Introduction To Robotics
Lab 04

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

10th May 2024

1 Task 4.1

The Provided Matlab Code for the function pointcloud example

```
function pointcloud_example()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();

    % Start streaming on an arbitrary camera with default settings
    profile = pipe.start();


    %% Acquire device parameters
    % Get streaming device's name
    dev = profile.get_device();

    % Access Depth Sensor
    depth_sensor = dev.first('depth_sensor');

    % Find the mapping from 1 depth unit to meters, i.e. 1 depth unit =
    % depth_scaling meters.
    depth_scaling = depth_sensor.get_depth_scale();

    % Extract the depth stream
    depth_stream = profile.get_stream(realsense.stream.depth).as('
       video_stream_profile');

    % Get the intrinsics
    depth_intrinsics = depth_stream.get_intrinsics();

    %% Align the frames and then get the frames
    % Get frames. We discard the first couple to allow
    % the camera time to settle
    for i = 1:5
        fs = pipe.wait_for_frames();
    end

    % Alignment is necessary as the depth cameras and RGB cameras are
    % physically separated. So, the same (x,y,z) in real world maps to
    % different (u,v) in the depth image and the color images. To build a
    % point cloud we only need depth image, but if we want the color the
    % cloud then we'll need the other image.

    % Since the two images are of different sizes, we can either align the
    % depth to color image, or the color to depth.
    % Change the argument to realsense.stream.color to align to the color
    % image.
    align_to_depth = realsense.align(realsense.stream.depth);
    fs = align_to_depth.process(fs);

    % Stop streaming
    pipe.stop();

    % Extract the depth frame
    depth = fs.get_depth_frame();
    depth_data = double(depth.get_data());
    depth_frame = permute(reshape(depth_data',[ depth.get_width(),depth.get_height()]),[2 1]);

    % Extract the color frame
    color = fs.get_color_frame();
    color_data = color.get_data();
    color_frame = permute(reshape(color_data',[3,color.get_width(),color.get_height()]),[3 2 1]);

    %% Create a point cloud using MATLAB library
    % Create a MATLAB intrinsics object
    intrinsics = cameraIntrinsics([depth_intrinsics.fx,depth_intrinsics.fy],[depth_intrinsics.ppx,depth_intrinsics.ppy],size(depth_frame));

    % Create a point cloud
    ptCloud = pcfromdepth(depth_frame,1/depth_scaling,intrinsics,ColorImage=color_frame);

    % Display point cloud
    pcshow(ptCloud,'VerticalAxisDir','Down');


end
```

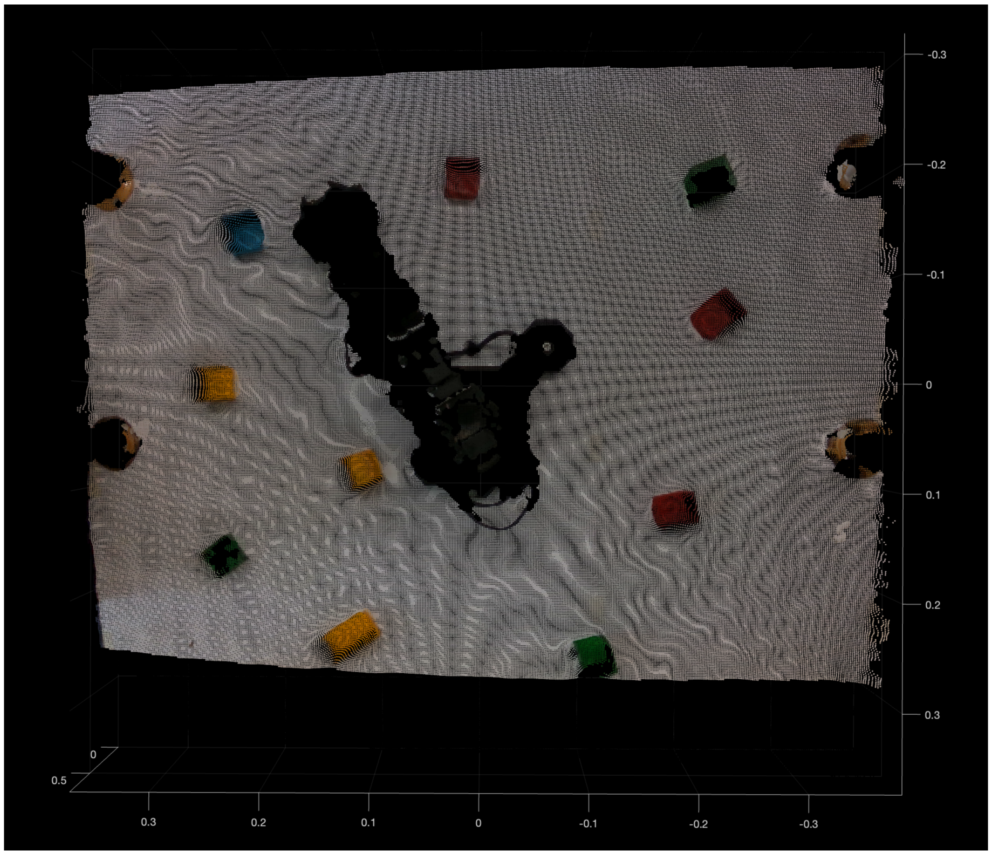

Figure 1: Point Cloud Output

2 Task 4.2

2.1 Strategy for Assigning Frames to Cubes

Our approach in this context revolves around creating a reference point cloud at the camera frame origin instead of directly assigning frames to individual cubes. Through the use of Iterative Closest Point (ICP) registration, we obtain a transformation matrix that maps this reference point cloud to each cube in the workspace. This method avoids the necessity of explicitly assigning frames, relying on the transformation matrices to determine the position of each cube relative to the camera frame.

2.2 Description of any Filters/Processing Applied to the Image Frames

During the initial image processing stage, a color-specific mask is used to separate cubes of the specified color. This is done using the ColorThresholder application to create a binary mask. Subsequently, the depth frame is adjusted to eliminate depth data associated with areas outside the color mask, concentrating solely on the target cube areas. This filtering step guarantees that subsequent analyses are conducted solely on pertinent data, thereby improving the accuracy of the workflow.

2.3 Description of Segmentation Strategy and Rationale Behind it

After creating a masked point cloud that includes cubes of the specified color, we proceed with segmentation based on distance. Our approach relies on the spatial distance between cubes to divide the point cloud into separate clusters, each corresponding to a single cube within the environment. Using the pcsegdist function, we assign labels to points, facilitating the recognition of distinct clusters. This segmentation method proves highly efficient when cubes are adequately spaced apart, enabling precise differentiation between individual objects.

2.4 Description of Pose Estimation Stage

Once the segmentation is completed, the next step in the pipeline is to determine the orientation of each cube. Initially, any incorrect labels are eliminated to ensure precision. Subsequently, separate point clouds are generated for each cube based on their specific labels. Using the pcregistericp function, a matrix for transformation is calculated to align a reference point cloud with each segmented cube. To remove potential rotations around the X and Y axes, the rotation matrix obtained from this transformation is converted into Euler angles, with any rotations along these axes set to zero before being converted back into a rotation matrix. This method effectively establishes the orientation of each cube with respect to the camera frame.

2.5 Aptly Commented Code for the Entire Pipeline

2.5.1 Red Masking

```
colorData = ptCloud.Color; %stores an array of ((480*3) by (640*3))

    % Display color values
    % size(colorData);
    % R_layer = colorData(1:480, 1:640,1);
    % G_layer = colorData(1:480, 1:640,2);
    % B_layer = colorData(1:480, 1:640,3);
    % redmask = ~((R_layer >= 110) & (G_layer <= 200) & (B_layer <=
       200));

    red = imrotate(RedMask(colorData),180);
    % imshow(R_layer)
    imshow(red);

    gridStep = 0.1;
    ptCloudA = pcdownsample(ptCloud,'gridAverage',gridStep);
```

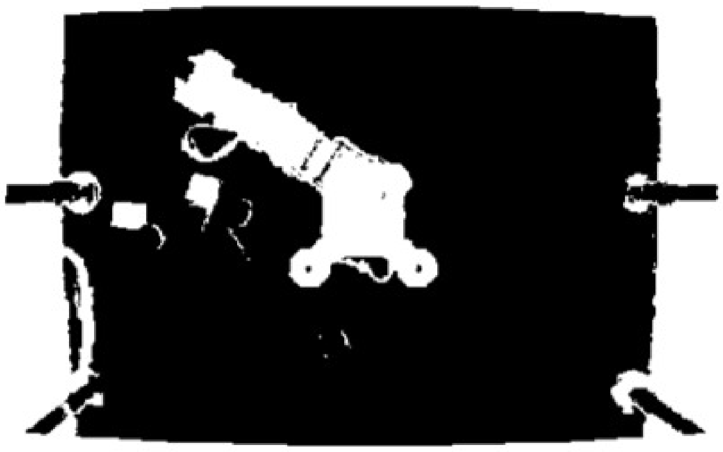

Figure 2: Red Masking

2.6 Point Cloud Generation

```
% Load point cloud data and frames
[ptCloud, color_frame, depth_frame, intrinsics, depth_scaling] =
   pointcloud_example();

% Apply color mask to isolate red cubes
[redmask, maskedRedImage] = bluemasklab4(color_frame);

% Set depth of non-red areas to infinity
depth_frame(~redmask) = inf;

% Generate a point cloud for the red cubes
ptCloudRed = pcfromdepth(depth_frame, 1/depth_scaling, intrinsics, "
   ColorImage", color_frame);

% Remove invalid points from the red point cloud
[ptCloudRed, indices] = removeInvalidPoints(ptCloudRed);

% Visualize the red point cloud
figure;
pcshow(ptCloudRed, 'MarkerSize', 30, 'BackgroundColor', 'w');
xlabel("X Axis");
ylabel('Y Axis');
zlabel('Z Axis');

% Segment the red point cloud into clusters
[label, numClusters] = pcsegdist(ptCloudRed, 0.03);
```

```matlab
idxValidPoints = find(label);
labelColorIndex = label(idxValidPoints);
linear_indices = find(label == 1);
segmentedPtCloud = select(ptCloudRed, idxValidPoints(linear_indices));

% Visualize segmented clusters
figure;
pcshow(segmentedPtCloud, "BackgroundColor", 'w', 'MarkerSize', 50);
hold on;

% Register and transform a 3D grid to align with the segmented point
   cloud
spacing = 8;
factor = 0.0003;
[X, Y, Z] = ndgrid(1:spacing:100, 1:spacing:100, 1:spacing:100);
X = X * factor;
Y = Y * factor;
Z = Z * factor;
pointCloud2 = [X(:), Y(:), Z(:)];
pointCloud2 = pointCloud(pointCloud2);
tform = pcregistericp(pointCloud2, segmentedPtCloud);
euler = rotm2eul(tform.R);
euler1 = [euler(:, 1) 0 0];
rotation = eul2rotm(euler1);
tform.R = rotation;
ptCloudTransformed = pctransform(pointCloud2, tform);

% Visualize the transformed point cloud
pcshow(ptCloudTransformed, 'BackgroundColor', 'w', 'MarkerSize', 20);
xlabel("X Axis");
ylabel('Y Axis');
zlabel('Z Axis');
hold off;

% Visualize clusters in the original red point cloud
figure;
colormap(hsv(numClusters));
pcshow(ptCloudRed.Location, labelColorIndex);
title('Point Cloud Clusters');
```

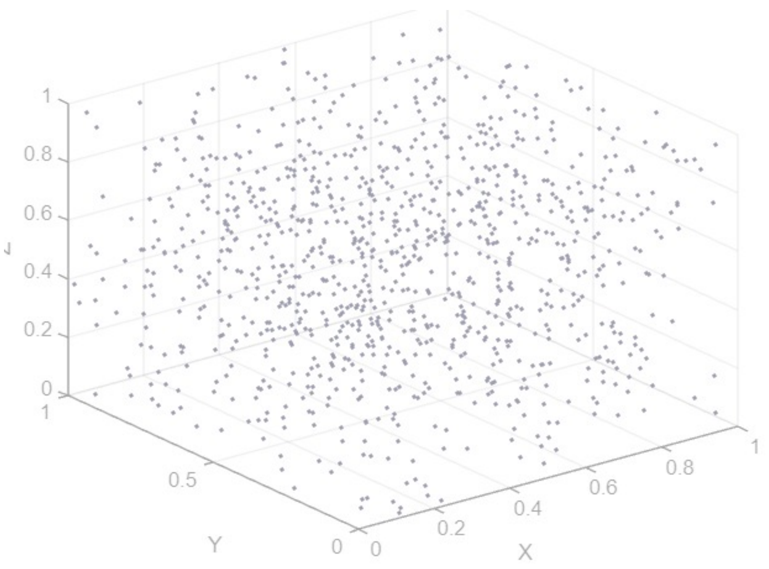

Figure 3: Mapping the reference point cloud to the segmented point cloud

2.7 Accuracy Evaluation of the Pipeline

In assessing the pipeline's capacity to accurately determine the pose (position and orientation) of cubes within the workspace, we designed an experiment focusing on the alignment between two point clouds. One point cloud was generated by the model, while the other came from segmenting the real scene (segmented block point cloud). This experiment encompassed various conditions: introducing different colored cubes like red and blue to test the pipeline's robustness in cluttered environments and its ability to discern cube types, placing cubes at different positions within the workspace to evaluate performance, and assessing accuracy in scenarios with both single and multiple cubes present simultaneously. The successful completion of this evaluation affirmed the pipeline's effectiveness in precisely determining the cube pose within the workspace.