Introduction to Robotics
Lab 7 - Adding visual sensing to our system

Names: Hamad Abdul Razzaq & Muhammad Ali Siddiqui
IDs: hr06899 & ms06875

Lab Report

Task 7.1

The provided code was studied and explored.

Task 7.2

The provided code was studied and explored.

Task 7.3

Object Detection Strategy & Post-Processing Steps

The given information to us is an RGB image containing the color information and a depth image containing the depth information. Using these two images, we will formulate our object detection strategy. It is as follows:

- We first differentiate all the objects in the scene with the background by assigning them different colors. For example, assigning all the objects a white color and assigning black to the background.
- We store the information of pixel indices corresponding to each object in the screen separately.
- We remove those objects that are not cubes by looking at the pixel count.
- We extract those pixels of the cube which make up its top face and then determine the center pixel of the object. We also store the depth value for each object using depth image
- We then find out the average and maximum red, green and blue values corresponding to each object's top face using the RGB image and then classify whether the object contains the red, green or blue color based on some threshold for average and maximum red, green and blue values.

Final Parameters of interest

- R, G and B values corresponding to each object of interest in the scene.
- Depth Values of each object of interest in the scene.
- Center pixel of each object of interest in the scene.

Code for Vision Pipeline

After getting the RGB image and depth data from the provided `depth_example.m` code, we run the following code:

```
BW = imbinarize(im); % Binarizing the given rgb image
figure; % Displaying Binarized Image along with original rgb image
imshowpair(im,BW,'montage');
title('RGB Image & its binarized version');
```

```
len = size(BW); % Size of image
BW2 = ~(BW(:,:,1) & BW(:,:,2) & BW(:,:,3)); % This expression is used to assign the color ...
   black to the background that white part in the rgb image. And it assigns white to the ...
   non-white part in the rgb image. This is done to differentiate the background with the ...
   objects in the scene.
BW3 = BW2;
figure; % Displaying the obtained black and white image that differentiates background from ...
   objects
imshow(BW3);
title('Inverted Binarized Image');
cc8 = bwconncomp(BW3); % This function finds all the connected components, i.e., objects in ...
   the scene and stores the pixel indices for each object
remove_idx = []; % This array will contain the pixel indices of those objects which are not cube
% Filtering out stuff other than cubes
for i = 1:length(cc8.PixelIdxList)
    if (length(cc8.PixelIdxList{i}) < 200) || (length(cc8.PixelIdxList{i}) > 2000) % For any ...
       object containing less than 200 pixels or more than 2000 pixels, they cannot be ...
       cubes. So indices of those pixels are added to remove_idx
        remove_idx(end + 1) = i;
    end
end
for i=1:length(remove_idx)
    BW2(cc8.PixelIdxList{remove_idx(i)}) = 0; % Setting the value of the pixel indices ...
       corresponding to non-cube objects to 0, i.e. turning them black which is equivalent ...
       to removing them.
end
figure;
imshowpair(BW3, BW2, 'montage'); % Displaying Black & White image after removing the ...
   non-cube objects
title('Before & After');
cc8 = bwconncomp(BW2); % Finding the connected components in the new image so that we only ...
   have the information of cubes stored
BW4 = BW2;
object_info = struct; % Making a structure that contains all the information from our image ...
   processing
object_info.Index_data = cc8.PixelIdxList; % Object Indices
object_info.rgb_image = im2double(im); % RGB Image
object_info.depth_info_cm = ig*100; % Depth Info in cm
object_info.red_val = zeros(1,length(object_info.Index_data)); % Avg Red for each obj
object_info.green_val = zeros(1,length(object_info.Index_data)); % Avg Green for each obj
object_info.blue_val = zeros(1,length(object_info.Index_data)); % Avg Blue for each obj
object_info.red_val_max = zeros(1,length(object_info.Index_data)); % Max Red for each obj
object_info.green_val_max = zeros(1,length(object_info.Index_data)); % Max Green for each obj
object_info.blue_val_max = zeros(1,length(object_info.Index_data)); % Max Blue for each obj
object_info.top_data = object_info.Index_data; % Indices for top faces of each object
object_info.top_data_width = object_info.Index_data; % x Indices for top faces of each object
object_info.top_data_column = object_info.Index_data; % y Indices for top faces of each object
object_info.no_top_data = object_info.Index_data; % This will store all the indices of the ...
   object that is not a top face pixel
object_info.center_x = zeros(1, length(object_info.Index_data)); % x value of the center ...
   pixel of each object
object_info.center_y = zeros(1, length(object_info.Index_data)); % y value of the center ...
   pixel of each object
object_info.red_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array telling ...
   whether each object contains red color or not
object_info.green_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array ...
   telling whether each object contains green color or not
```

```
object_info.blue_val_bin = zeros(1,length(object_info.Index_data)); % A Binary array telling ...
    whether each object contains blue color or not
object_info.depth_val = zeros(1,length(object_info.Index_data)); % Depth value for each ...
    object in cm
for i = 1:length(object_info.Index_data) % Looping over each object
    arr = []; % This array contains those pixels of an object whose depth value is non-zero
    for j = 1:length(object_info.Index_data{i})
        if object_info.depth_info_cm(object_info.Index_data{i}(j)) ~= 0
            arr(end +1) = object_info.depth_info_cm(object_info.Index_data{i}(j));
        end
    end
    counts = hist(arr, 10); % A Histogram is generated on the pixels having non-zero depth ...
        values
    th = min(arr) + (otsuthresh(counts)*(max(arr) - min(arr))); % Determining threshold for ...
        top face detection using otsu algorithm
    object_info.depth_val(i) = th;
    r = 0;
    l = 0;
    for j = 1:length(object_info.Index_data{i})

        if object_info.depth_info_cm(object_info.Index_data{i}(j)) > th % Separating top ...
            face indices using this condition
            BW4(object_info.Index_data{i}(j)) = 0;
            object_info.top_data{i}(j-r) = [];
            r = r + 1;
        else
            object_info.no_top_data{i}(j-l) = [];
            l = l + 1;
        end
    end
end

for i = 1:length(object_info.top_data)
    object_info.top_data_column{i} = floor(object_info.top_data{i} ./ len(1)) + 1;
    object_info.top_data_width{i} = mod(object_info.top_data{i}, len(1));
    object_info.center_x(i) = floor(mean(object_info.top_data_width{i})); % Calculating x ...
        value of Center pixel
    object_info.center_y(i) = floor(mean(object_info.top_data_column{i}));% Calculating y ...
        value of Center pixel
    BW4(object_info.center_x(i), object_info.center_y(i)) = 0;  % Setting color for center ...
        value to be black so that it is visible
end
figure;
imshowpair(BW2, BW4, 'montage'); % Showing the objects with both top face and non-top face ...
    pixels and objects with only top face pixels
title('Face top detection');
avg_th = 0.3; % Threshold for average RGB value
max_th = 0.4; % Threshold for maximum RGB value
for k = 1:length(object_info.top_data_width) % Looping for every object
    % Finding the average and maximum values
    val_r = 0;
    val_g = 0;
    val_b = 0;
    max_r = -inf;
    max_g = -inf;
    max_b = -inf;
    for i = 1:length(object_info.top_data_width{k})
```

```
            if (object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 1) > max_r)
                max_r = object_info.rgb_image(object_info.top_data_width{k}(i), ...
                    object_info.top_data_column{k}(i), 1);
            end
            if (object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 2) > max_g)
                max_g = object_info.rgb_image(object_info.top_data_width{k}(i), ...
                    object_info.top_data_column{k}(i), 2);
            end
            if (object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 3) > max_b)
                max_b = object_info.rgb_image(object_info.top_data_width{k}(i), ...
                    object_info.top_data_column{k}(i), 3);
            end
            val_r = val_r + object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 1);
            val_g = val_g + object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 2);
            val_b = val_b + object_info.rgb_image(object_info.top_data_width{k}(i), ...
                object_info.top_data_column{k}(i), 3);
    end
    object_info.red_val(k) = val_r / length(object_info.top_data_width{k});
    object_info.green_val(k) = val_g / length(object_info.top_data_width{k});
    object_info.blue_val(k)= val_b / length(object_info.top_data_width{k});
    object_info.red_val_max(k) = max_r;
    object_info.green_val_max(k) = max_g;
    object_info.blue_val_max(k) = max_b;
    % Setting RGB values if the required thresholds are met
    if object_info.red_val(k) >= avg_th && object_info.red_val_max(k) > max_th
        object_info.red_val_bin(k) = 1;
    end
    if object_info.green_val(k) >= avg_th && object_info.green_val_max(k) > max_th
        object_info.green_val_bin(k) = 1;
    end
    if object_info.blue_val(k) >= avg_th && object_info.blue_val_max(k) > max_th
        object_info.blue_val_bin(k) = 1;
    end
end
```

Images for Good Test Run 1

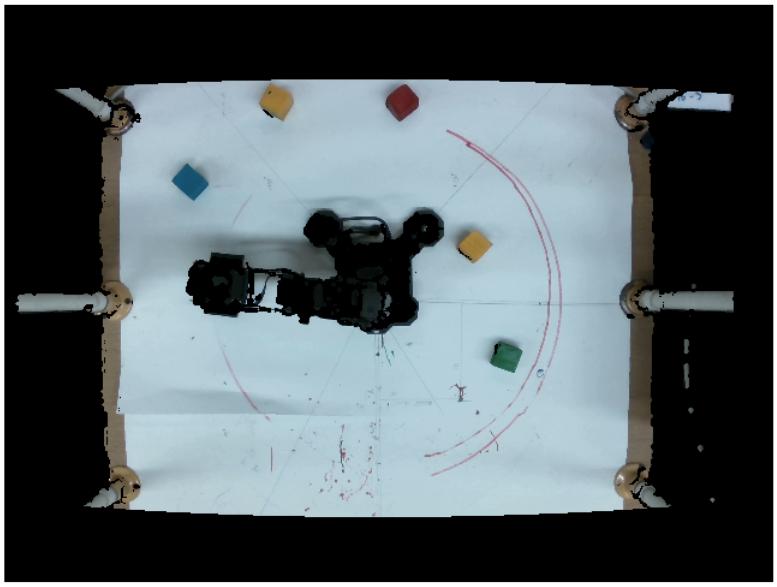

Figure 1: Captured RGB Image

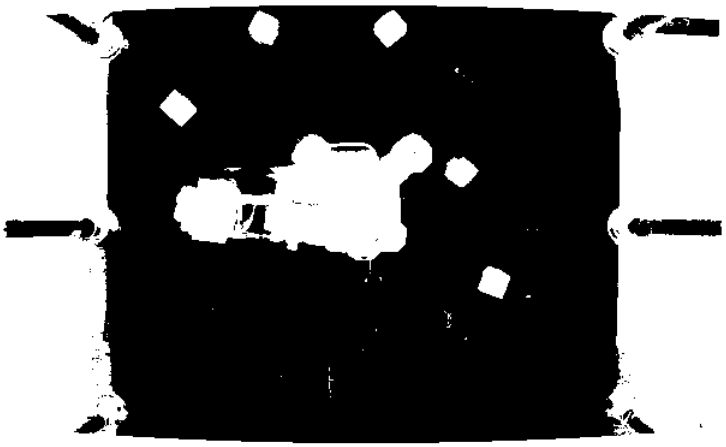

Figure 2: Background assigned Black, Objects assigned White

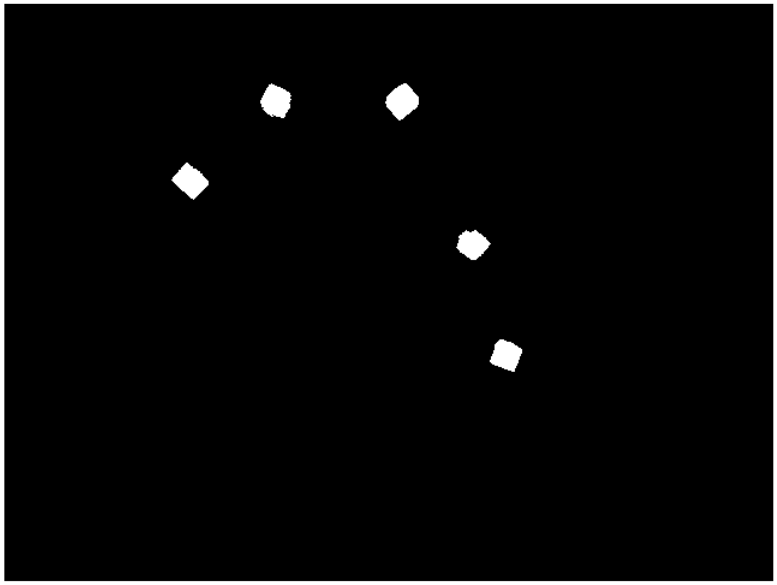

Figure 3: Removed Unwanted Objects

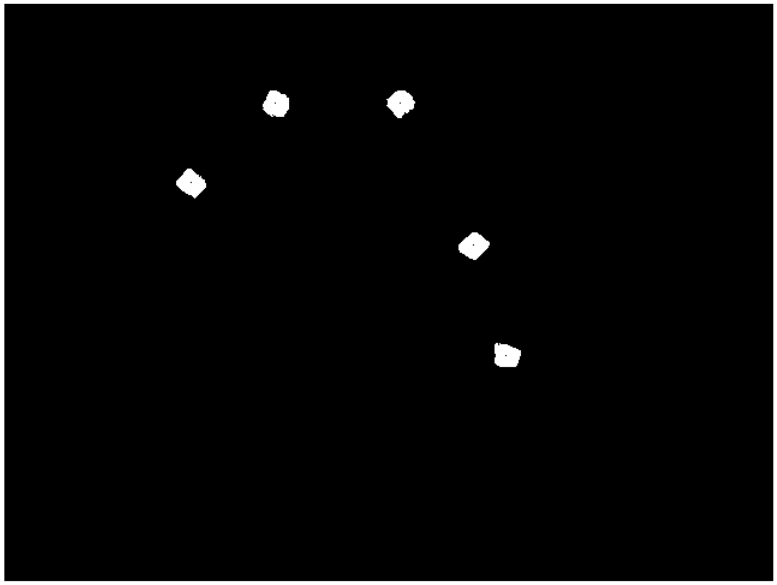

Figure 4: Top-Face Extraction and Determining Center Pixel

Images for Good Test Run 2

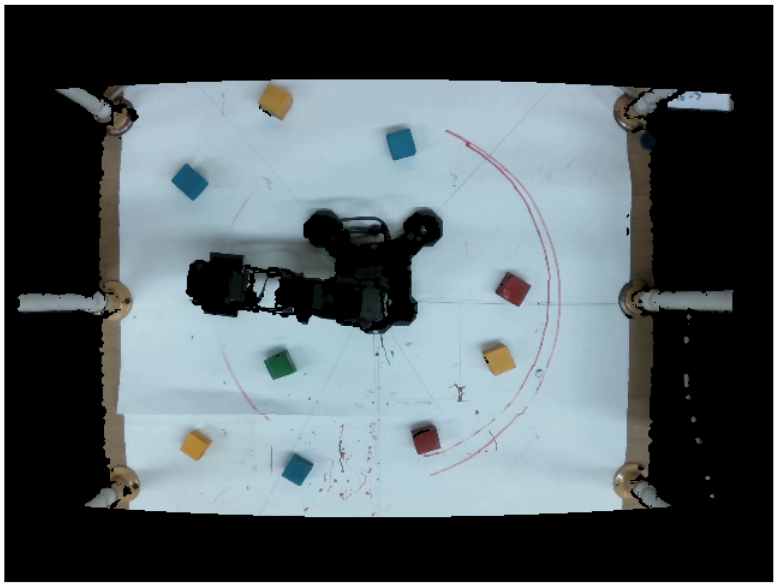

Figure 5: Captured RGB Image

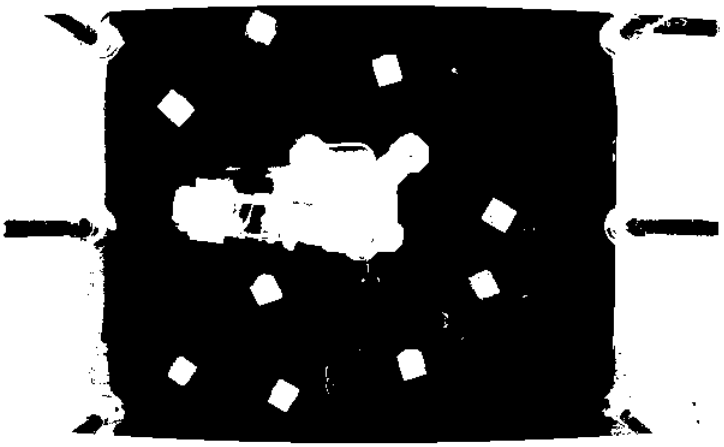

Figure 6: Background assigned Black, Objects assigned White

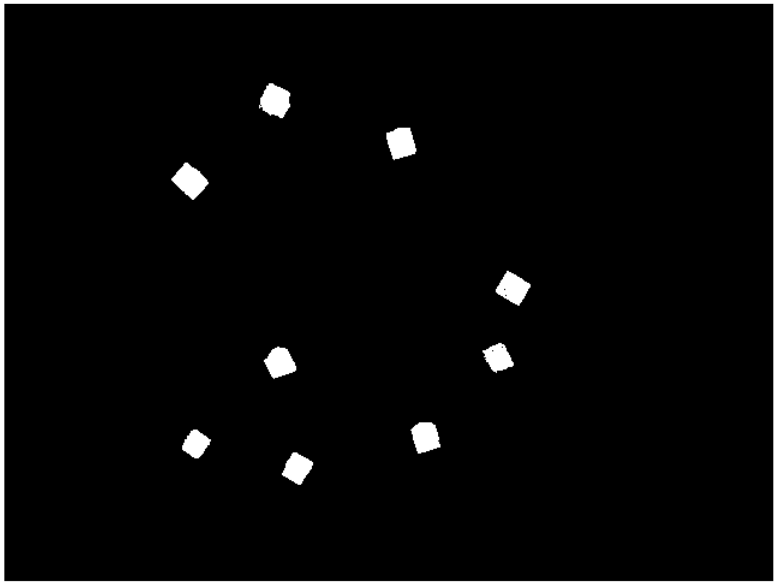

Figure 7: Removed Unwanted Objects

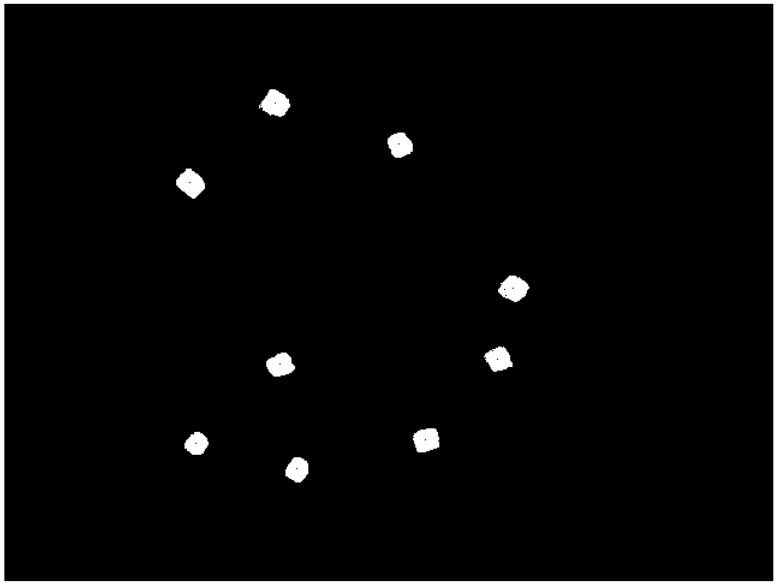

Figure 8: Top-Face Extraction and Determining Center Pixel

Statistics

Placing different types of colors (there were 4 different colors at max) did not caused any problem in the color detection. Infact, all the individual color is correctly detected too in both the above good test runs. The results are below:

```
  red_val_bin: [0 1 1 1 0]
green_val_bin: [1 1 0 1 1]
 blue_val_bin: [1 0 0 0 0]
```

Figure 9: RGB Detection for Test 1

```
  red_val_bin: [0 1 1 0 0 0 1 1 1]
green_val_bin: [1 1 1 1 1 1 0 1 0]
 blue_val_bin: [1 0 0 0 1 1 0 0 0]
```

Figure 10: RGB Detection for Test 2

However after running multiple trials it was seen that the accuracy detection for both the red and blue colors were great, i.e., more than 90% but for green color the accuracy fell around 75%