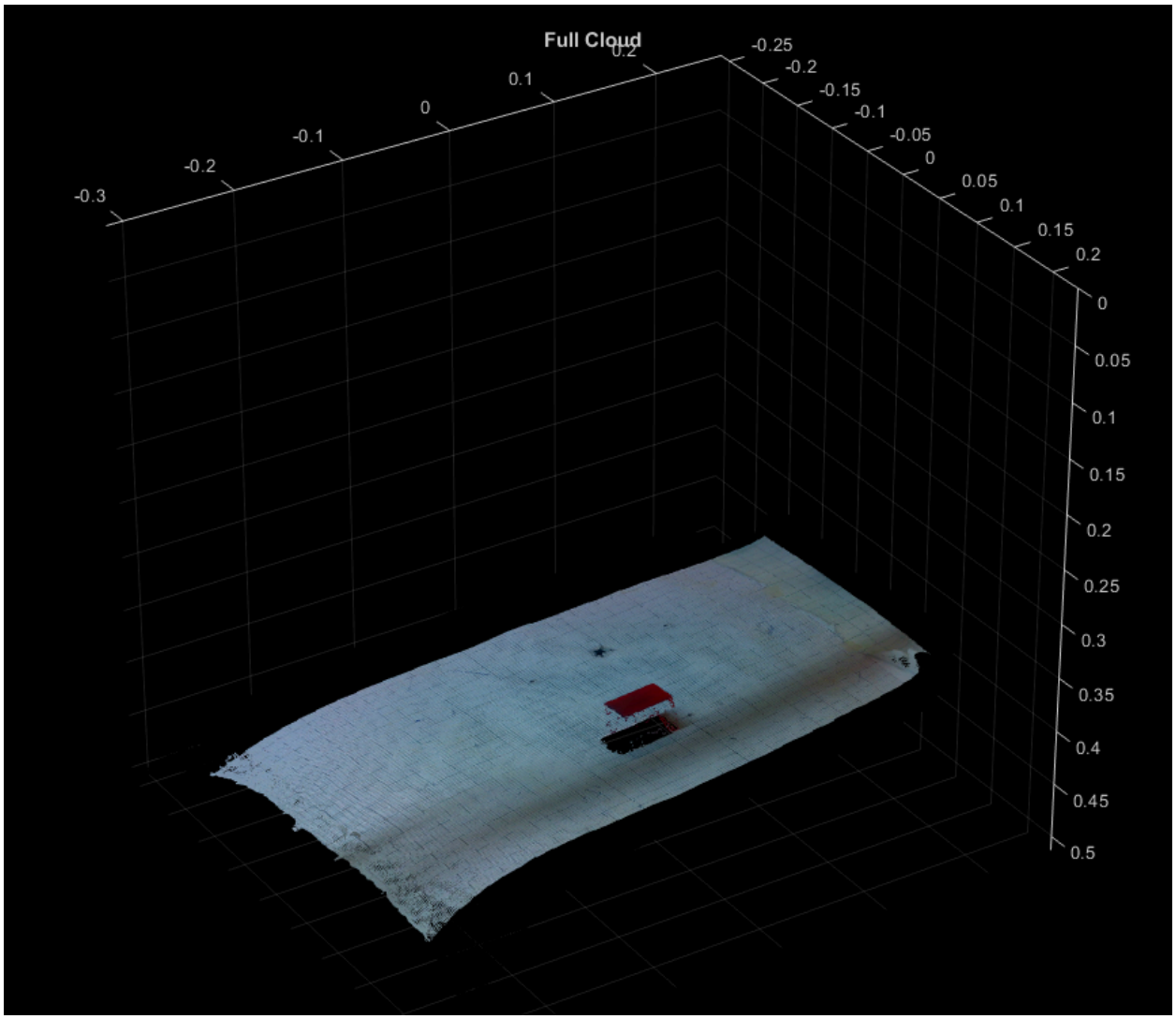
Perception Pipeline

Fatima Adamjee, Nabiha Hasnain, Yusra Shahid

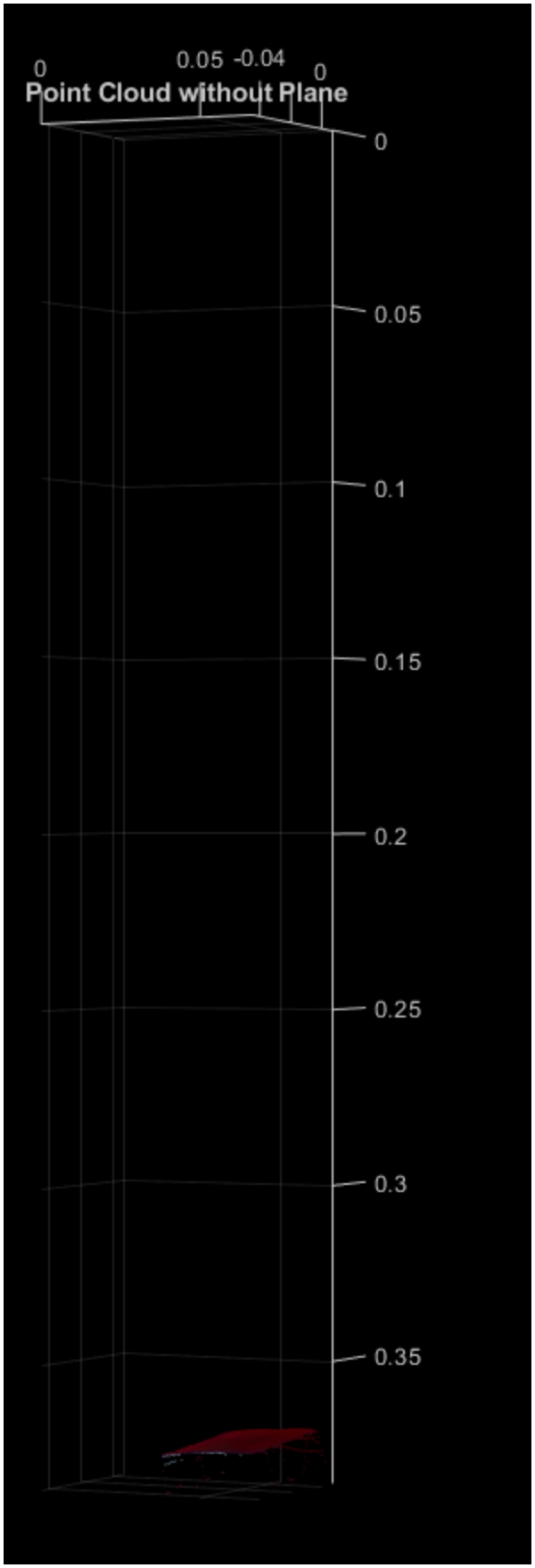
Appraoch: Using point cloud to detect block and find its orientation

Step 1: Extracting the point cloud:



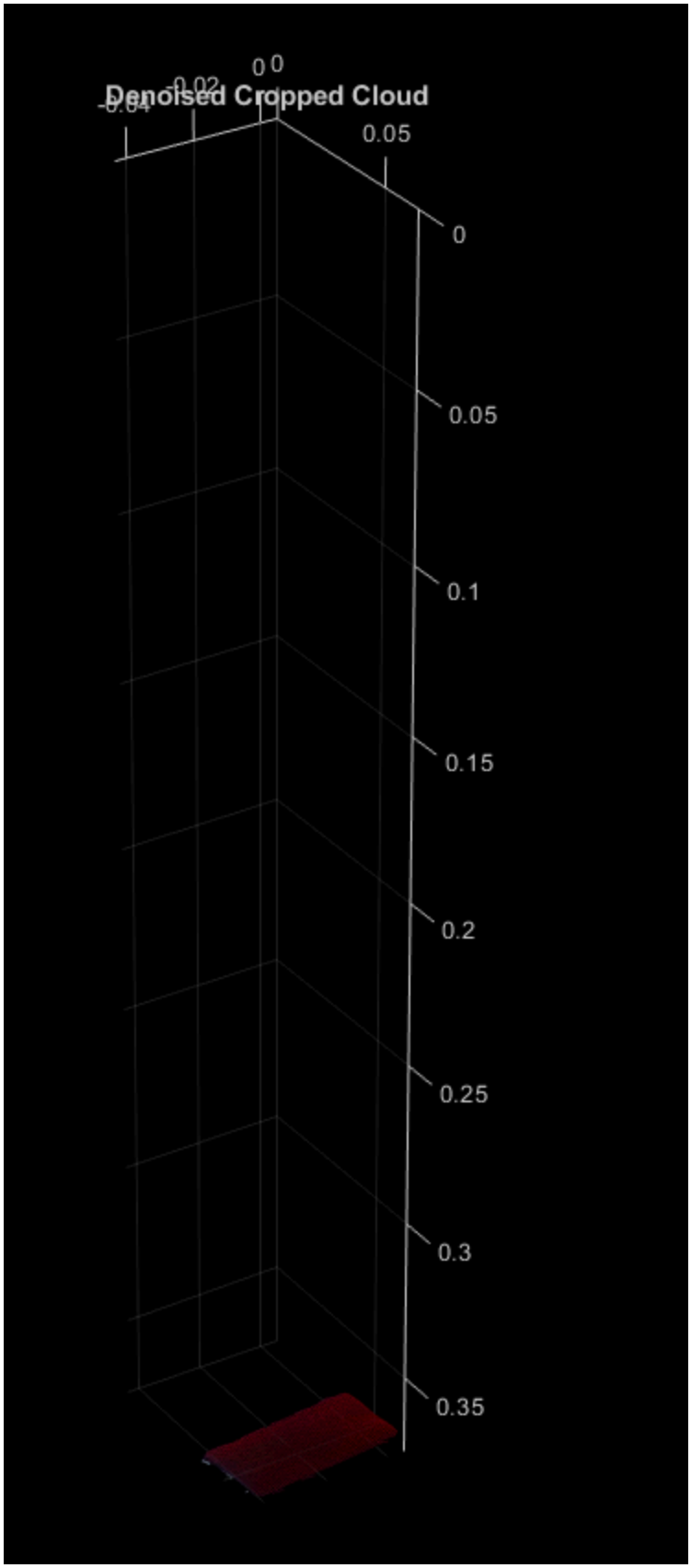
```
function pointcloud_example4()
% Make Pipeline object to manage streaming
pipe = realsense.pipeline();

% 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

% 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);


figure; pcshow(ptCloud,'VerticalAxisDir','Down'); title('Full Cloud');
```

Step 2: using ROI crop to remove table plane:


0
0.05
-0.04
0
Point Cloud without Plane
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

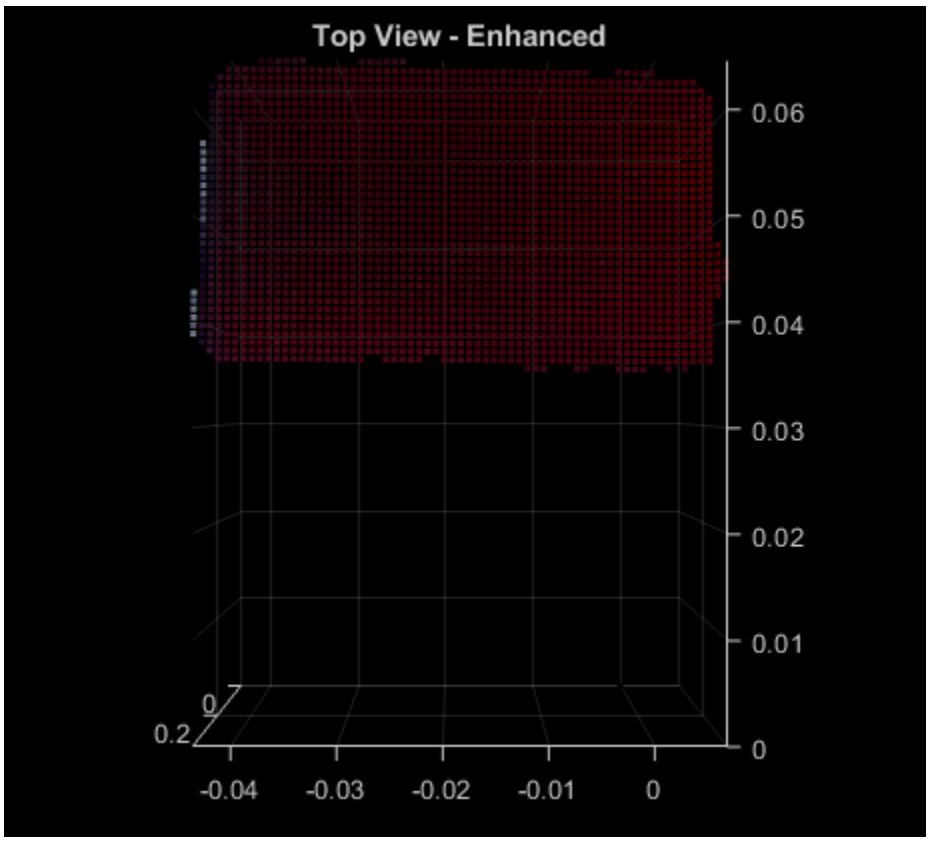
```
xLimits = [-0.1 0.1];   % left-right range
yLimits = [0 0.10];   % front-back range

% Keep ALL Z values
zLimits = [0.2 0.385];

roi = [xLimits yLimits zLimits];

% Get indices inside ROI
indices = findPointsInROI(ptCloud, roi);

% Extract cropped cloud
croppedCloud = select(ptCloud, indices);
pcwrite(croppedCloud, "croppedcloud1.pcd")
save( "mcropcloud1", "croppedCloud")

% Show result
figure
pcshow(croppedCloud, 'VerticalAxisDir','Down');
title('Point Cloud without Plane');
```

Step 3: Denoising the cropped point cloud

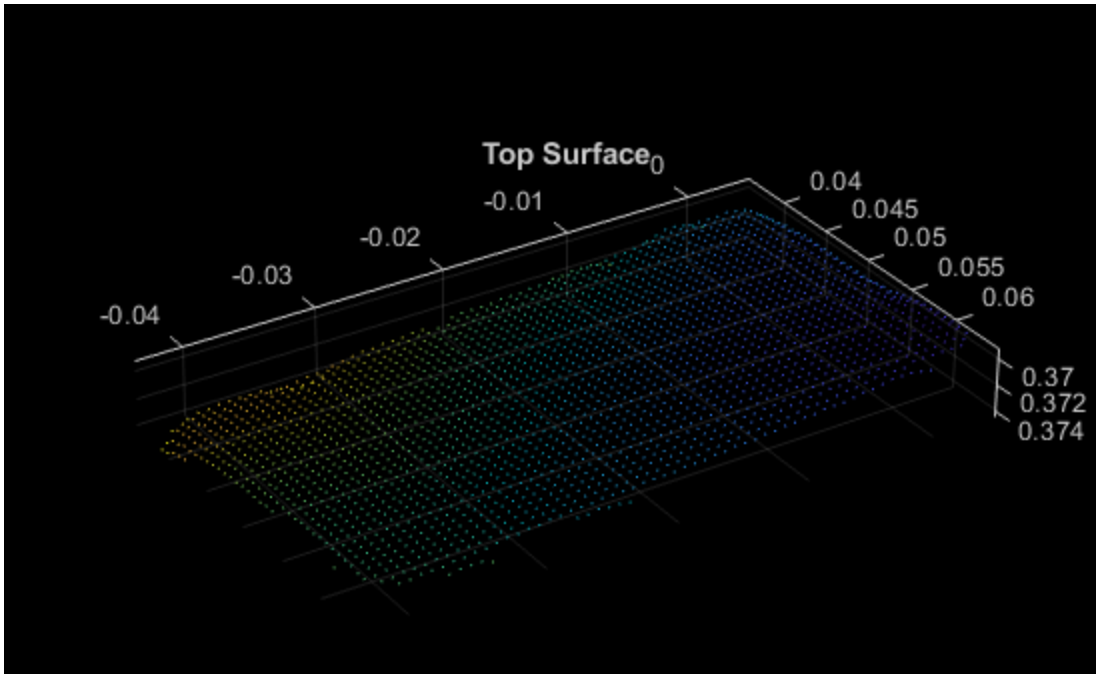

Denoised Cropped Cloud
-0.04
-0.02
0
0
0.05
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

```
% --- DENOISE CROPPED CLOUD ---
croppedCloud = pcdenoise(croppedCloud, ...
                         'NumNeighbors', 20, ...
                         'Threshold', 1);
```

Step 4:extracting Top Surface:



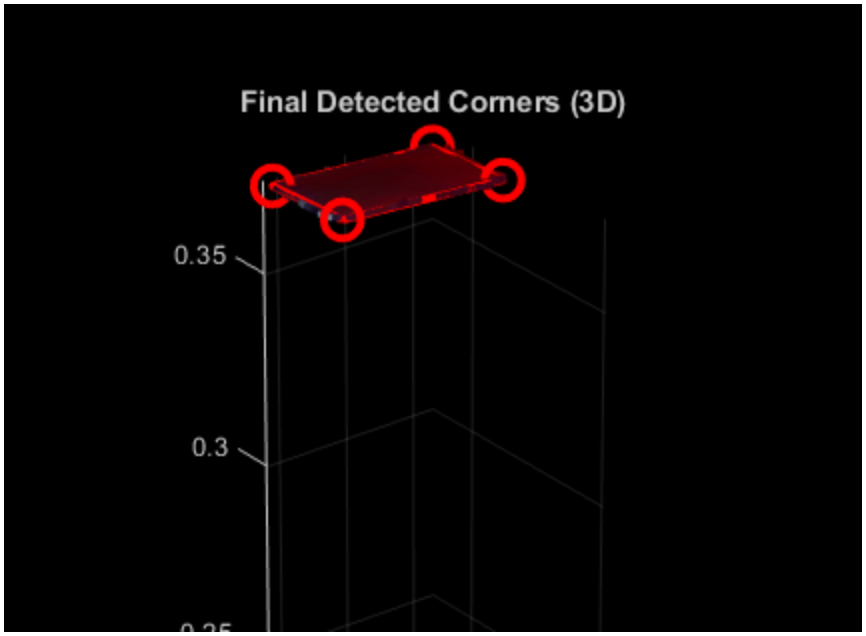
```
xyz = croppedCloud.Location;
xyz = reshape(xyz, [], 3);
xyz = xyz(~any(isnan(xyz),2), :);
maxDistance = 0.0015; % 3 mm tolerance
% --- EXTRACT NUMERIC XYZ ---
xyzAll = croppedCloud.Location;
xyzAll = reshape(xyzAll, [], 3);
xyzAll = xyzAll(~any(isnan(xyzAll),2), :);

% --- KEEP ONLY TOP 20% HIGHEST POINTS ---
zThreshold = prctile(xyzAll(:,3), 80);
xyzTopCandidate = xyzAll(xyzAll(:,3) > zThreshold, :);

% Build temporary point cloud from candidates
topCandidateCloud = pointCloud(xyzTopCandidate);

% Now fit plane ONLY to highest region
[modelTop, inliersTop] = pcfitplane(topCandidateCloud, 0.0015);

topCloud = select(topCandidateCloud, inliersTop);

topCloud = select(croppedCloud, inliersTop);
figure
pcshow(topCloud, 'VerticalAxisDir','Down');
axis equal
title('Top surface');
 figure
pcshow(croppedCloud, ...
       'VerticalAxisDir','Down', ...
       'MarkerSize', 60);
```

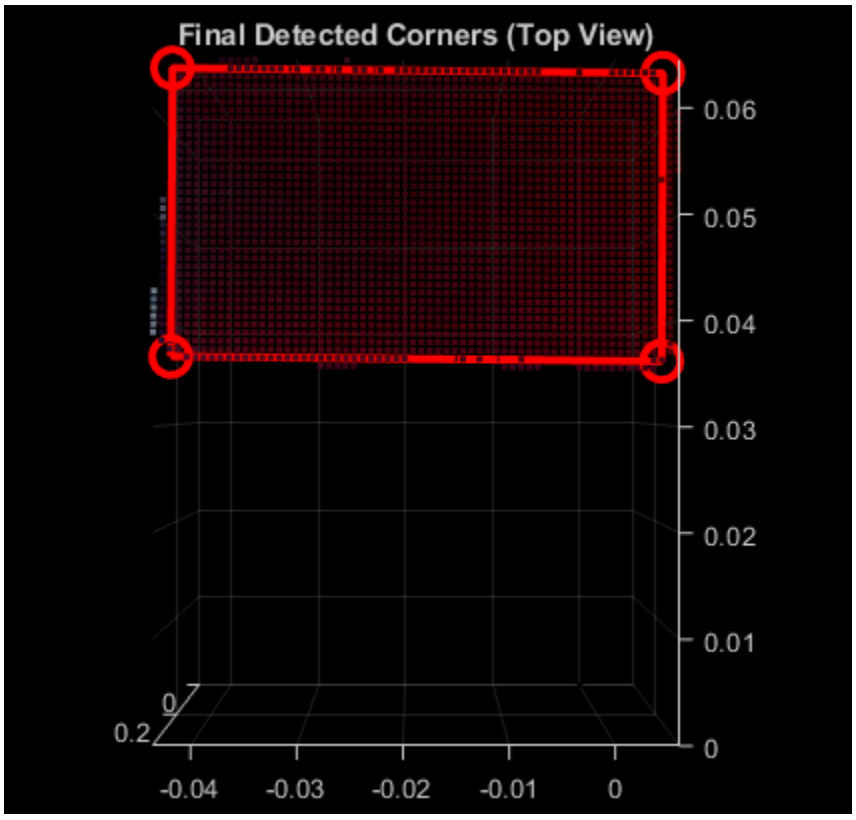



```
%% --- Extract XYZ cleanly from croppedCloud ---
xyzAll = croppedCloud.Location;
xyzAll = reshape(xyzAll, [], 3);
xyzAll = xyzAll(~any(isnan(xyzAll), 2), :);
```

```
%% --- Fit plane to TOP points and keep correct indices ---
zThreshold = prctile(xyzAll(:,3), 80);
topMask = xyzAll(:,3) > zThreshold;
xyzTopCandidate = xyzAll(topMask, :);

topCandidateCloud = pointCloud(xyzTopCandidate);
[modelTop, inliersTop] = pcfitplane(topCandidateCloud, 0.0015);

topCloud = select(topCandidateCloud, inliersTop);

figure; pcshow(topCloud, 'VerticalAxisDir','Down'); axis equal; title('Top Surface');
```

Step 5:Detecting corners:


Final Detected Corners (3D)
0.35
0.3


Final Detected Corners (Top View)
0.06
0.05
0.04
0.03
0.02
0.01
0
0
0.2
-0.04
-0.03
-0.02
-0.01
0

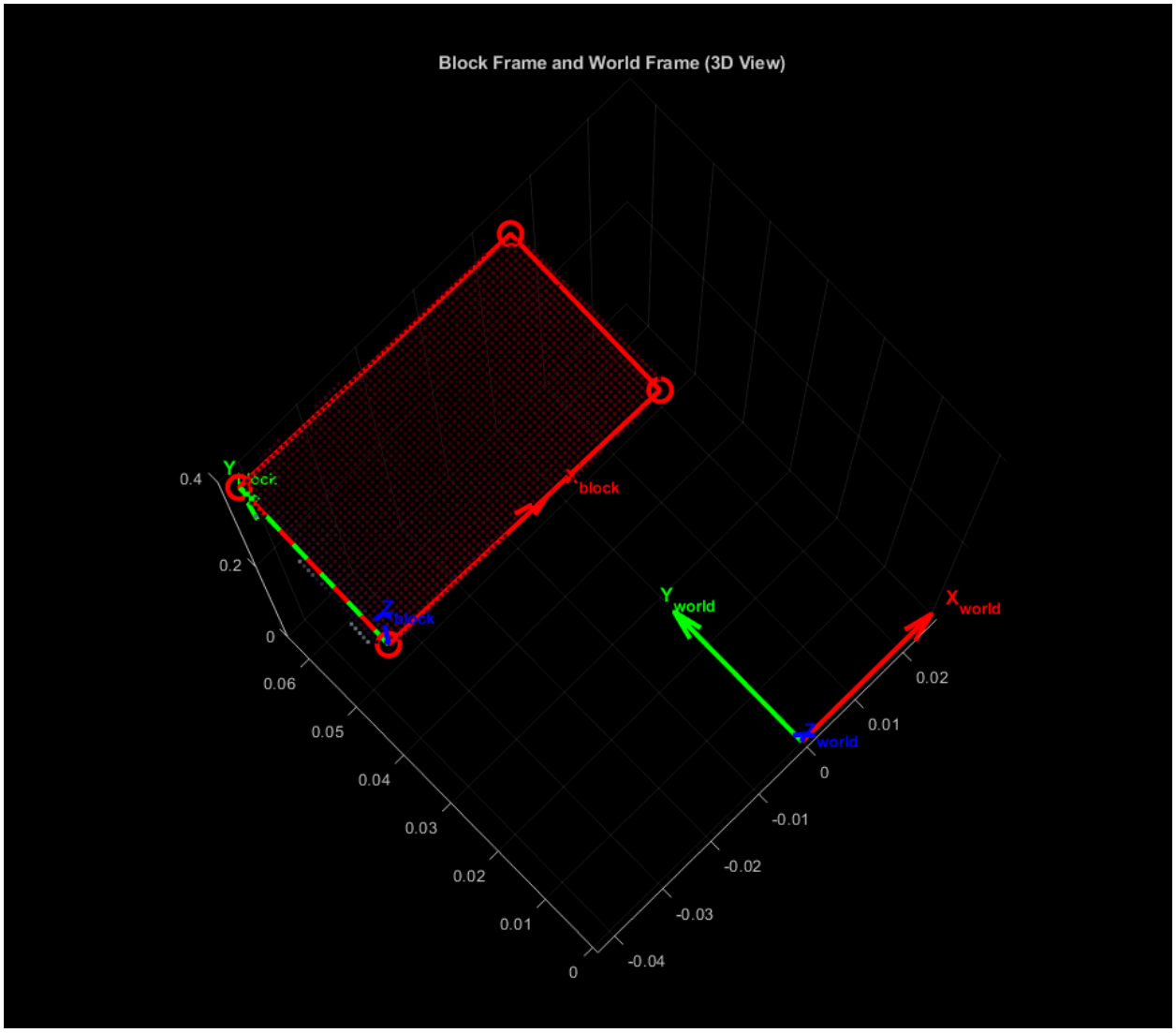
```
xyzTop = topCloud.Location;
xyzTop = reshape(xyzTop, [], 3);
xyzTop = xyzTop(~any(isnan(xyzTop), 2), :);

centroid = mean(xyzTop, 1);
centered = xyzTop - centroid;

% PCA: columns of coeff are principal axes
[coeff, ~, ~] = pca(centered);

% FIX: Ensure the 3rd axis (plane normal) points consistently
if coeff(3,3) < 0
    coeff(:,3) = -coeff(:,3);
end

% FIX: Also ensure axes 1 and 2 are consistently oriented (avoid flipping)
if coeff(1,1) < 0
    coeff(:,1) = -coeff(:,1);
end
if coeff(2,2) < 0
    coeff(:,2) = -coeff(:,2);
end

% Project all top points into PCA frame
rotated = centered * coeff;  % N x 3

% Robust bounding box using tighter percentiles to reject outliers
xmin = prctile(rotated(:,1), 2);
xmax = prctile(rotated(:,1), 98);
ymin = prctile(rotated(:,2), 2);
ymax = prctile(rotated(:,2), 98);
```

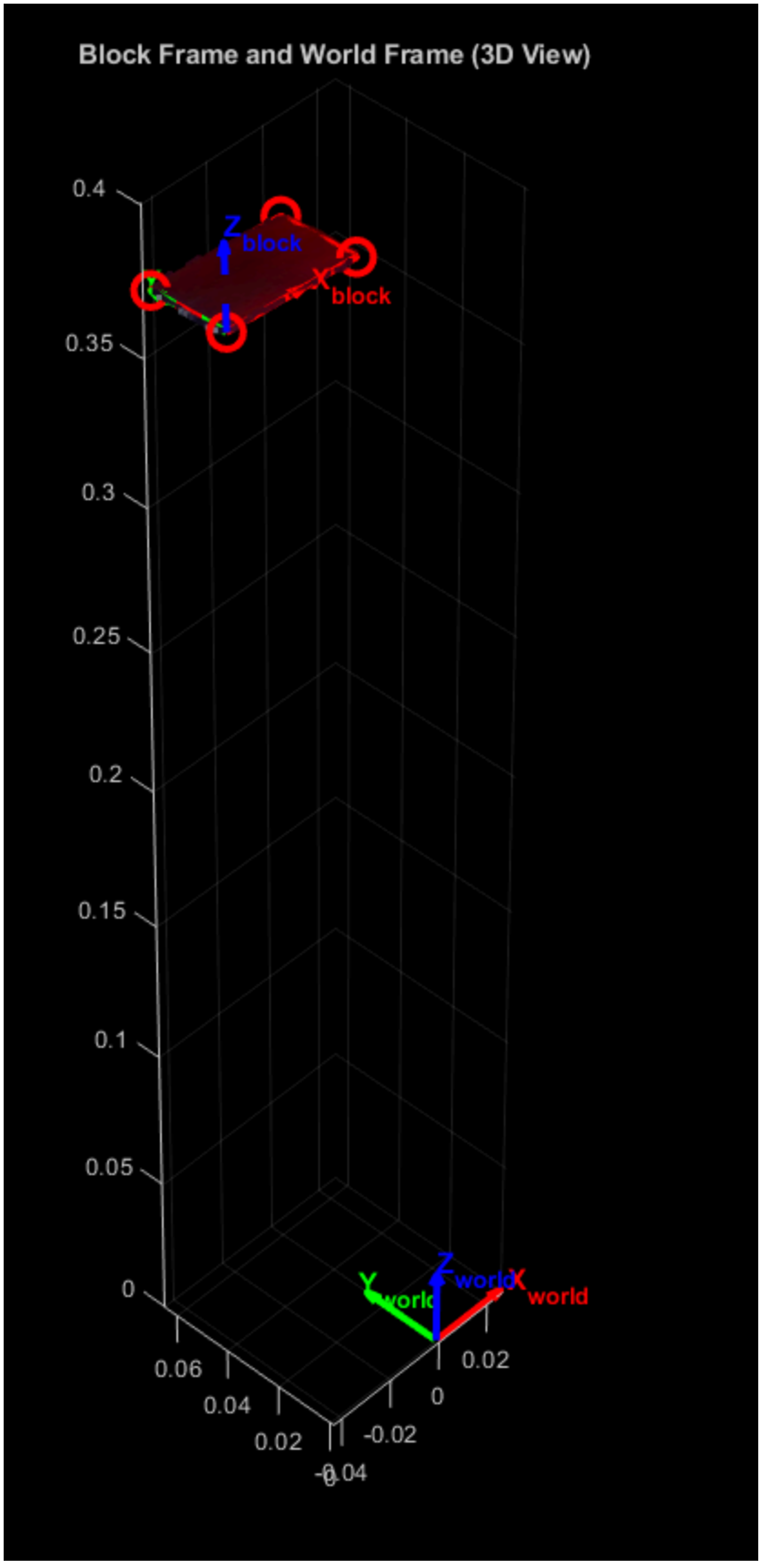
```
%% Build corners in PCA frame, then unrotate correctly ---
% Z=0 in PCA frame = centroid plane; we'll project onto actual plane
corners_rot = [
    xmin, ymin, 0;
    xmax, ymin, 0;
    xmax, ymax, 0;
    xmin, ymax, 0
];

% Unrotate: corners in camera frame (before plane projection)
corners_temp = corners_rot * coeff' + centroid;
```

```
%% Project corners onto actual fitted plane ---
A = modelTop.Parameters(1);
B = modelTop.Parameters(2);
C = modelTop.Parameters(3);
D = modelTop.Parameters(4);

for k = 1:4
    x = corners_temp(k,1);
    y = corners_temp(k,2);
    % Ax + By + Cz + D = 0  =>  z = -(Ax + By + D) / C
    corners_temp(k,3) = -(A*x + B*y + D) / C;
end
corners = corners_temp;
```

```
%% Close the rectangle for plotting ---
cornersPlot = [corners; corners(1,:)];  % close the loop

% Compute side lengths to verify aspect ratio
side1 = norm(corners(2,:) - corners(1,:));
side2 = norm(corners(3,:) - corners(2,:));
fprintf('Corner side lengths: %.4f m  x  %.4f m\n', side1, side2);
fprintf('Corners (camera frame XYZ):\n');
disp(corners);
```

Step 6: Making world and block frames:

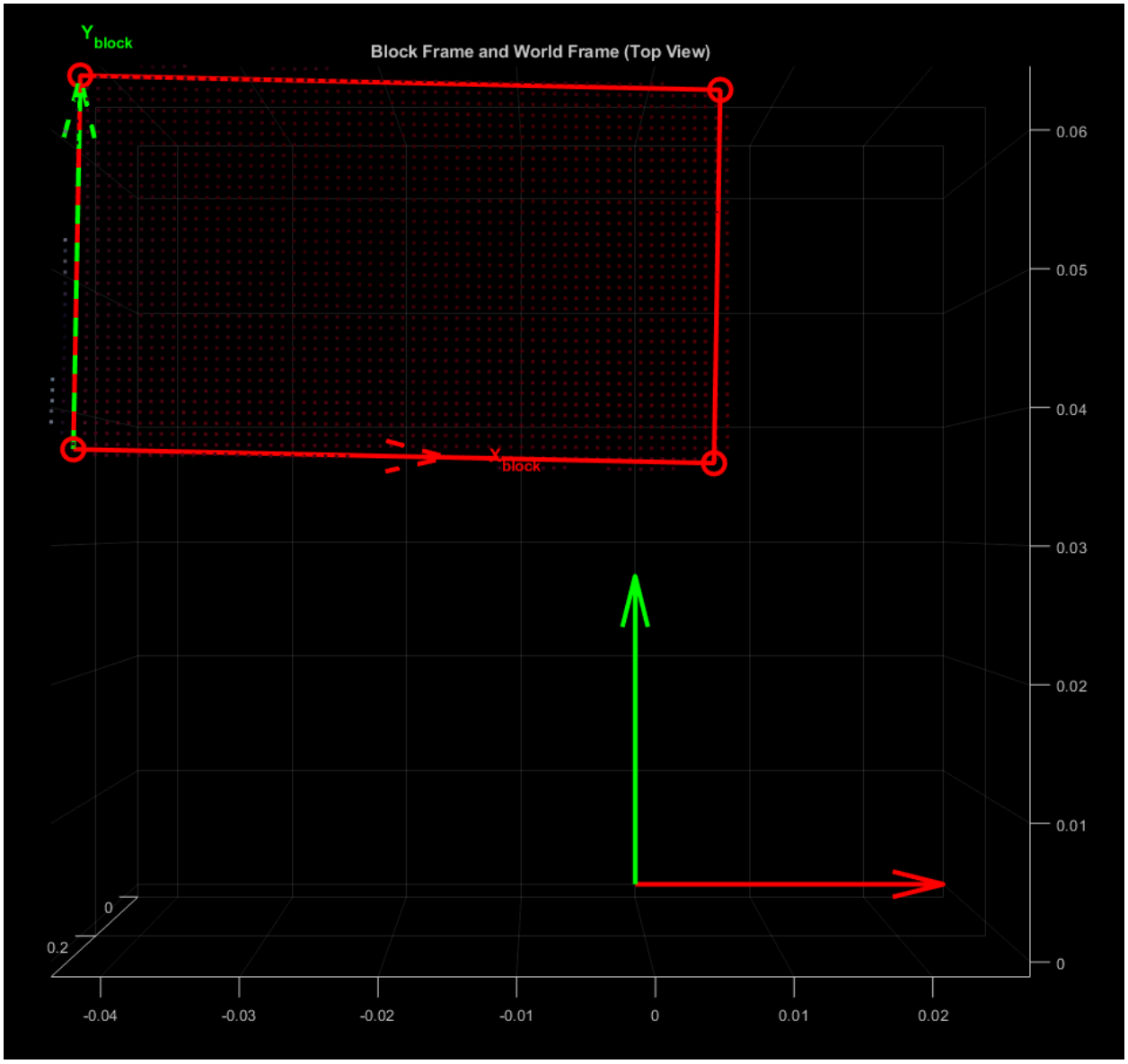

Block Frame and World Frame (3D View)
Y
block
X
block
Z
block
Y
world
X
world
Z
world
0.4
0.2
0
0.06
0.05
0.04
0.03
0.02
0.01
0
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04


Block Frame and World Frame (3D View)
Z block
X block
Y world
Z world
X world
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.06
0.04
0.02
-0.02
-0.04
0.02
0

```
    %% ========== DEFINE BLOCK FRAME ==========
% Origin: Corner 1 (you can choose any corner: 1, 2, 3, or 4)
block_origin = corners(1, :);

% X-axis: along edge from corner 1 to corner 2
block_x_axis = (corners(2,:) - corners(1,:)) / norm(corners(2,:) - corners(1,:));

% Y-axis: along edge from corner 1 to corner 4
block_y_axis = (corners(4,:) - corners(1,:)) / norm(corners(4,:) - corners(1,:));

% Z-axis: perpendicular to top surface (pointing upward)
% Use the plane normal from fitted plane
block_z_axis = [A, B, C] / norm([A, B, C]);

% Ensure Z points upward (positive Z direction)
if block_z_axis(3) < 0
    block_z_axis = -block_z_axis;
end
```

```
%% ========== DEFINE WORLD FRAME ==========
% World frame origin at (0, 0, 0) with standard axes
world_origin = [0, 0, 0];
world_x_axis = [1, 0, 0];
world_y_axis = [0, 1, 0];
world_z_axis = [0, 0, 1];
```

Step 7: Finding Rotation matrix and Homogenous Transformation matrix:




```
% Construct rotation matrix: columns are the axes
R_block = [block_x_axis', block_y_axis', block_z_axis'];

% Homogeneous transformation matrix in METERS
T_block_meters = eye(4);
T_block_meters(1:3, 1:3) = R_block;
T_block_meters(1:3, 4) = block_origin';

% Homogeneous transformation matrix in CENTIMETERS
T_block_cm = eye(4);
T_block_cm(1:3, 1:3) = R_block;  % Rotation stays the same
T_block_cm(1:3, 4) = block_origin' * 100;  % Convert translation to cm
```

This Pipeline current;y works for only one block for multiple blocks we would apply segmentation, color masking and clustering