Introduction To Robotics

Lab 03 - Task 3.7 and 3.8

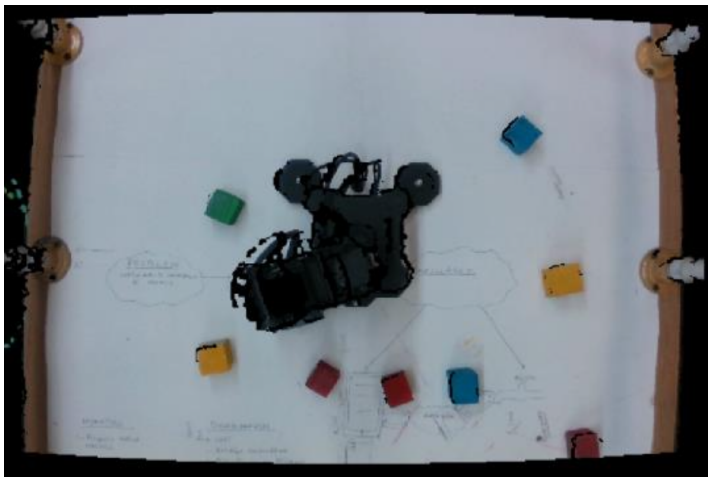
Huzaifah Tariq Ahmed - ha07151
Syed Asghar Abbas Zaidi - sz07201
Daniyal Rahim Areshia - d07605

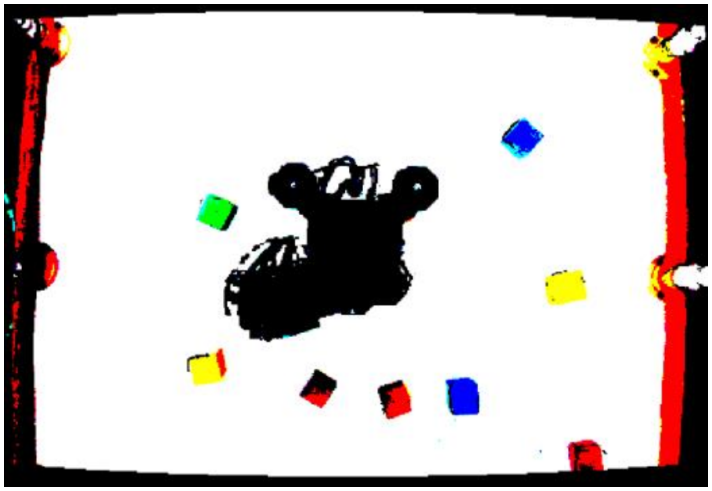
10th May 2024

1 Task 3.7

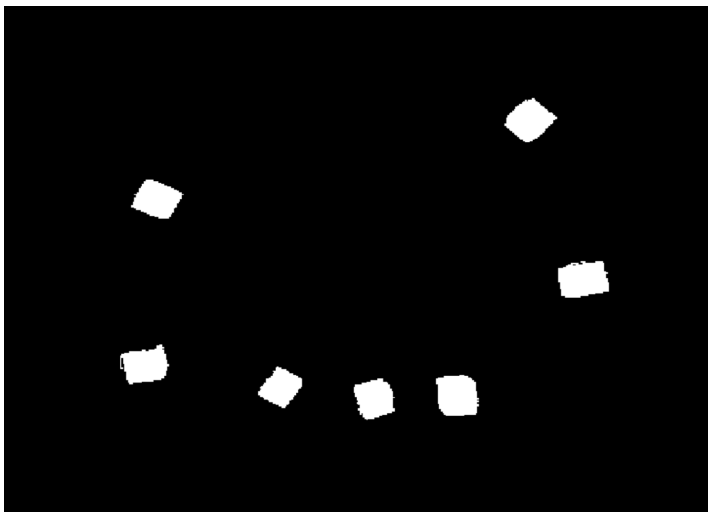
1.1 Part 1,2 and 4 - Steps and Images

Firstly we binarized the RGB image, producing the depicted figure. This process generated RGB vectors comprising only 1s and 0s, simplifying subsequent processing compared to the previous state where RGB values were combined with pixel values ranging from 0 to 255.

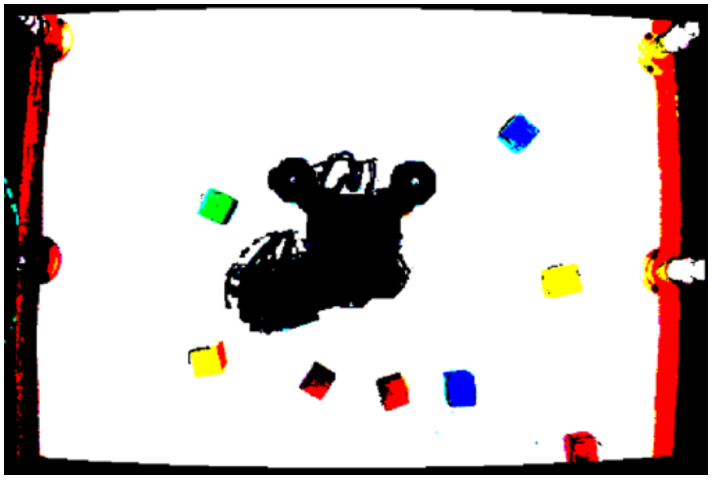




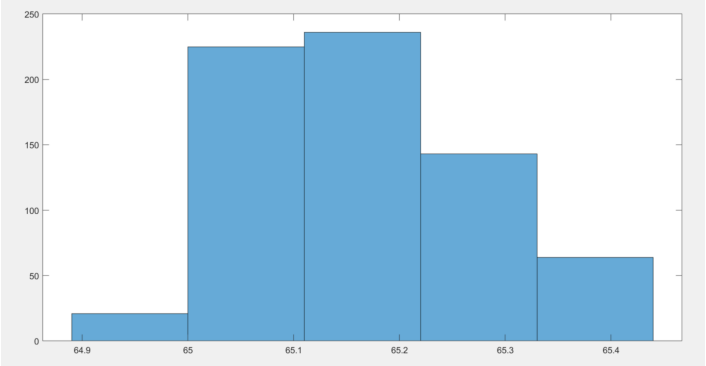
Secondly, we inverted the colors of the binarized image to create a new image called ′tmp img′. This new image had inverted colors, where RGB composed objects appeared white and everything else appeared black, facilitating cube identification. Additionally, since there was an arm in the center, it was necessary to remove it. This was achieved by analyzing pixel counts and eliminating objects with pixel counts less than 500 or greater than 800 from the ′tmp img′. These steps resulted in the final outcome.



The next step involved identifying the color of the blocks. This required examining the identified blocks and determining their respective RGB values. Our main objective was to establish the threshold value to categorize the objects as either "Red," "Green," or "Blue." The threshold values for Mean Pixel Values of RGB were as follows: Red > 65, Green > 65, Blue > 67. The red cube in the bottom right corner is partially cut off because it is connected to the background, causing its pixel count to surpass the threshold. To prevent this, the blocks should be positioned within the reachable workspace.



Next, our process involves identifying the upper faces of the cubes to determine their center points using centroids. To ascertain the maximum depth from the camera to these upper faces and exclude points where depth equals 0 cm, we employed a 'depth code' to create a mat image. This mat image was then employed to eliminate points with a depth of 0, and the resulting heights were plotted on a histogram to analyze depth trends. The histogram is depicted below:



Afterwards, we identified the maximum value in the histogram. If there are heights exceeding this maximum index, we set the upper limit to be 2 indices higher than the maximum value. In cases where there is no histogram range beyond the maximum index, we set the upper limit of the threshold to be the next index after the maximum of the histogram. Subsequently, we marked the top faces of the cubes as 1, while assigning 0 or black to the other pixels. This process allowed us to detect the top faces of the cubes effectively. Upon completion of a successful iteration, we obtained the following result:

1.2 Part 3 - Aptly Commented Code

1.2.1 Create all objects to be used in this file

```
% Initialize pipeline for RealSense streaming
pipe = realsense.pipeline();
colorizer = realsense.colorizer();
cfg = realsense.config();

% Configure pipeline for depth and color streams
cfg.enable_stream(realsense.stream('depth'), realsense.format('Distance'));
cfg.enable_stream(realsense.stream('color'), realsense.format('rgb8'));

% Start streaming with specified configuration
profile = pipe.start();

% Acquire device parameters and set sensor options
dev = profile.get_device();
depth_sensor = dev.first('depth_sensor');
rgb_sensor = dev.first('roi_sensor');
depth_scaling = depth_sensor.get_depth_scale();
rgb_sensor.set_option(realsense.option('enable_auto_exposure'), 1);
rgb_sensor.set_option(realsense.option('enable_auto_white_balance'), 1);

% Align color and depth frames
for i = 1:5
    fs = pipe.wait_for_frames();
end
align_to_depth = realsense.align(realsense.stream.depth);
fs = align_to_depth.process(fs);
```

```
pipe.stop();

% Depth post-processing (spatial and temporal filtering)
depth = fs.get_depth_frame();
spatial = realsense.spatial_filter(.5, 20, 2);
depth_p = spatial.process(depth);
temporal = realsense.temporal_filter(.13, 20, 3);
depth_p = temporal.process(depth_p);

% Color frame processing
color = fs.get_color_frame();
depth_color = colorizer.colorize(depth_p);
data = depth_color.get_data();
img = permute(reshape(data',[3,depth_color.get_width(),depth_color.
   get_height()]),[3 2 1]);

% Display colorized depth frame
%imshow(img);

% Convert depth values to meters and display depth frame
data3 = depth_scaling * single(depth_p.get_data());
ig = permute(reshape(data3',[width,height]),[2 1]);
%imshow(mat2gray(ig));

% Display RGB frame
data2 = color.get_data();
im = permute(reshape(data2',[3,color.get_width(),color.get_height()])
   ,[3 2 1]);
```

1.2.2 Process Starts Here

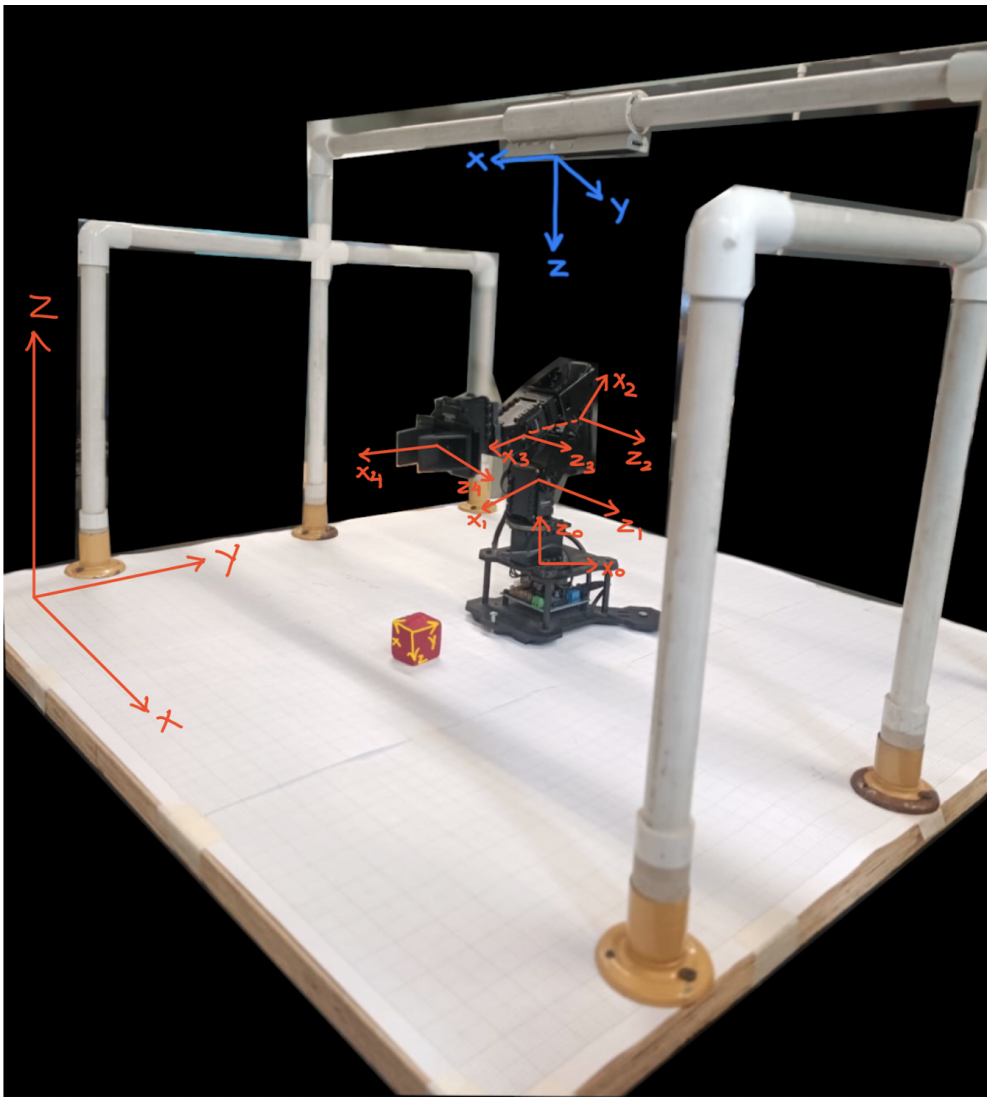
```
% Convert image to binary using imbinarize function
BW = imbinarize(im);

% Invert colors to isolate cubes (white) from background (black)
tmp_img = ~(BW(:,:,1) & BW(:,:,2) & BW(:,:,3));

% Connect components in the image to identify individual cubes
cc4 = bwconncomp(tmp_img,4);

% Remove small or large objects that are not cubes based on size
   thresholds
for k = 1:cc4.NumObjects
    siz_ = size(cc4.PixelIdxList{k});
    if siz_(1) < 500 || siz_(1) > 800
        tmp_img(cc4.PixelIdxList{k}) = 0;
```

```
    end
end

% Connect components again to identify cubes after removing unwanted
   shapes
cc4_ = bwconncomp(tmp_img,4);

% Initialize arrays to store color information for each cube
red_col = zeros(1,cc4_.NumObjects);
green_col = zeros(1,cc4_.NumObjects);
blue_col = zeros(1,cc4_.NumObjects);

% Extract RGB channels of the original image
red_im = im(:,:,1);
green_im = im(:,:,2);
blue_im = im(:,:,3);

% Determine color of each cube based on mean RGB values
for k = 1:cc4_.NumObjects
    mean_red_pix = mean(red_im(cc4_.PixelIdxList{k}));
    mean_green_pix = mean(green_im(cc4_.PixelIdxList{k}));
    mean_blue_pix = mean(blue_im(cc4_.PixelIdxList{k}));
    if mean_red_pix > 65
        red_col(k) = 1;
    end
    if mean_green_pix > 65
        green_col(k) = 1;
    end
    if mean_blue_pix > 67
        blue_col(k) = 1;
    end
end

% Display RGB color information for each cube
red_col
green_col
blue_col

% Load depth information and convert to centimeters
load ig.mat
new_ig = ig*100;

% Create a blank image to mark top faces of cubes
blank_img = tmp_img;
blank_img(:,:) = 0;

% Find maximum depth for each cube and mark top faces in blank_img
```

```
for k = 1:cc4_.NumObjects
    non_zero_pix = [];
    for j = 1 : length(cc4_.PixelIdxList{k})
        if new_ig(cc4_.PixelIdxList{k}(j)) ~= 0
            non_zero_pix(end + 1) = new_ig(cc4_.PixelIdxList{k}(j));
        end
    end
    non_zero_his = histogram(non_zero_pix,5);
    [maxx_pixels, idx] = max(non_zero_his.Values);
    if idx + 2 <= length(non_zero_his.BinEdges)
        max_threshold = non_zero_his.BinEdges(idx+2);
    else
        max_threshold = non_zero_his.BinEdges(idx+1);
    end
    for j = 1 : length(cc4_.PixelIdxList{k})
        if(new_ig(cc4_.PixelIdxList{k}(j)) <= max_threshold)
            blank_img(cc4_.PixelIdxList{k}(j)) = 1;
        end
    end
end

% Connect components in blank_img to find centroids of top faces
cc4_for_cen = bwconncomp(blank_img);
coords_centroids = zeros(2,cc4_for_cen.NumObjects);

% Calculate centroids and mark them on blank_img
for i = 1:cc4_for_cen.NumObjects
    col_ = mod(cc4_for_cen.PixelIdxList{i},480);
    row_ = ceil(cc4_for_cen.PixelIdxList{i}/480);
    row_fin = ceil(mean(row_));
    col_fin = ceil(mean(col_));
    coords_centroids([1,2],i) = [row_fin,col_fin];
    blank_img(row_fin,col_fin) = 0;
end

% Display centroids and save the results
coords_centroids'
save tmp1
```

2 Task 3.8

2.1 Part 1 - Assignment of a Frame of an Object



2.2 Part 2,3 - Algorithm

Our aim is to identify the top face as the primary geometric feature that indicates how the blocks are positioned. Initially, we create separate binary masks for each color of the blocks found in the workspace. These masks are tested by overlaying them onto the captured image to ensure that only the corresponding block is visible through the overlay. Once confirmed, we combine these binary masks to produce a comprehensive mask that accurately represents all the blocks in the workspace image, effectively masking out any other elements.

Once we have identified the blocks, we use depth data to isolate their top faces by removing other visible surfaces. This is possible because the top face typically has the lowest z-axis value when viewed from the top-mounted camera, especially given the relatively small size of the blocks. This isolation is based on a fixed z-value.

Following the isolation of the top faces, we draw a rectangle around each one, aligning its sides with

the edges of the top face and its corners with those of the top face. This step allows us to precisely determine the position of the top faces within the visible workspace and ascertain the orientation of the blocks relative to the x-axis. This information is crucial for determining both the position and orientation of the blocks in relation to any other frame, provided we have the necessary transformation data.

2.3 Part 4 - Aptly Commented Code

```
clear;

%% Initialization
pipe = realsense.pipeline();
colorizer = realsense.colorizer();
cfg = realsense.config();

%% Configure streaming
streamTypeDepth = realsense.stream('depth');
formatTypeDepth = realsense.format('Distance');
cfg.enable_stream(streamTypeDepth, formatTypeDepth);

streamTypeColor = realsense.stream('color');
formatTypeColor = realsense.format('rgb8');
cfg.enable_stream(streamTypeColor, formatTypeColor);

profile = pipe.start();

%% Set device parameters
dev = profile.get_device();
depthsensor = dev.first('depth_sensor');
rgbsensor = dev.first('roi_sensor');
depthscaling = depthsensor.get_depth_scale();

depthsensor.set_option(realsense.option('visual preset'), 9);
rgbsensor.set_option(realsense.option('enable auto exposure'), 1);
rgbsensor.set_option(realsense.option('enable auto white balance'), 0);

%% Align frames
aligntodepth = realsense.align(realsense.stream.depth);
fs = aligntodepth.process(pipe.wait_for_frames());

%% Post-processing
depth = fs.get_depth_frame();
spatial = realsense.spatial_filter(0.5, 20, 2);
depthp = spatial.process(depth);
temporal = realsense.temporal_filter(0.13, 20, 3);
depthp = temporal.process(depthp);

```

```
%% Colorize and display frames
color = fs.get_color_frame();
depthcolor = colorizer.colorize(depthp);
img = permute(reshape(depthcolor.get_data(), [3, depthcolor.get_width()
   , depthcolor.get_height()]), [3 2 1]);
imshow(img)

%% Block identification
r = img(:, :, 1);
g = img(:, :, 2);
b = img(:, :, 3);
redMask = r - g > 25 & r - b > 25;
blueMask = b - g > 32 & b - r > 90;
yellowMask = r - g > 30 & r - b > 80;
greenMask = abs(g - b) > 4 & g - r > 14;
finalMask = redMask + greenMask + blueMask + yellowMask;

depthdata = double(depthp.get_data()) * depthscaling;
depthdata = depthdata .* (finalMask > 0);

%% Display depth image
depthdata = reshape(depthdata, [depth.get_width(), depth.get_height()])
   ';
figure
imshow(depthdata)
depthdata = depthdata .* (finalMask > 0);

%% More processing and visualization steps...

%% Display final results
imshow(topcolor)
blocksCorners = regionprops("struct", Xmasknew, "Extrema");
blockCentroids = regionprops("struct", Xmasknew, "Centroid");
blockAngles = blocksOrientation.Orientation;
theta = -blockAngles(1);
blocksCornersRot = zeros(4, 2);
for i = 1:4
    coords = blocksCorners(i).Extrema - blockCentroids(i).Centroid;
    R = [cosd(theta), -sind(theta); sind(theta), cosd(theta)];
    temp = R * coords';
    temp = temp' + blockCentroids(i).Centroid;
    blocksCornersRot(:, :, i) = temp;
end
disp(blocksCornersRot)
```

2.4 Part 5 - Images At Various Steps of a Good Test Run

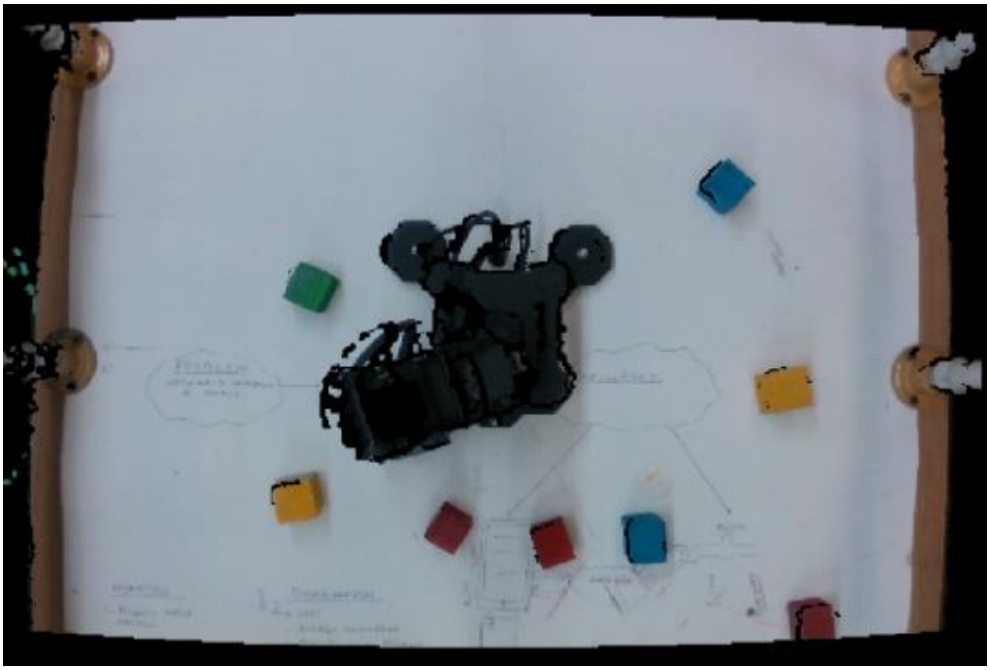


Figure 1: Original RGB Image from Camera

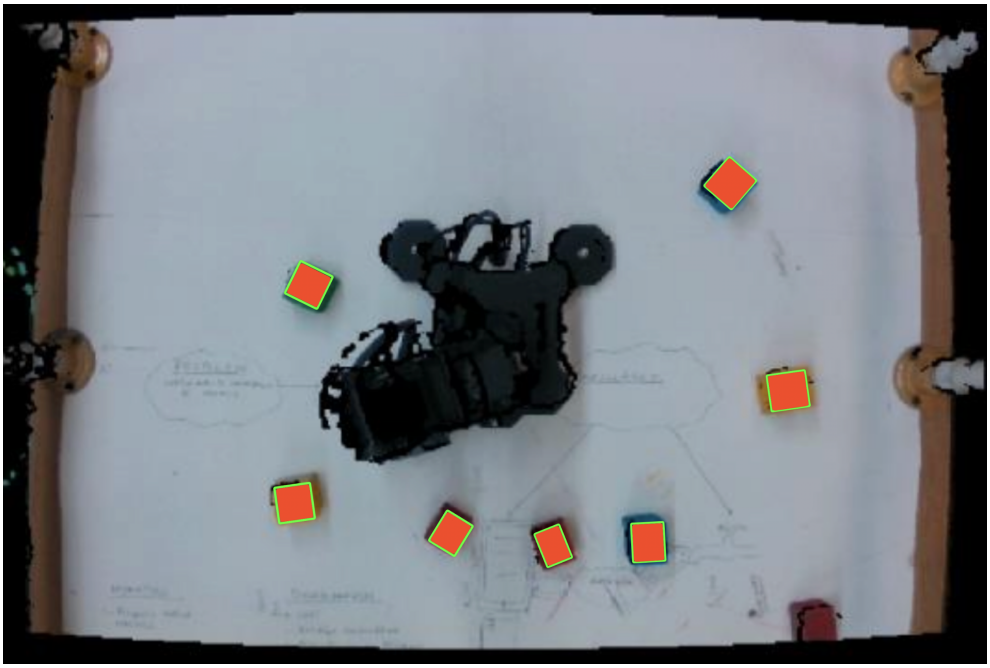


Figure 2: Image with blocks identified

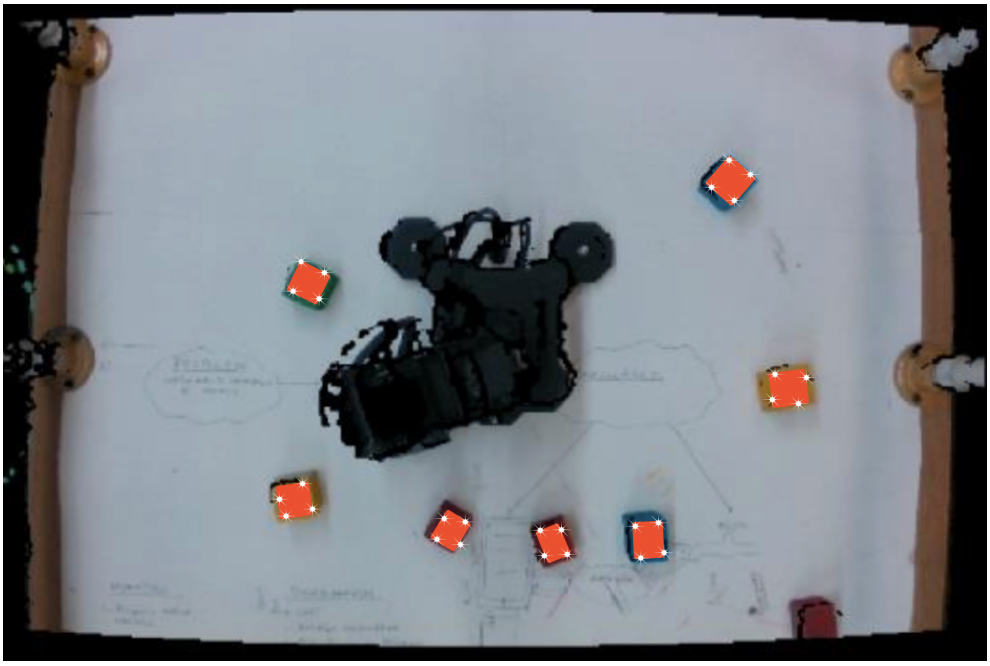


Figure 3: Image with top faces identified with sides and corners of each of the blocks

Github Repository Link