = max(cube_corners(:, 1));
min_y = min(cube_corners(:, 2));
max_y = max(cube_corners(:, 2));
% Determine the corners based on the minimum and maximum
coordinates
bottom_left = [min_x, min_y]
bottom_right = [max_x, min_y]
top_left = [min_x, max_y]
top_right = [max_x, max_y]
% yellow_mask=bwpropfilt(yellow_BW,"Area",[100 2500]);
% B=imoverlay(img,yellow_mask,"yellow");
% % imshow(B)
%
```

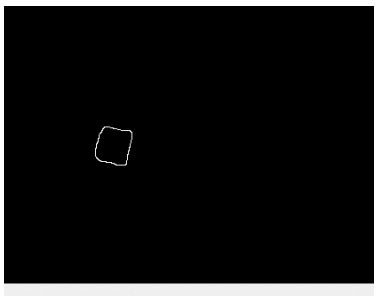
```
% green_mask=bwpropfilt(green_BW,"Area",[100 2500]);
% red_mask=bwpropfilt(red_BW,"Area",[100 2500]);
% blue_mask=bwpropfilt(blue_BW,"Area",[100 2500]);
% %
% imshowpair(blue_BW,blue_mask,"montage")
```

1.Initial image, color segmented to isolate blue block

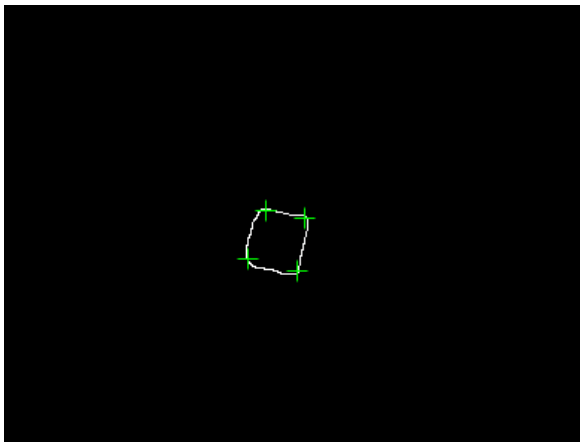


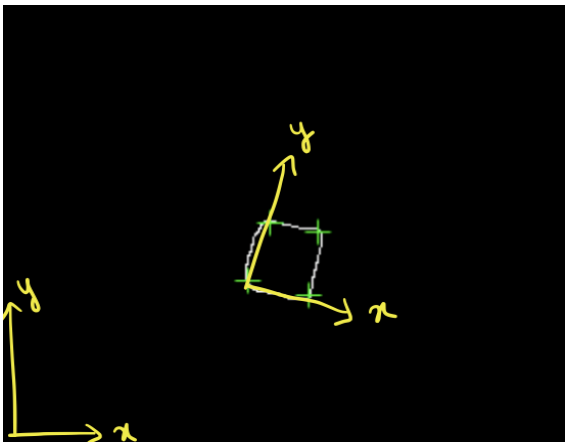
2. Fill in the mask to make the body smoother and easier to detect edges

3. Edge detection

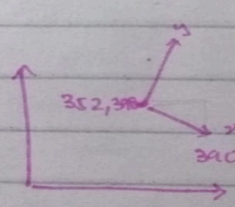


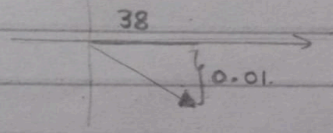
4. Corner detection (using trial and error to find the best fitting parameters for most accurate corners)





angle of new x axis wrt to previous:





$\tan^{-1}\left(\frac{0.01}{38}\right) = -0.015°$

Pose is given by the following T matrix. (rotation about z + translation)

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & \text{bottom left } x \\ \sin\theta & \cos\theta & 0 & \text{BL } y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
cube_corners =

  4×2 single matrix

  384.6898  437.6746
  363.7133  397.6967
  389.7588  403.2412
  351.7201  429.9793


bottom_left =

  1×2 single row vector

  351.7201  397.6967


bottom_right =

  1×2 single row vector

  389.7588  397.6967


top_left =

  1×2 single row vector

  351.7201  437.6746


top_right =

  1×2 single row vector

  389.7588  437.6746
```