Perception 2

Intro to Robotics

Iffat Fatima if09270

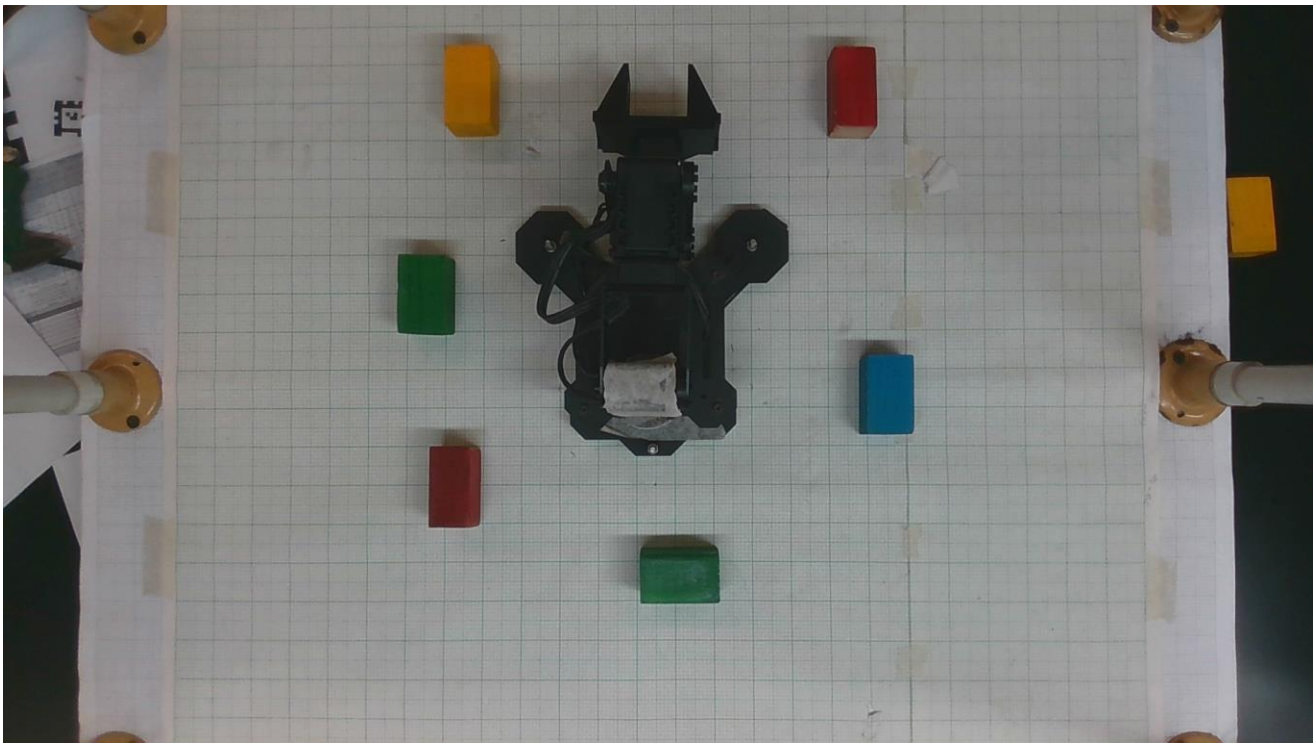
Hijab Fatima Siddiqui hs09195

Faiza Salman Khatri fs09192

Color Segmentation Stage for the Identification of a Single Color:

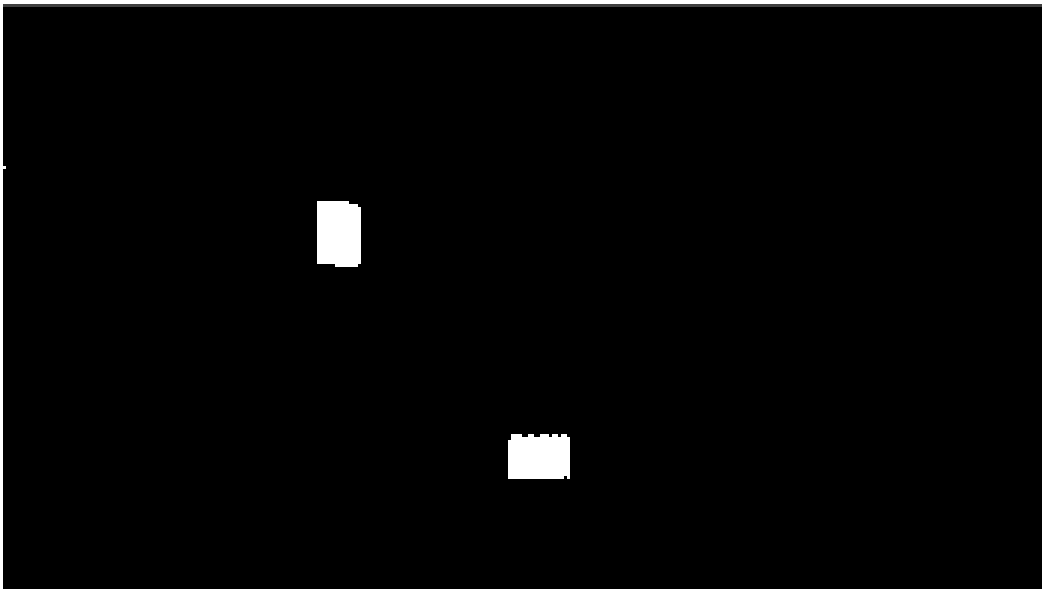
The task of exposing only certain blocks of a single color was done by first splitting the image into h,s,v components and then creating a color mask for the identification of a certain color.

Original Image:



Red Mask

Green Mask



Code:

```
rgb = imread("Downloads/rgb_image_Color.png");

figure('Position', [100 100 2500 400]);

subplot(1,3,1)
imshow(rgb);
title('Original Image')

hsv_img = rgb2hsv(rgb);
[h, s, v] = imsplit(hsv_img);

% extracting red blocks
redMask = (h < 0.03 | h > 0.95) & s > 0.5 & v > 0.25;
subplot(1,3,2)
imshow(redMask);
title('Red Mask')

% extracting green blocks
greenMask = (h >= 0.25 & h <= 0.45) & (s >= 0.23) & (v >= 0.2);
subplot(1,3,3)
imshow(greenMask);
title('Green Mask')
```

- How did you choose your segmentation strategy?
  The segmentation strategy of using HSV values for color thresholding was chosen because, as mentioned in [1], it works best for RGB images with variable lighting.

- Are areas of the object missed (zoom in)?
  Yes. Naturally, some values of the object itself fail to match the criteria set and the object mask has small holes here and there. However, this is unavoidable and does not significantly hurt our calculations

- Does the location of the object have an effect?
  The location of the object mattered only as much as the lighting variance the spot provided, as expanded on in the next part.

- Does ambient lighting have an effect?
  The lighting was found to matter a lot in the segmentation success. Pixels in darkness were less likely to be recognized as the color they are if they did not receive sufficient exposure. The tuning of the camera parameters to set the brightness and exposure was thus necessary and was done through trial and error.

Task 2.12: Determining Pose:

To determine the pose, we relied on the following geometric features:

- Planar Surface: The cube is always placed on a planar surface (table), and can easily be removed, isolating the objects on the plane.
- Spatial Clustering: Euclidean distance-based clustering can be used to separate clusters of multiple identified objects.
- Cube Geometry: The cube has a known width, shape and height. Its sides are mutually orthogonal along all three axes. It has a symmetry about its centroid. All these facts may be used to determine its pose.

The algorithm to determine the pose was devised as follows:

- The depth frame must be aligned with the color frame, and camera intrinsics extracted and used to create a 3D model of the image using the depth information.
- The table must be removed from the image and objects on the table isolated for inspection.
- The HSV color segmentation strategy must be used to separate objects of the desired color.
- The objects identified must be clustered into different groups based on Euclidean distances for multiple objects of the same color.
- The centroid of each block is calculated as the mean of all 3D points belonging to a specific cluster.
- PCA strategy is applied on the centered points of all clusters and eigenvectors extracted, which give the three principal axes of the cube, forming the rotation matrix.
- The transformation matrix is formed using the distance vector (vector incorporating the translation of the centroid from the origin), and the rotation matrix.

The accuracy was judged to be fairly high for images with ambient lighting and depth image. Based on multiple testing, position was stable and centroid visually matched the cube centers. Small error variance caused by sensor noise were expected and accepted. Orientation also repeatedly matched the cube axes and was also judged to be stable.

After experimenting with different conditions, it is found that accuracy improves when:

- Cube is well-separated from other objects.
- Lighting is uniform.
- A large number of cube points are visible (the mask is correctly and sufficiently applied).

Task 3.1: Explore Point Clouds

Milestone 2: Perception Pipeline

1. Task Definition:
    a. What objects are being perceived?

The system is designed to perceive colored cubes (blocks) placed on a planar table surface. Each cube has a known geometry (fixed width, height, and orthogonal faces), distinct color (used for segmentation in HSV space) and is placed on a flat, static surface. The perception system isolates these cubes from the background (table) and from other objects.

    b. What quantities are being estimated, e.g. position, orientation, pose?

The following quantities are estimated for each detected cube:

- Position (3D centroid coordinates - x,y,z) Computed using depth data and camera intrinsics.
- Orientation (Rotation matrix) Obtained using PCA on the 3D clustered points. The eigenvectors represent the cube's principal axes.
- Full Pose (Homogeneous transformation matrix)

Thus, the system estimates the full 6-DoF pose of each cube.

    c. In which frame(s) is the output being expressed?

The pose is expressed in the camera coordinate frame since depth data is aligned with the RGB frame and 3D reconstruction uses camera intrinsics. No transformation to robot base frame is yet applied. If needed for manipulation, the pose can later be transformed to the robot base frame using extrinsic calibration.

    d. Which camera is used? Where is it mounted?

A RGB-D camera (color + depth sensor) is used. The camera captures both color images and depth frames and is mounted above the workspace. It faces downward toward the table surface and is rigidly fixed to avoid calibration drift. This mounting ensures clear view of cubes, minimal occlusion and stable depth measurements

    e. How will the output be used by the rest of the system?

The perception pipeline outputs the full 6-DoF pose of each detected cube in the camera coordinate frame in the form of position, orientation as a rotation matrix and a homogeneous Transformation Matrix.

In our implementation, this is expressed as:

```
T = eye(4);
T(1:3,1:3) = R;
T(1:3,4) = position';
```

The pose output will be used in the following way:

- Frame Transformation: Since pose is expressed in the camera frame, it must be transformed into the robot base frame using extrinsic calibration:
- Grasp Planning: The cube's centroid provides the grasp target position and the PCA-derived axes provide alignment information for the gripper. The robot can align its end-effector with one of the cube's principal axes.
- Motion Planning: The position feeds into inverse kinematics while the rotation matrix defines the desired end-effector orientation. As such, a collision-free trajectory is generated toward the grasp pose.

Thus, the perception module provides structured geometric information that directly enables robotic manipulation.

f. What is the success criteria of your perception pipeline?

The perception pipeline is considered successful if it meets the following quantitative and qualitative criteria:

Detection Success:

- The correct number of colored cubes is detected.
- No false positives (table or noise incorrectly classified as cube).
- Clustering separates multiple cubes correctly.

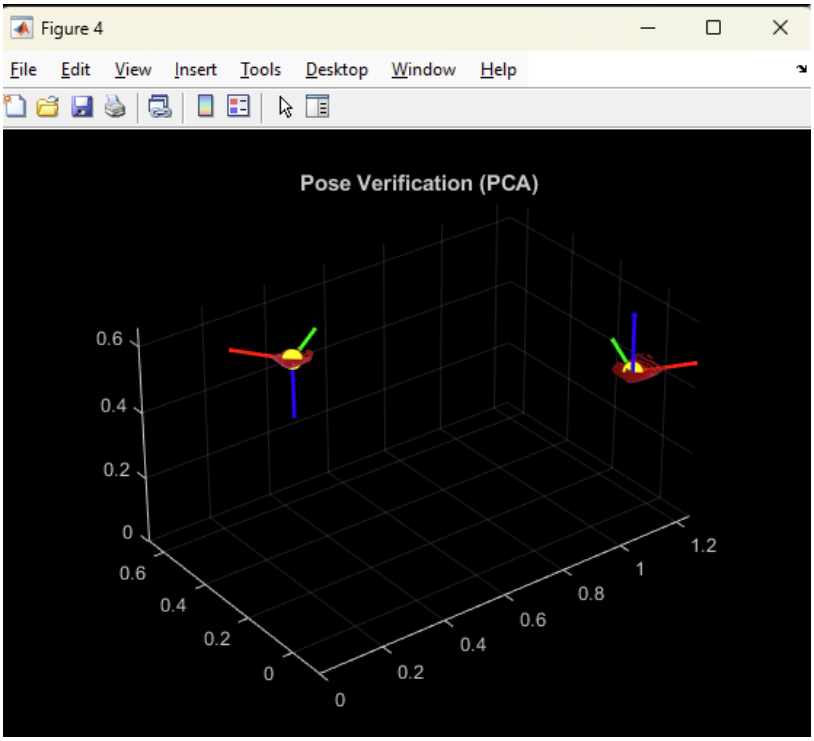
This is verified using:

```
[labels, numClusters] = pcsegdist(redCloud, minDistance);
```

Position Accuracy:

- The centroid visually aligns with the geometric center of the cube.
- Repeated captures under same conditions yield low variance.
- Depth noise causes only small acceptable fluctuations (millimeter-level variation).

The position output is verified using the plot generated:



Orientation Accuracy:

- PCA eigenvectors align with cube edges.
- Rotation matrix forms a valid right-handed orthonormal basis.
- Determinant check ensures physical validity:

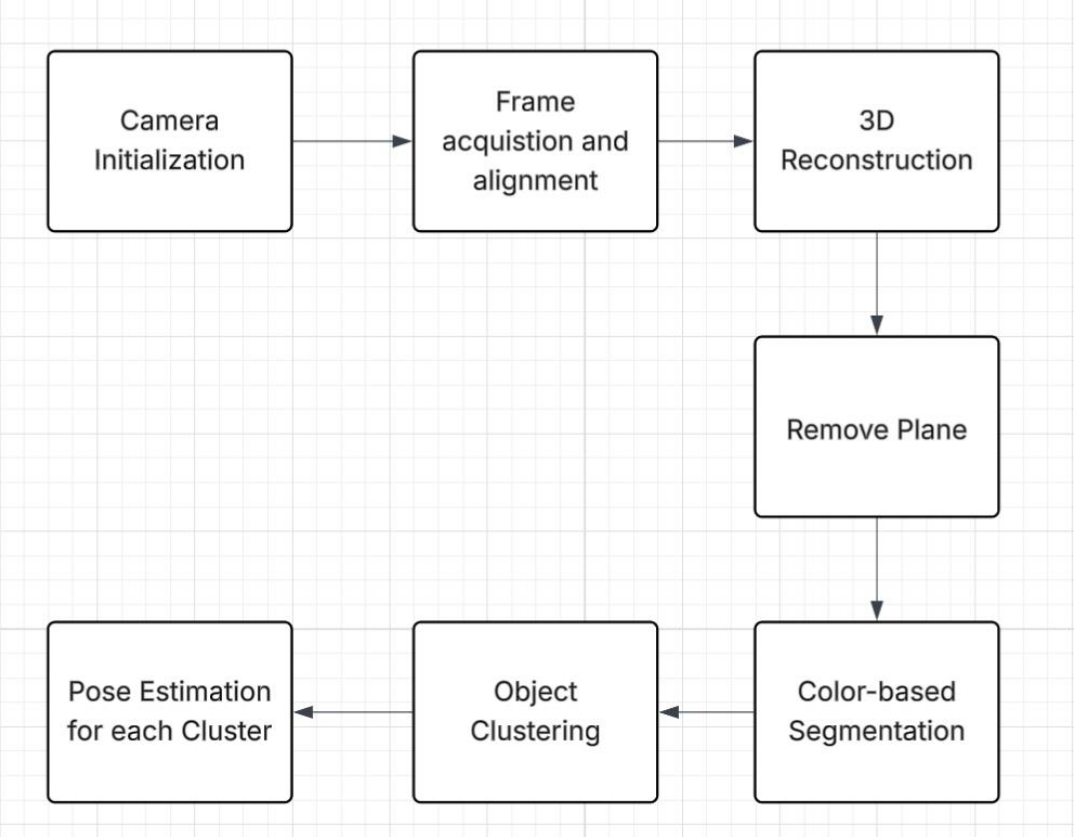
```
if det(R) < 0
    R(:,3) = -R(:,3);
end
```

The system must maintain performance when:

- Lighting is uniform but not perfect
- Minor sensor noise exists
- Cubes are slightly repositioned
- Multiple cubes of same color exist

Measurements must remain stable across different measurements.

2. Pipeline:
    a. Block Diagram (Camera to the Final Pose Output)



Set camera -> start streaming on camera -> acquire device parameters -> access depth sensor and scale 1 depth unit to meters -> extract the depth stream -> get the depth intrinsics -> align and get the frames after discarding first few ones -> align depth image to the color image -> stop streaming -> extract the depth and color frames -> create a camera intrinsics object -> create a point cloud using the intrinsics and tune it-> visualize the point cloud -> remove the plane -> extract colors -> compute HSV values -> apply red mask -> create red point cloud and tune it-> show the red point cloud -> cluster the red blocks -> count points in each cluster -> sort clusters by size -> process each block -> extract the location (x,y,z coordinates) of all the points in red point cloud in matrix form -> remove invalid points-> compute centroid -> compute orientation using PCA -> treat PCS axes as rotation matrix -> ensure right handed coordinate system -> convert to Euler angles (yaw, pitch, roll) -> create homogenous transformation matrix -> visualize pose -> plot centroid -> plot orientation axes -> print pose information

b. Descriptions (Provide I/O info for each block, operations performed, assumptions, reference to relevant functions, example output image for each stage and code explanations)

1. Camera Initialization:
   Input: connected intel realsense RGB-D camera
   Output: active streaming pipeline, device object, depth scaling factor (depth unit -> meters), depth camera intrinsic parameters

   Operations performed: create realsense pipeline object, start streaming with default configuration, access device and depth sensor, retrieve depth scaling factor, extract depth stream profile, obtain intrinsic calibration parameters.

   Assumptions: the realsense camera is connected and recognized by the system, default configuration for stream gives synchronized depth and color

   Reference to functions: realsense.pipeline(), pipe.start(), profile.get_device(), first('depth_sensor'), get_depth_scale(), get_stream(), get_intrinsics()

   Output image: no image is produced at this stage since we are initializing parameters here

   Code explanation:
   ```
   %Make Pipeline object to manage streaming
   pipe = realsense.pipeline();
   % Start streaming on an arbitrary camera with default settings
   profile = pipe.start();
   % Acquire device parameters
   dev = profile.get_device();
   % Access Depth Sensor
   depth_sensor = dev.first('depth_sensor');
   % Find the mapping from 1 depth unit to 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();
   ```

2. Frame acquisition and alignment:

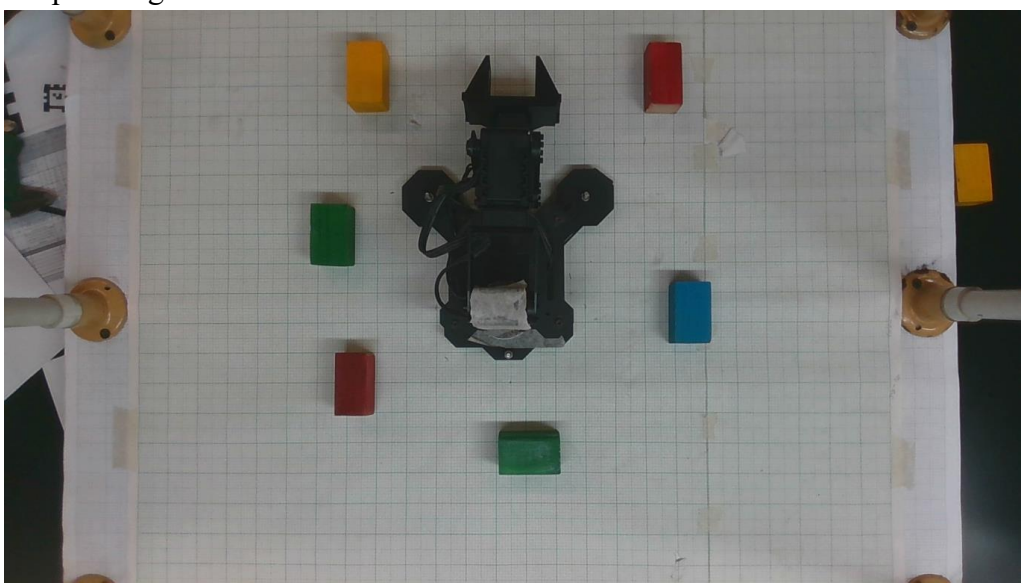
Input: active camera stream
Output: aligned depth and color frames

Operations performed: capture multiple frames to stabilize auto-exposure and discard the first few ones, align depth frame to color frame, extract depth and rgb image matrices, stop streaming

Assumptions: depth and color sensors are physically offset and alignment is required so depth is aligned to color since segmentation uses color coordinates.

Reference to functions: pipe.wait_for_frames(), realsense.align(), process(), get_depth_frame(), get_color_frame(), pipe.stop()

Output image:



Code explanation:

```
% Align the frames and then get the 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.color);
% I want it to align to color as my coordiantes [u v] will be from color
image

    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]);
```

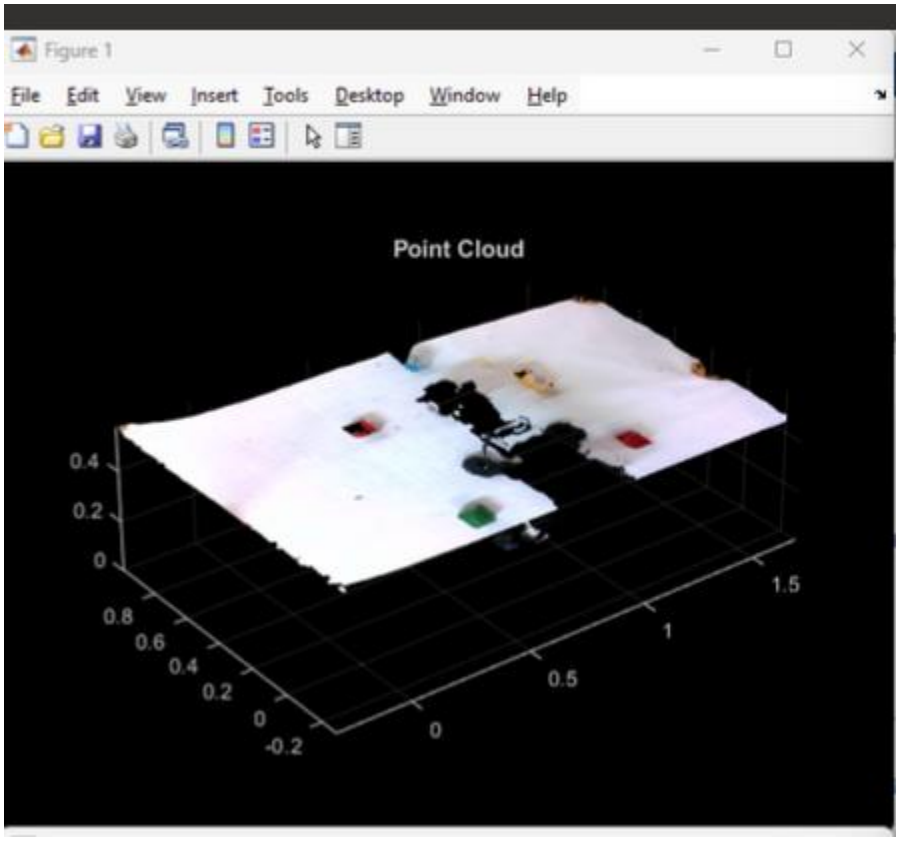
3. 3D Reconstruction:
   Input: depth image, rgb image, camera intrinsics, depth scaling
   Output: colored 3D point cloud

   Operations performed: create a camera intrinsics object, convert depth image to metric units, generate 3D point cloud, attach rgb values

   Assumptions: no lens distortion compensation needed

Reference to functions: cameraIntrinsics(), pcfromdepth(). pcshow()

Output image:



Code explanation:

```
% 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);
figure; pcshow(ptCloud);
title('Point Cloud');
```

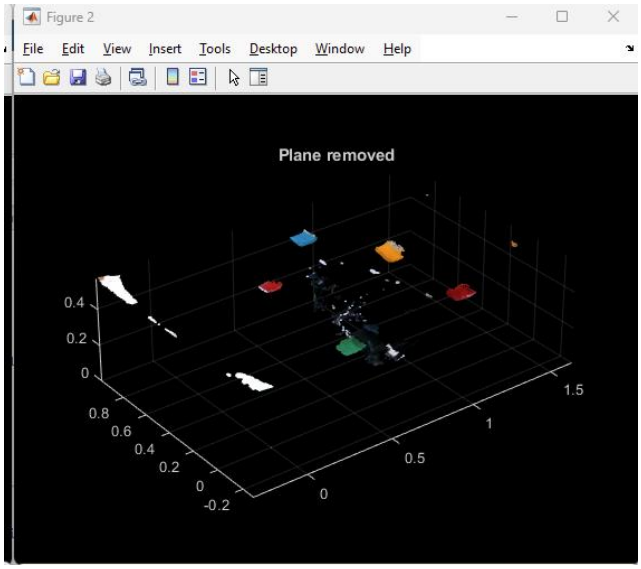
4. Removing Plane:
   Input: full scene point cloud
   Output: Object cloud with isolated objects only, and plane removed

Operations performed: fit dominant plane using RANSAC, identify and remove plane,

Assumptions: largest plane corresponds to table, objects lie above plane

Reference to functions: pcfitplane() , select()

Output image:



Code explanation:

```
maxDistance = 0.015;
[planeModel, inlierIdx, outlierIdx] = pcfitplane(ptCloud, maxDistance);

% Remove table AND keep non-plane points
objectsCloud = select(ptCloud, outlierIdx);
figure; pcshow(objectsCloud);
title('Plane removed');
```

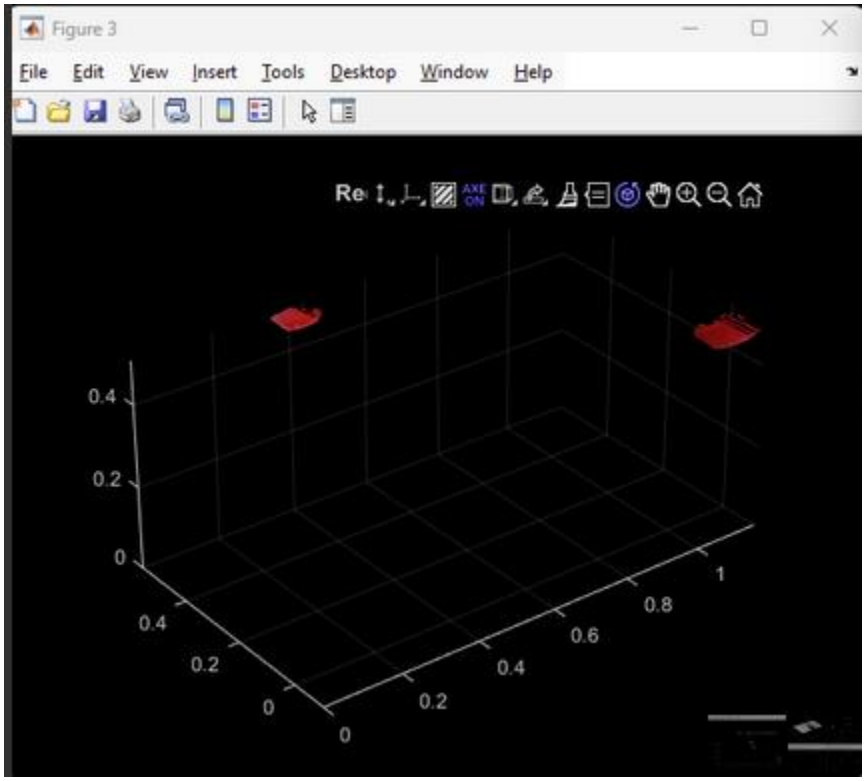
5. Color-based Segmentation:
   Input: full scene point cloud
   Output: red-only point cloud

   Operations performed: fextract rgb values, convert rgb to hsv, apply red mask, remove noise

   Assumptions: target objects are red

Reference to functions: .Color , rgb2hsv() , im2double(), pcdenoise()

Output image:



Code explanation:

```
% extarct colors
   colors = objectsCloud.Color;
%convert to hsv
   hsvVals = rgb2hsv(im2double(colors));
   %extract hsv
   H = hsvVals(:,1);
   S = hsvVals(:,2);
   V = hsvVals(:,3);
   %apply red mask
   redMask = (H < 0.05 | H > 0.95) & S > 0.5 & V > 0.2;
   %create red point cloud
   redCloud = select(objectsCloud, find(redMask));
   redCloud = pcdenoise(redCloud);
   figure; pcshow(redCloud);
   title('Red blocks extracted');
```

6. Object clustering:
   Input: red point cloud
   Output: cluster labels, sorted cluster indices

   Operations performed: Perform Euclidean distance clustering, Count points per cluster, Sort clusters by size

   Assumptions: Blocks are spatially separated

   Reference to functions: pcsegdist(), sort()

   Output image:
   Clusters

   Code explanation:

```
 minDistance = 0.02;   % tune based on spacing
   [labels, numClusters] = pcsegdist(redCloud, minDistance);

%count points in each cluster
   clusterSizes = zeros(numClusters,1);

   for i = 1:numClusters
      clusterSizes(i) = sum(labels == i);
   end

   % Sort clusters by size
   [sortedSizes, sortedIdx] = sort(clusterSizes,'descend');
```

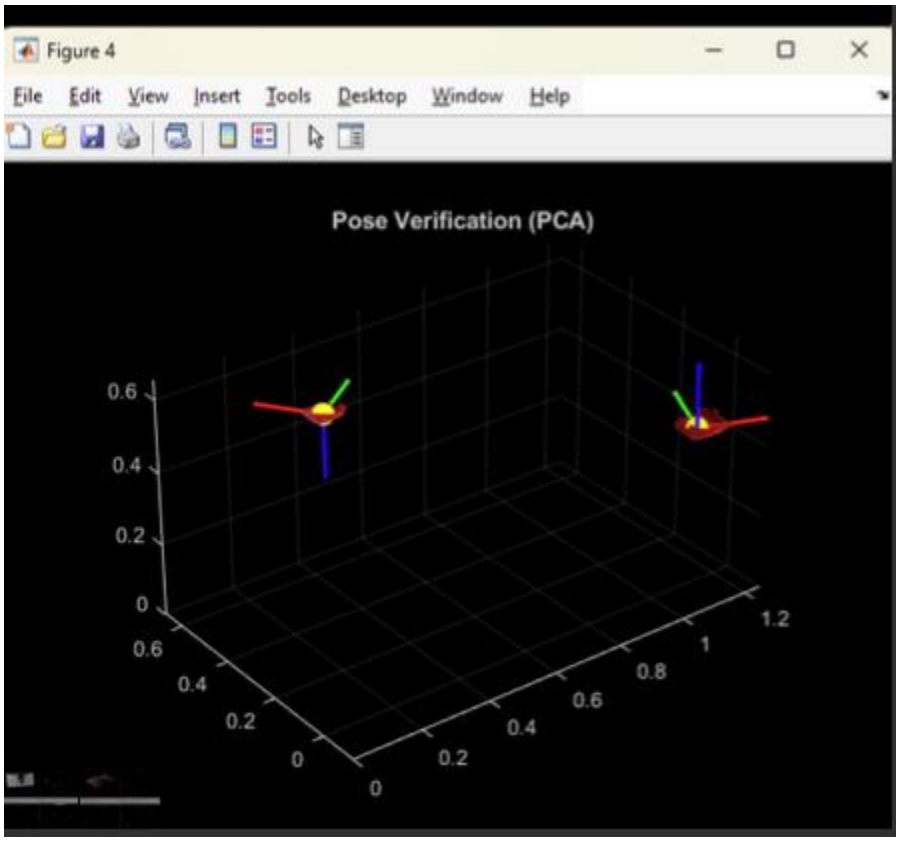
7. Pose estimation for each cluster:
   Input: single block point cloud
   Output: centroid position, rotation matrix, euler angles, homogenous transform

   Operations performed: Extract XYZ coordinates, Remove invalid points, Compute centroid, Perform PCA, Construct rotation matrix, Enforce right-handed frame, Convert to Euler angles, Construct 4×4 transform, Plot centroid, Plot local coordinate axes, Print pose values

   Assumptions: object shape has dominant axes, PCA principal axes align with object orientation

Reference to functions: mean(), pca(), det(), rotm2eul(), rad2deg(), pcshow() plot3() quiver3() fprintf()

Output image:



Code explanation:

```
figure;

for k = 1:numBlocksExpected
    i = sortedIdx(k);
    idx = find(labels == i);
    blockCloud = select(redCloud, idx);

    pts = blockCloud.Location;
    pts = reshape(pts, [], 3);

    pts = pts(~any(isnan(pts),2),:);

    % ---- Position ----
    position = mean(pts,1);

    % ---- Orientation via PCA ----
    ptsCentered = pts - position;

    [coeff, ~, ~] = pca(ptsCentered);

    R = coeff;   % 3x3 rotation matrix
    if det(R) < 0
        R(:,3) = -R(:,3);
    end
    eul = rotm2eul(R);   % [yaw pitch roll] in radians
    eul_deg = rad2deg(eul);
    T = eye(4);
    T(1:3,1:3) = R;
    T(1:3,4) = position;
    pcshow(redCloud);
    hold on;

    % Plot centroid
    plot3(position(1), position(2), position(3),'o','MarkerSize',10, 'MarkerEdgeColor','y','MarkerFaceColor','y');

    % Plot orientation axes
    scale = 0.2;

    quiver3(position(1),position(2),position(3), ...
            scale*R(1,1),scale*R(2,1),scale*R(3,1), ...
            'r','LineWidth',2);

    quiver3(position(1),position(2),position(3), ...
            scale*R(1,2),scale*R(2,2),scale*R(3,2), ...
            'g','LineWidth',2);

    quiver3(position(1),position(2),position(3), ...
            scale*R(1,3),scale*R(2,3),scale*R(3,3), ...
            'b','LineWidth',2);

    title('Pose Verification (PCA)');

    fprintf('\n============================\n');
    fprintf('Block %d\n', k);
    fprintf('============================\n');

    fprintf('Position (meters):\n');
    fprintf('x = %.4f\n', position(1));
    fprintf('y = %.4f\n', position(2));
    fprintf('z = %.4f\n\n', position(3));

    fprintf('Rotation Matrix (R):\n');
    disp(R);

    fprintf('Euler Angles ZYX (degrees):\n');
    fprintf('Yaw   = %.2f\n', eul_deg(1));
    fprintf('Pitch = %.2f\n', eul_deg(2));
    fprintf('Roll  = %.2f\n\n', eul_deg(3));

    fprintf('Homogeneous Transform (T):\n');
    disp(T);


end

    hold off;
```

3. Results:
    a. Description of the Testing Setup:
        i. How were the objects placed?

The objects were placed such that the robot arm was at the center in most cases, with numerous differently colored blocks around it. All objects were placed on a planar surface (table). This formed the workspace for the robot.

        ii. What were the lighting conditions?

All objects were directly placed under a overhead white light, providing sufficient ambient lighting for clear determination and image processing.

    b. Details of the Performed Tests (conducted N times with the objects reset at the same place)

The perception pipeline was evaluated through repeated controlled experiments using the Intel RealSense RGB-D camera mounted above a planar table. The objective was to assess detection reliability, pose accuracy, and robustness of the full pipeline from point cloud generation to final pose estimation.

Testing setup

Colored cubes were placed on a flat table within the field of view of the overhead camera. The camera remained fixed throughout all experiments to avoid recalibration. The cubes were positioned at different locations on the table, sometimes well separated and sometimes closer together.

Lighting conditions were tested under normal indoor ambient lighting. Additional trials were conducted with slightly dimmed lighting and with light coming from one side to introduce mild shadows.

Nominal condition tests

Under uniform lighting and clear separation between cubes, the pipeline was executed multiple times (approximately 8–10 trials) with the cubes reset to the same positions before each run.

Repositioning tests

Cubes were moved to different positions on the table while maintaining similar lighting. The pipeline was run again without modifying parameters.

Non-nominal condition test

One stress test was conducted by reducing lighting and introducing partial shadow on the cube.

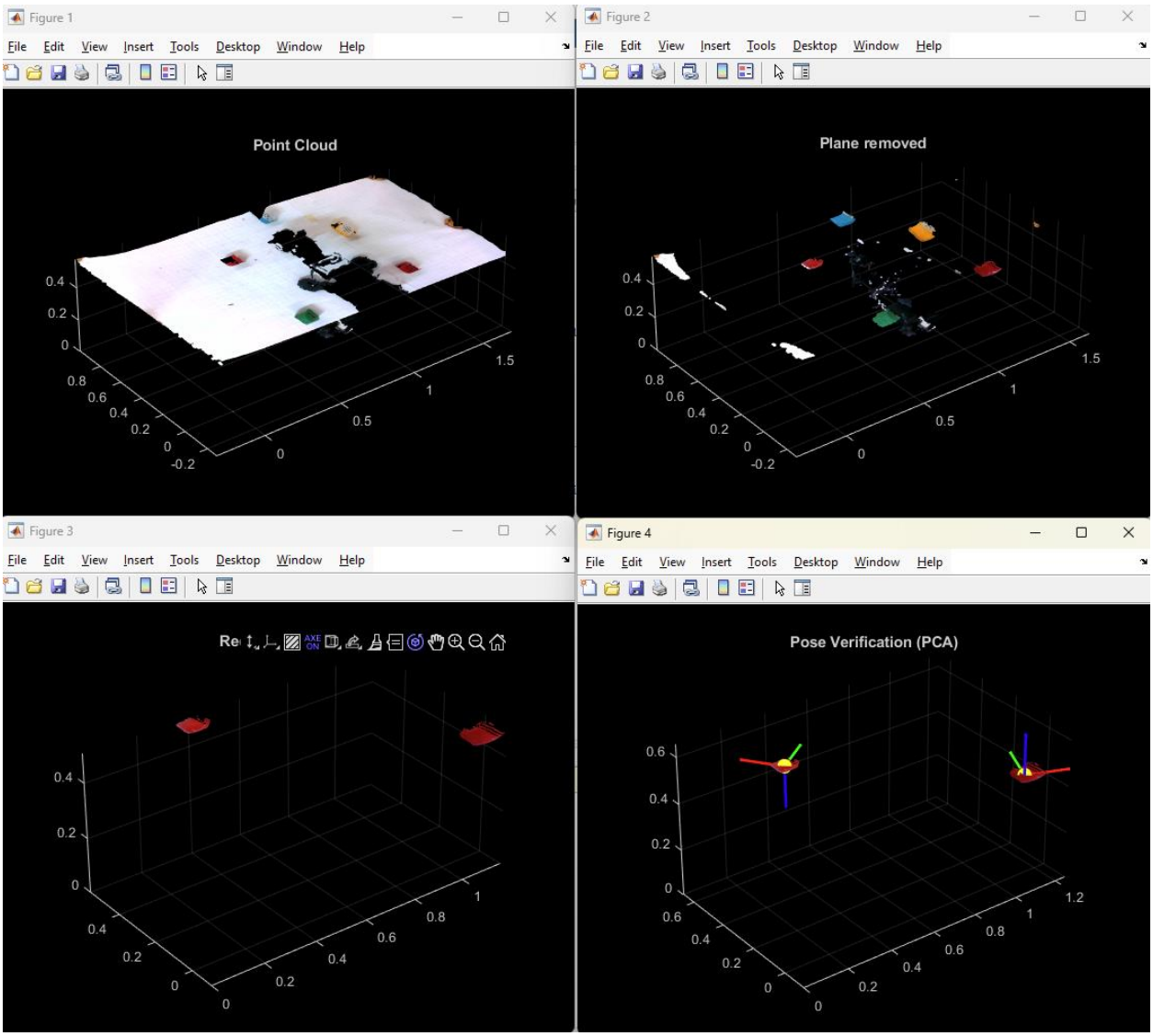
c. What were the test results? Were the tests a success?

In nominal conditions:

- Plane fitting successfully removed the table in all trials.
- HSV segmentation correctly extracted the red blocks.
- Euclidean clustering separated multiple cubes reliably.
- The centroid visually aligned with the geometric center of each cube.
- PCA axes consistently aligned with cube edges.
- Position estimates showed small variation between runs (minor millimeter-level fluctuations due to depth noise).
- Orientation remained stable with no random axis flipping after enforcing the right-handed constraint.

All nominal tests were considered successful since the detected pose was visually correct and stable across repetitions.

The output was seen as follows:



In repositioning tests:

Results showed that object location did not affect pose estimation accuracy, provided the cube remained within good depth sensing range and sufficiently illuminated. Centroid and orientation remained consistent and reliable.

In non-nominal case:

- Some pixels failed the HSV threshold, resulting in small holes in the segmented cloud.
- The number of segmented points decreased.
- Centroid estimation remained reasonably accurate because enough points were still available.
- PCA orientation showed slightly higher variance but still aligned approximately with cube faces.

The system degraded gracefully rather than failing completely. Detection still worked, but segmentation quality was visibly reduced.

Another non-nominal test involved placing cubes closer than the clustering threshold distance. In this case, clusters occasionally merged into a single group, causing incorrect separation. This highlighted sensitivity to the minDistance parameter in pcsegdist.

Overall results

The pipeline performed reliably under nominal conditions and moderately robustly under mild lighting changes and minor segmentation loss. Failures were mainly caused by strong lighting variation or insufficient separation between objects.

The tests confirm that the implemented perception pipeline is suitable for stable pose estimation of colored cubes in controlled indoor environments, and that its assumptions (planar table, sufficient lighting, visible surfaces) are critical for consistent performance.l

d. Details of at least one test that pushes the system outside nominal conditions, e.g. changing lighting, partial occlusions. What was changed? What happened?

Non-nominal condition test: Yellow colored block was placed very close to red block and placed diagonally

An edge test was done to test robustness of the pipeline by placing a yellow-colored cube very close to a red, yellow cube positioned diagonally relative to the camera frame. This configuration pushed the system outside its nominal assumptions of clear spatial separation between cubes and minimal color interference.

First, although HSV segmentation correctly isolated red regions, small segmentation noise appeared near the boundary between the red and yellow cubes due to both colors being connected and lighting reflection. Because the cubes were very close, clustering struggled to clearly separate the two objects. In some runs, the red cluster included stray points near the yellow cube, slightly distorting the cluster's shape.

Second, the diagonal orientation reduced the symmetry of visible points. Since PCA relies on the spatial distribution of points, incomplete or skewed surface visibility caused slight shifts in centroid estimation and minor deviations in alignment. While a transformation matrix was still generated, there were a few small orientation errors compared to nominal cases where cubes were well separated and axis aligned.

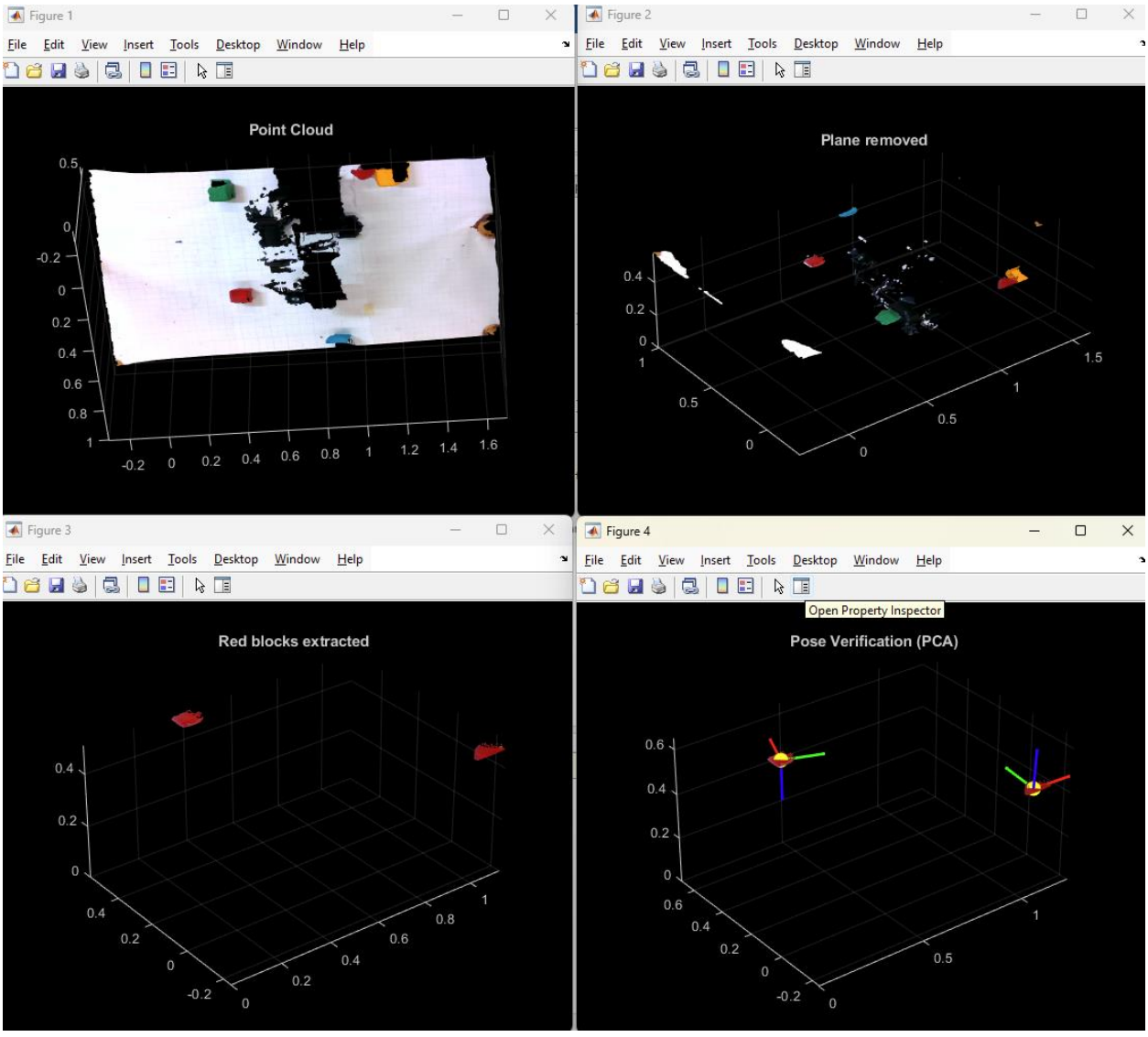
This test highlights two important assumptions in the pipeline:

1. The objects of interest are sufficiently separated in 3D space for clustering.

2. A balanced distribution of cube surface points is visible for stable PCA estimation.

   When these assumptions are weakened, the system does not fail completely, but the pose accuracy degrades.

   The results seen are as follows:

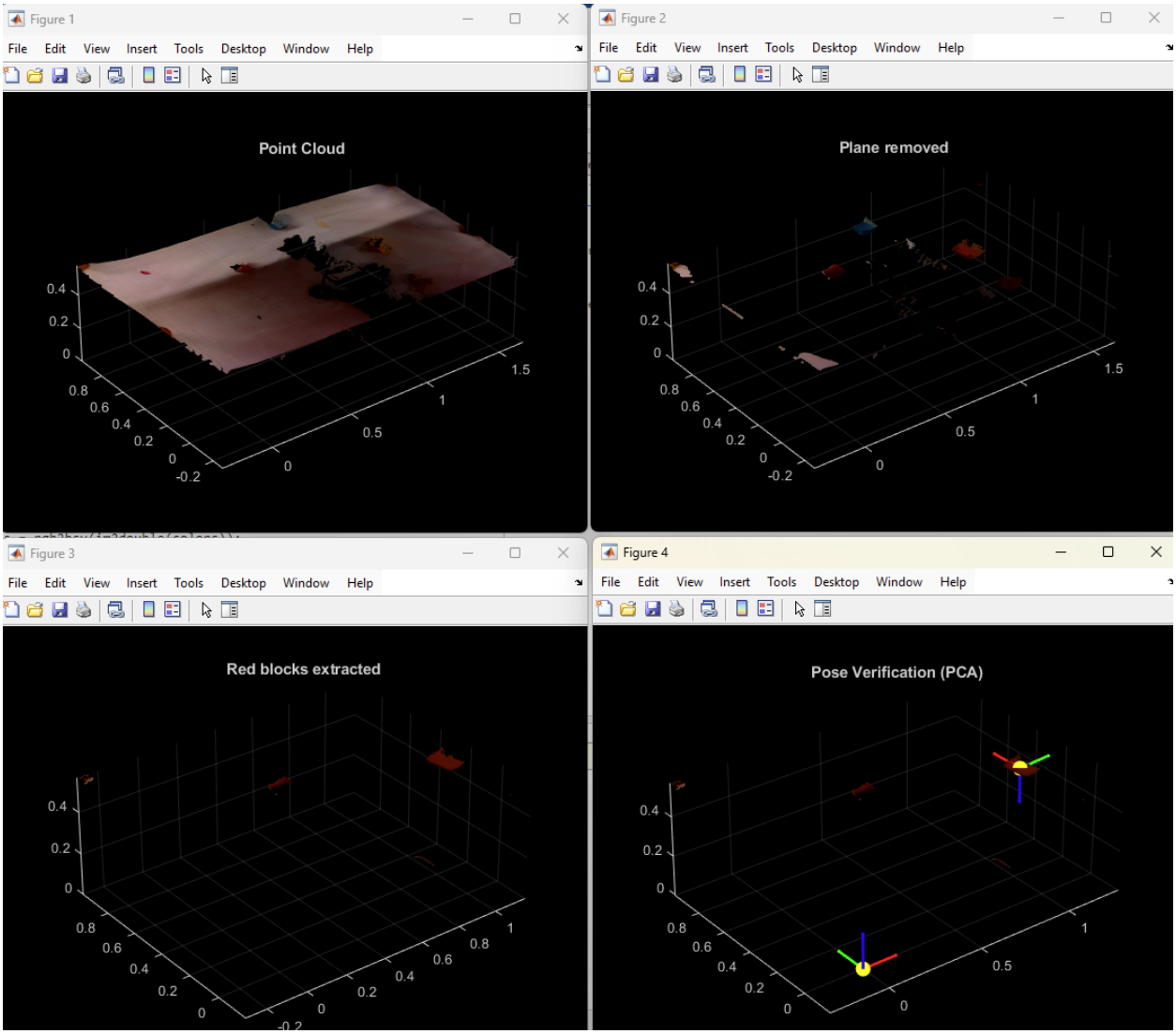


4. Failure Models and Limitations:
   a. Concrete failure case and its identification

A major failure case occurred in very poor lightning and strong shadow on the workspace. When the cube was placed under dense shadow, the HSV color segmentation failed to correctly classify

the cube and large portions of the cube were missing from the mask. The point of cloud after plane removal was sparse and fragmented. Clustering either failed or produced very small clusters, and PCA could not compute stable principal axes due to insufficient valid 3D points. The perception pipeline failed at color segmentation, causing a domino effect of failures in the end resulting in wrong pose estimation.

The results of the failure case were:



b. Reason about the cause by explicitly highlighting the specific failed pipeline assumption or algorithmic choice.

This highlights an algorithmic limitation. The pipeline assumes relatively uniform lighting; it also assumes that cube surfaces are sufficiently exposed and that HSV thresholds remain valid

across the workspace therefore, no adaptive thresholding or illumination normalization is implemented.

c. Highlight how this failure would not affect downstream tasks during pick and place.

While this type of failure can cause incorrect centroid positioning, resulting in grasp position error, this does not affect our perception pipeline because the pipeline must simply assume that it has ambient lighting to work with. In downstream tasks, we must simply always ensure that we provide sufficient lighting or increase the robustness of our pipeline to be able to handle low light settings.

This type of failure will directly affect downstream pick-and-place tasks. Since a transformation matrix is still generated, the robot interprets it as valid. An incorrect centroid results in grasp position error, which may cause the gripper to miss the cube or grasp it off-center. An incorrect orientation matrix may cause the end-effector to align improperly with the cube's faces, increasing the risk of slippage or collision. Unlike a complete detection failure where the robot would simply not move, this case is more critical because it produces a wrong pose estimate, which can lead to manipulation errors.

5. Next Steps:
    a. Changes you would make for increased robustness in your pipeline.

To improve robustness of the perception pipeline, several modifications can be made at different stages of the algorithm.

- The color segmentation stage can be improved by introducing adaptive thresholding instead of fixed HSV ranges. The current implementation assumes consistent lighting, which makes it sensitive to brightness variation and shadows. Applying illumination normalization, histogram equalization on the V channel, or dynamically tuning HSV bounds based on scene statistics would reduce segmentation failure under non-uniform lighting.
- Morphological operations such as closing and hole filling can be applied after mask creation. This would reduce small holes in the segmented object and improve the stability of the centroid and PCA orientation estimation by increasing the number of valid points.
- Clustering robustness can be improved by automatically tuning the minDistance parameter in pcsegdist based on point density or expected cube size. This would reduce the risk of cluster merging when cubes are placed close to each other.
- Instead of relying purely on PCA for orientation, constraints based on cube geometry can be incorporated. Since the cube lies on a planar surface, one axis should be approximately perpendicular to the table plane. Enforcing this constraint would stabilize orientation estimation in cases of partial occlusion or uneven point distribution.

- A confidence metric can be added before sending pose to downstream modules. For example, checking minimum number of segmented points, validating cluster size against expected cube dimensions, or verifying orthogonality of PCA axes can help detect unreliable estimates. If confidence is low, the system can reject the pose instead of forwarding an incorrect transformation matrix.

b. Which assumptions are you most worried about going forward?

- Uniform and sufficient lighting. Since segmentation depends heavily on HSV thresholds, strong shadows or reflections can significantly degrade performance.
- Cubes are well separated in 3D space. If objects are too close, clustering may merge them, leading to incorrect pose estimation.
- A large portion of the cube surface is visible to the camera. Severe occlusions reduce point density and affect PCA stability.
- The method assumes that the table is the dominant planar surface in the scene. If additional planar objects exist or if the table is partially occluded, plane fitting may remove incorrect regions.

Improving robustness against these assumptions would be the primary focus in the next development stage.

References:

[1] N. W. Christian and F. T. Sabilillah, "Color Detection Using Webcam with Matlab HSV Color Model Segmentation Based," *(IJECAR) International Journal of Engineering Computing Advanced Research*, vol. 1, no. 1, pp. 43–51, Jul. 2024, Accessed: Feb. 12, 2026. [Online]. Available: https://journals.arces.org/ijecar/article/viewFile/11/7