Introduction to Robotics

Lab 2b

Shaaf Farooque 08405, Mysha 08443

Task 2.12 Camera Intrinsic Parameters (10 points)

The MATLAB function `determineIntrinsics()`, provided on LMS, uses Intel's SDK[a] to display the intrinsic parameters of the color camera. Elaborate each entry in light of acquired knowledge from class and https://github.com/IntelRealSense/librealsense/wiki/Projection-in-RealSense-SDK-2.0.

[a] Intel provides an SDK ([7]) for its RealSense line of cameras. The SDK includes a MATLAB wrapper and we'll be making use of it in our project. You can install the SDK by using the provided installer. Make sure to check the MATLAB Developer Package| during installation. The package will be installed to C:/Program Files (x86)/Intel RealSense SDK 2.0/matlab/realsense/|. You will have to add the RealSense library to your MATLAB path. You can do this by navigating to 'Set Path' in the 'Home' ribbon on MATLAB and 'Add with subfolders' the library directory .

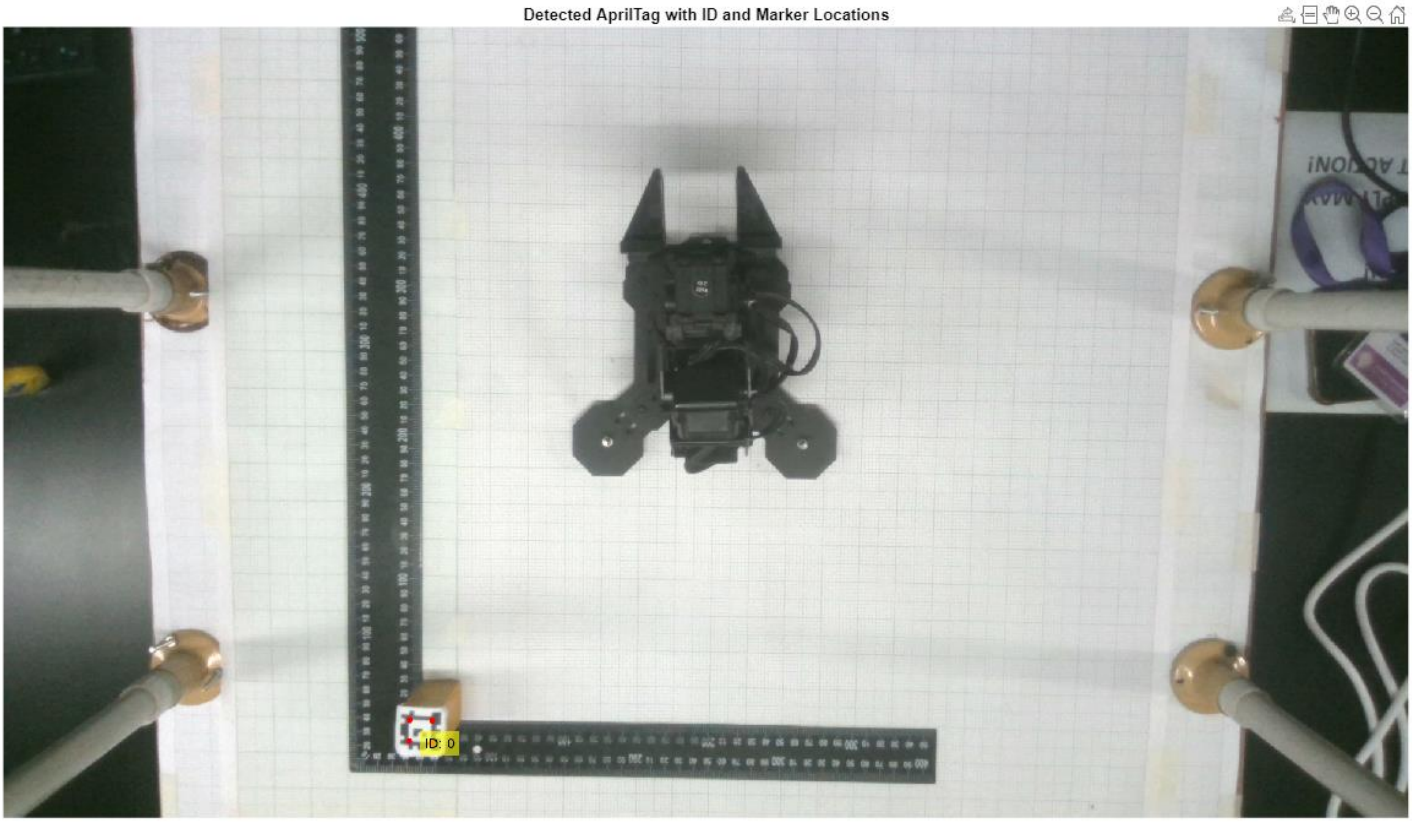
```
color_intrinsics =

  struct with fields:

      width: 1920
     height: 1080
        ppx: 951.5109
        ppy: 526.6981
         fx: 1.4053e+03
         fy: 1.4053e+03
      model: 0
     coeffs: [0 0 0 0 0]
```

The width and height tell the dimensions of the image in pixels. The RGB camera takes photos of size 1920x1080. Ppx and ppy tell the co-ordinates of the principal axes. Fx and fy tell the focal length. The model: 0 tells us the distortion model of the camera. A value of 0 typically corresponds to the Brown-Conrady distortion model, which includes radial and tangential distortion. The coeffs array contains distortion coefficients. The coefficients usually represent: k1 ,k2 ,k3 (Radial distortion), p1 ,p2 (Tangential distortion). Since all values are 0, it suggests that this camera model does not have distortion or has already been corrected.

Task 2.13 Detect AprilTags (10 points)

Use the MATLAB function `readAprilTag` to obtain the pose of an AprilTag placed on the top face of a cube. Use a ruler to physically measure the frame transformation and verify the accuracy of the result obtained from the MATLAB function.

```
focalLength = [1.4053e+03, 1.4053e+03];

% Define principal point (cx, cy)
principalPoint = [951.5109, 526.6981];

% Define distortion coefficients (assuming radial and tangential coefficients)
distortionCoefficients = [0 0 0 0 0];

% Define image size
imageSize = [1080, 1920];

% Create cameraIntrinsics object
intrinsics = cameraIntrinsics(focalLength, principalPoint, imageSize, 'RadialDistortion', distortionCoefficients(1:3), ...
    'TangentialDistortion', distortionCoefficients(4:5));

% Read the image containing the AprilTag
p1 = imread("aprilTagp1.png");

% Define the tag family
tagFamily = "tagStandard41h12";

% Define the actual tag size in meters (3 cm = 0.03 m)
tagSize = 0.015;

% Detect the AprilTag and estimate pose
[id, loc, pose] = readAprilTag(p1, tagFamily, intrinsics, tagSize);

% Insert markers to indicate the detected tag locations
markerRadius = 4;
I = p1; % Copy the original image

for idx = 1:length(id)
    numCorners = size(loc, 1);
    markerPosition = [loc(:,:,idx), repmat(markerRadius, numCorners, 1)];
    I = insertShape(I, "FilledCircle", markerPosition, "ShapeColor", "red", "Opacity", 1);

    % Display tag ID as text on the image
    tagCenter = mean(loc(:,:,idx), 1); % Compute the center of the tag
    I = insertText(I, tagCenter, sprintf("ID: %d", id(idx)), "FontSize", 18, ...
                   "BoxColor", "yellow", "TextColor", "black");
end

% Display the result
imshow(I);
title("Detected AprilTag with ID and Marker Locations");
```




```
% Display the estimated pose for each tag
for idx = 1:length(id)
    fprintf("Tag ID: %d\n", id(idx));
    disp(pose(idx)); % Pose contains translation and rotation info
end
```

```
Tag ID: 0
  rigidtform3d with properties:

    Dimensionality: 3
       Translation: [-0.1867 0.2124 0.6871]
                 R: [3×3 double]

                 A: [0.8969    0.1323   -0.4221   -0.1867
                     0.1765    0.7678    0.6158    0.2124
                     0.4056   -0.6268    0.6653    0.6871
                          0         0         0    1.0000]
```

The translation vector [-0.1867 0.2124 0.6871] is telling us the x y and z distances from the center of the camera. Using a ruler and the depth camera, the values are verified to be approximately correct.

Task 2.14 Camera datasheet (0 points)

From the provided datasheet [8], determine the values of the following and provide an explanation for each quantity:

(a) Resolution of color camera and IR camera;

(b) Frame rates of both cameras;

(c) Depth field of view;

(d) Depth start point.

(a) Resolution of Color Camera and IR Camera

RGB Camera: 1920×1080

This represents the maximum resolution of the color (RGB) camera.

IR Camera: 640×480 (VGA)

This is the resolution of the infrared (IR) camera, which is lower than the RGB camera.

(b) Frame Rates of Both Cameras

RGB Camera: 10 FPS, 30 FPS

The RGB camera supports two frame rates: 10 frames per second (FPS) and 30 FPS.

10 FPS is useful for low-power applications or when real-time processing isn't critical.

30 FPS provides smoother motion, making it better for video streaming and real-time applications.

IR Camera: 10 FPS, 30 FPS, 60 FPS

The IR camera supports an additional 60 FPS mode, which is useful for improved depth precision.

(c) Depth Field of View (FOV)

Horizontal FOV: 69° ± 3°

This represents the angular coverage of the depth camera in the horizontal direction.

A wider FOV captures more of the scene but may introduce distortion at the edges.

Vertical FOV: 54° ± 2°

The depth camera’s vertical coverage angle.

A higher vertical FOV allows for better full-body tracking and improves depth sensing accuracy at different heights.

These FOV values define the viewing cone of the depth sensor, meaning objects outside these angles will not be captured properly.

(d) Depth Start Point

Front of Lens (Z’): 0.9mm

The depth measurement reference point is 0.9 mm from the front of the lens.

This means the camera starts measuring depth just in front of the lens, making it necessary to add 0.9 mm to raw depth values for ground truth depth measurements.

Back of Module (Z”): 3.0mm

Alternatively, the reference can be measured from the back of the module, which is 3.0 mm from the rear.

In this case, 3.0 mm must be subtracted from the raw depth values to get accurate measurements.

Task 2.17 Determining pose (40 points)

Think about which geometric features could assist you in associating a pose with the object and determine the pose of an identified cube. You are required to submit

1. your assignment of a frame to an object;
2. description of an algorithm for constructing the pose homogeneous transformation (how do you determine the transformation from your identified geometric features?);
3. description of an algorithm for determining the required geometric features (determining the pixel coordinates of relevant pixels of the blob);
4. aptly commented code;
5. images at various steps of a good test run;
6. accuracy of the pose compared to physical measurements over multiple runs.

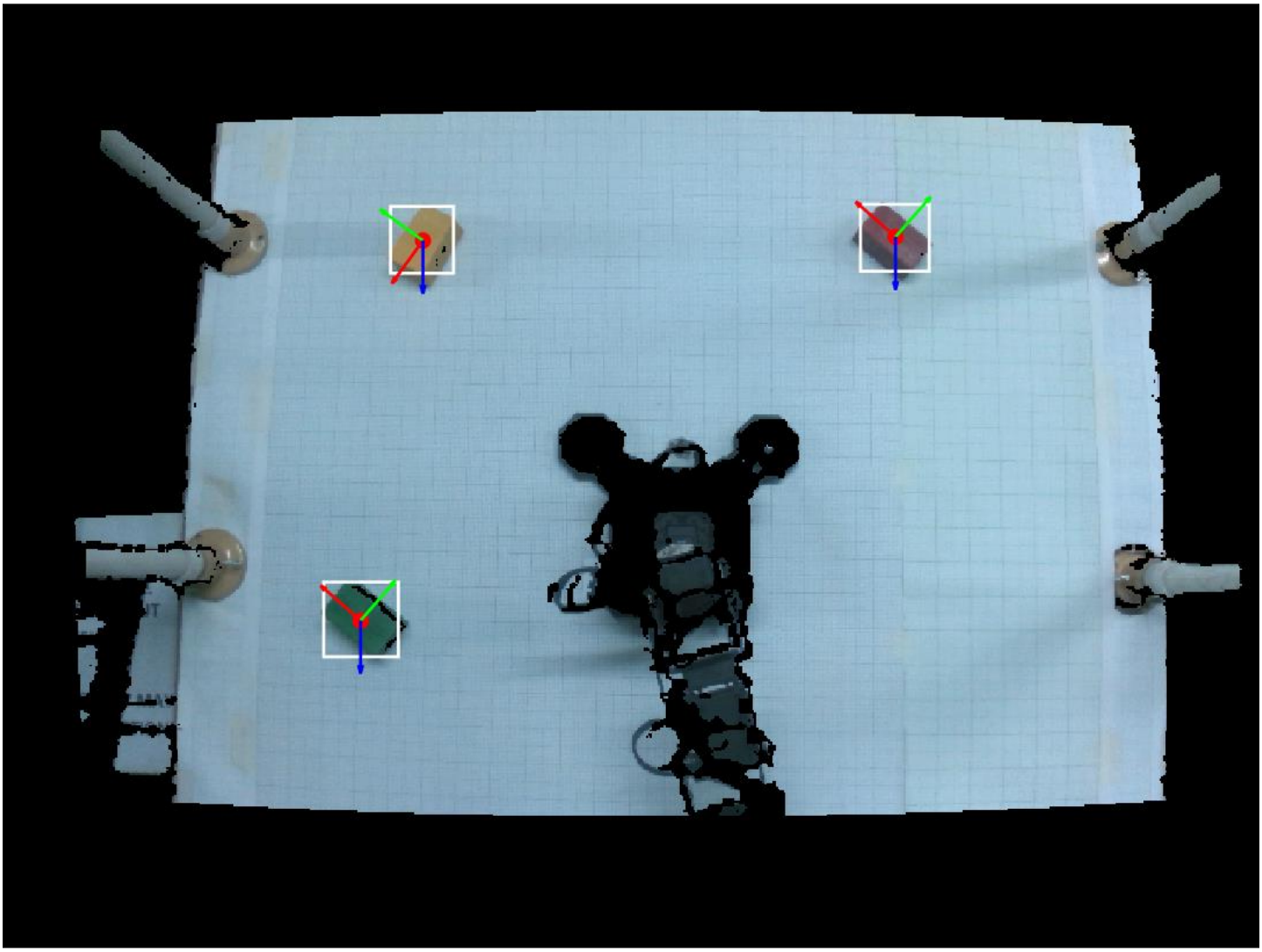
```
%% Create objects
pipe = realsense.pipeline();
colorizer = realsense.colorizer();
cfg = realsense.config();

%% Set configuration for SR305
% Enable depth stream (SR305 uses Z16 format)
streamType = realsense.stream('depth');
formatType = realsense.format('Z16');  %
cfg.enable_stream(streamType, formatType);

% Enable color stream
streamType = realsense.stream('color');
formatType = realsense.format('rgb8');
cfg.enable_stream(streamType, formatType);

% Start streaming
profile = pipe.start(cfg);

%% Acquire device parameters
dev = profile.get_device();
name = dev.get_info(realsense.camera_info.name);
fprintf('Connected Camera: %s\n', name);

% Access Depth Sensor
depth_sensor = dev.first('depth_sensor');
depth_scaling = depth_sensor.get_depth_scale();  % Correct depth scaling

%% Align Color to Depth
for i = 1:5  % Allow camera to settle
    fs = pipe.wait_for_frames();
end
align_to_depth = realsense.align(realsense.stream.depth);
fs = align_to_depth.process(fs);

%% Depth Post-processing (less aggressive for SR305)
depth = fs.get_depth_frame();
width = depth.get_width();
height = depth.get_height();
```

```
spatial = realsense.spatial_filter(0.3, 10, 1, 0); % Less aggressive
depth_p = spatial.process(depth);

temporal = realsense.temporal_filter(0.1, 10, 1);
depth_p = temporal.process(depth_p);

%% Color Frame Processing
color = fs.get_color_frame();

%% Convert frames to MATLAB format
depth_data = depth_scaling * double(depth_p.get_data());
depth_image = permute(reshape(depth_data', [width, height]), [2, 1]);

color_data = color.get_data();
color_image = permute(reshape(color_data', [3, color.get_width(), color.get_height()]), [3, 2, 1]);

figure;
imshow(color_image);
title('colour image')

%% Color Mask Processing
redMask = createRedMask(color_image);
greenMask = createGreenMask(color_image);
blueMask = createBlueMask(color_image);
yellowMask = createYellowMask(color_image);


figure;
imshow(imoverlay(color_image, yellowMask));
title('yellow mask')
figure;
imshow(imoverlay(color_image, greenMask));
title('green mask')
figure;
imshow(imoverlay(color_image, blueMask));
title('blue mask')
figure;
imshow(imoverlay(color_image, redMask));
title('red mask')
```

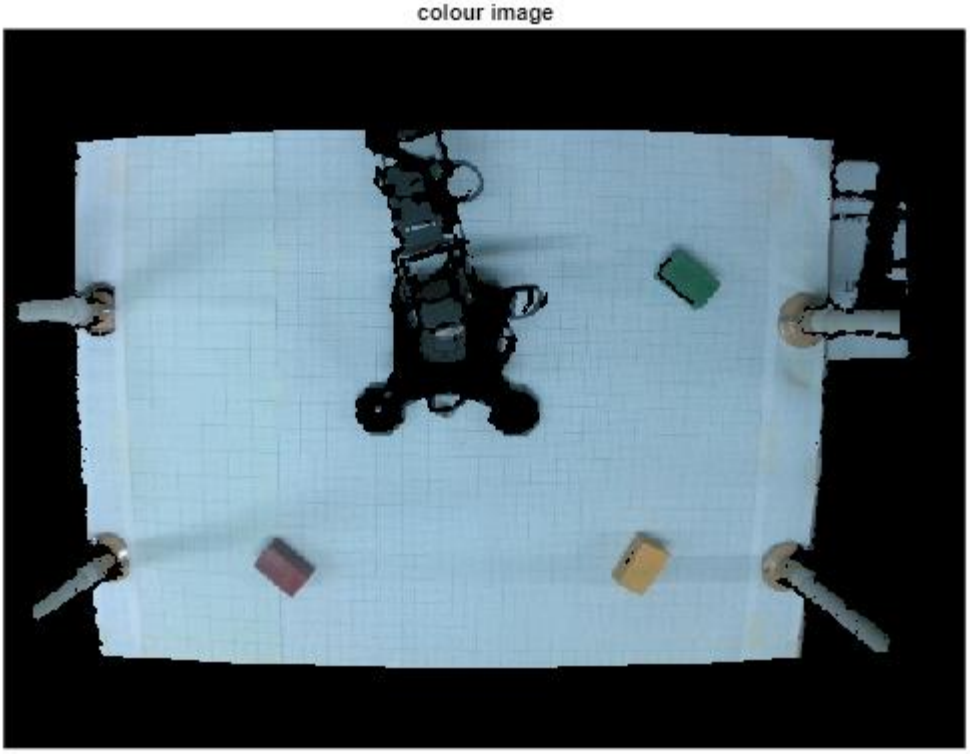
```
%% Get depth sensor intrinsics for coordinate transformation
depth_stream = depth_p.get_profile().as('video_stream_profile');
intrinsics = depth_stream.get_intrinsics();
fx = intrinsics.fx; % Focal length x
fy = intrinsics.fy; % Focal length y
cx = intrinsics.ppx; % Principal point x
cy = intrinsics.ppy; % Principal point y

axis_length = 30;

% Process each color mask to detect objects and compute poses
colors = {'Red', 'Green', 'Blue', 'Yellow'};
masks = {redMask, greenMask, blueMask, yellowMask};

% Initialize figure
figure, imshow(color_image); hold on;
```

```
% Loop over colors
for i = 1:length(masks)
    mask = masks{i};
    stats = regionprops(mask, 'BoundingBox', 'Centroid', 'Orientation');

    for j = 1:length(stats)
        % Get bounding box and centroid
        box = stats(j).BoundingBox;
        centroid = stats(j).Centroid;
        theta = stats(j).Orientation; % Orientation in degrees

        % Reshape depth_data into a 2D matrix of size [height, width]
        depth_matrix = reshape(depth_data, [width, height])';

        % Get depth value at centroid
        u = round(centroid(1)); % X-coordinate (column)
        v = round(centroid(2)); % Y-coordinate (row)
        Z = depth_matrix(v, u); % Corrected indexing

        % Convert 2D centroid to 3D world coordinates
        X = (u - cx) * Z / fx;
        Y = (v - cy) * Z / fy;

        % Convert angle to radians
        theta_rad = deg2rad(theta);

        % Correct X-axis (red) and Y-axis (green) to counteract clockwise orientation
        x_axis = [cos(theta_rad), -sin(theta_rad)];
        y_axis = [sin(theta_rad), cos(theta_rad)];

        % Compute end points for visualization
        x_end = centroid + axis_length * x_axis;
        y_end = centroid + axis_length * y_axis;

        % Plot bounding box
        rectangle('Position', box, 'EdgeColor', 'w', 'LineWidth', 2);

        % Plot centroid
        plot(centroid(1), centroid(2), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

        % Plot X-axis (red)
        quiver(centroid(1), centroid(2), x_axis(1) * axis_length, x_axis(2) * axis_length, ...
               'r', 'LineWidth', 2, 'MaxHeadSize', 0.5);

        % Plot Y-axis (green)
        quiver(centroid(1), centroid(2), y_axis(1) * axis_length, y_axis(2) * axis_length, ...
               'g', 'LineWidth', 2, 'MaxHeadSize', 0.5);

        % Simulated Z-axis (blue) - assuming top-down view
        quiver(centroid(1), centroid(2), 0, -axis_length, 'b', 'LineWidth', 2, 'MaxHeadSize', 0.5);

        % Display real-world pose info
        fprintf('%s Cube: 2D Centroid (%.2f, %.2f) → 3D Coords (X: %.2f, Y: %.2f, Z: %.2f), Orientation %.2f°\n', ...
                colors{i}, centroid(1), centroid(2), X, Y, Z, theta);
    end
end

hold off;
```

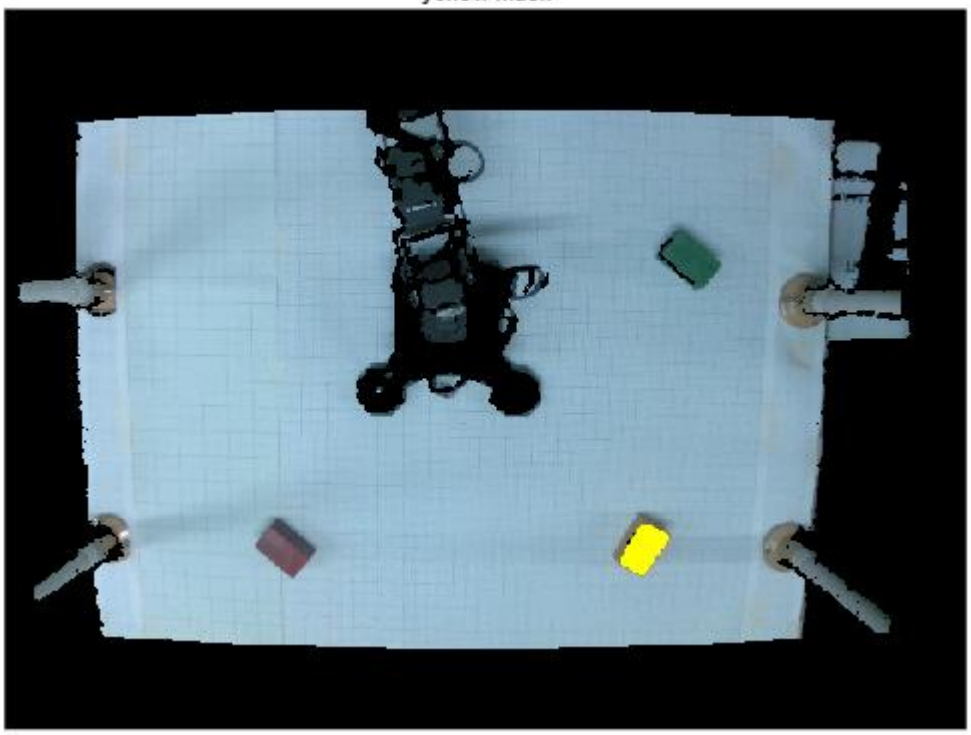
```
pipe.stop();
```



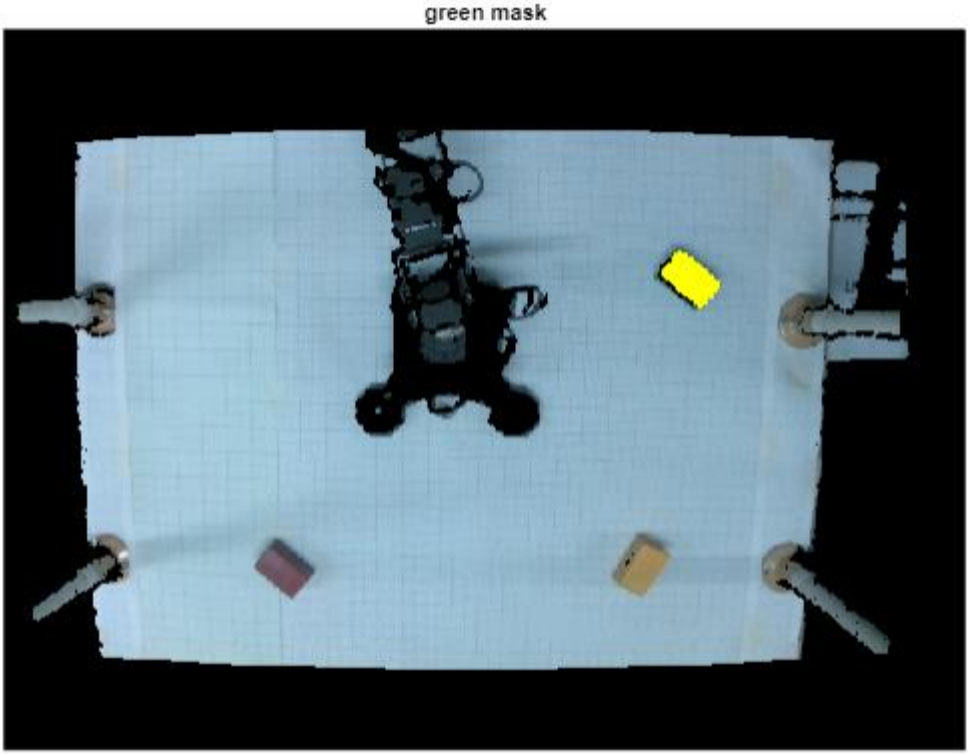
colour image



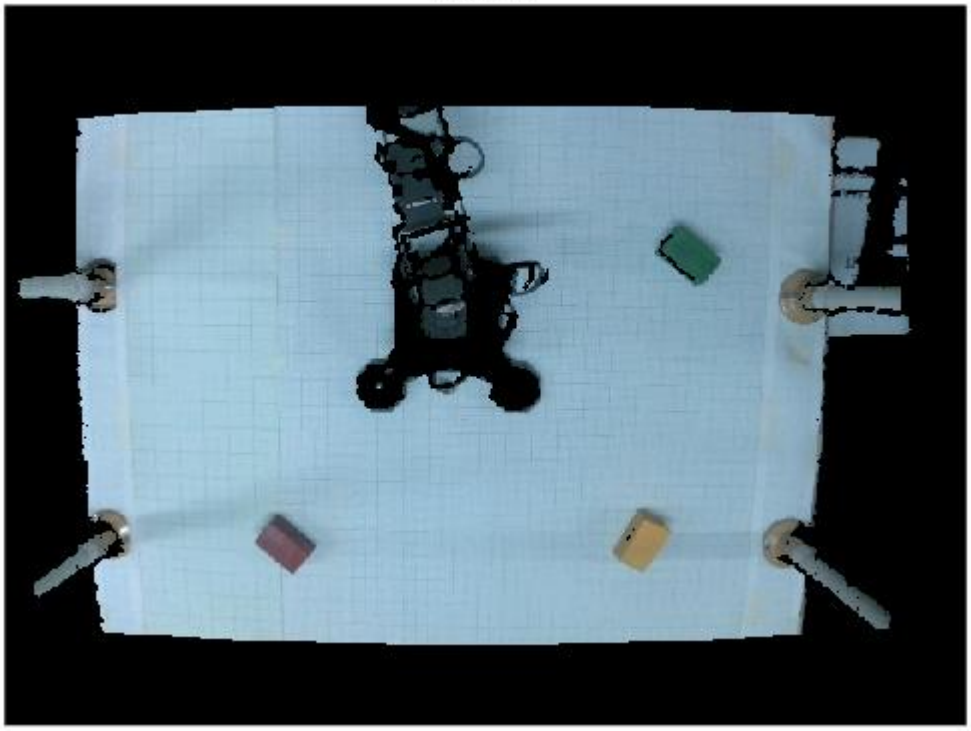
yellow mask



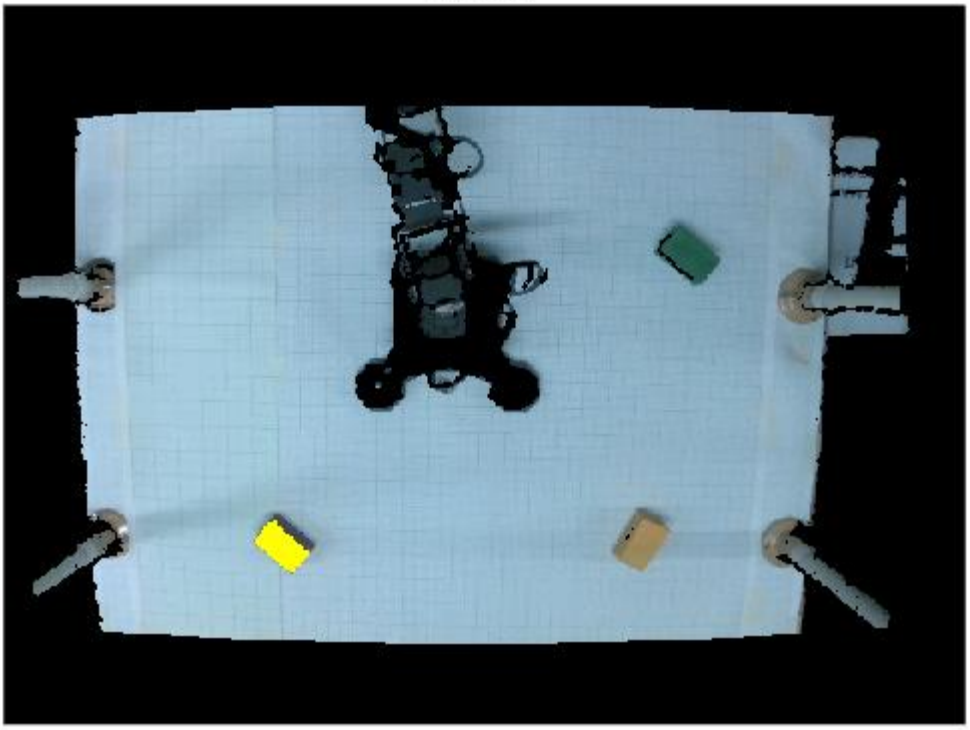
green mask



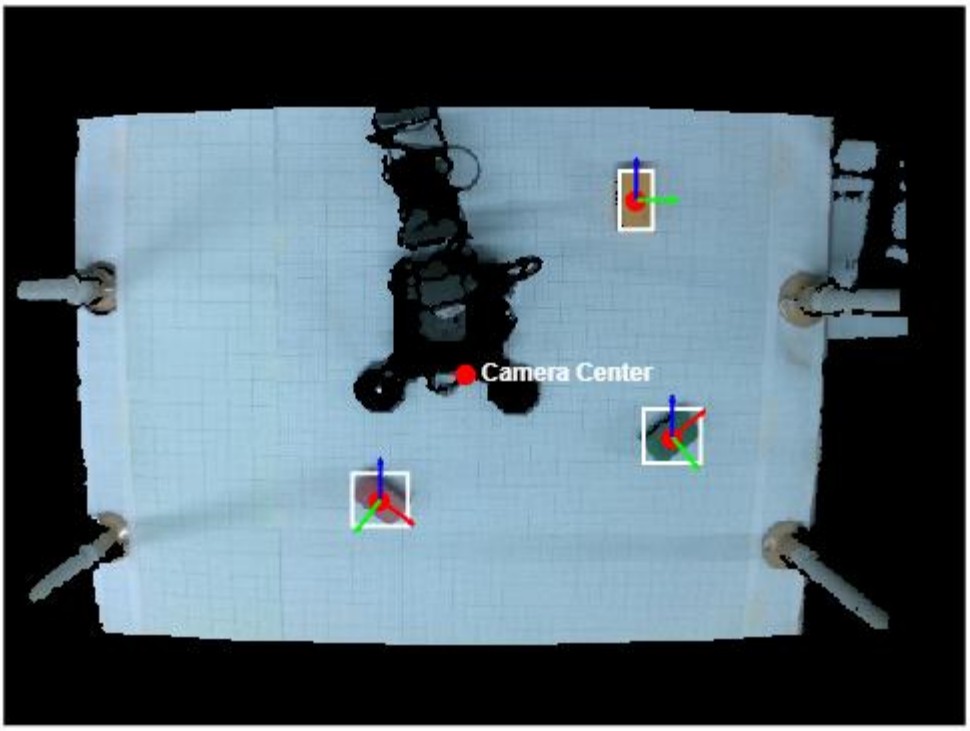
blue mask



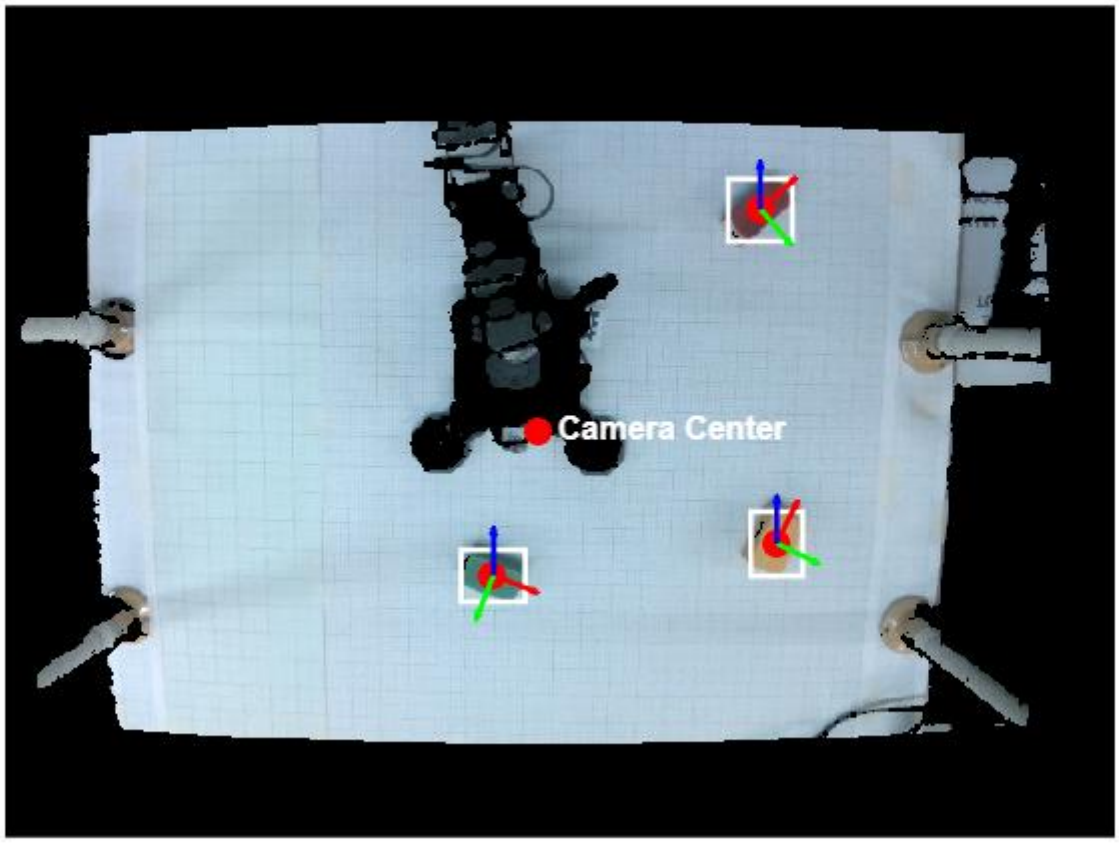
red mask



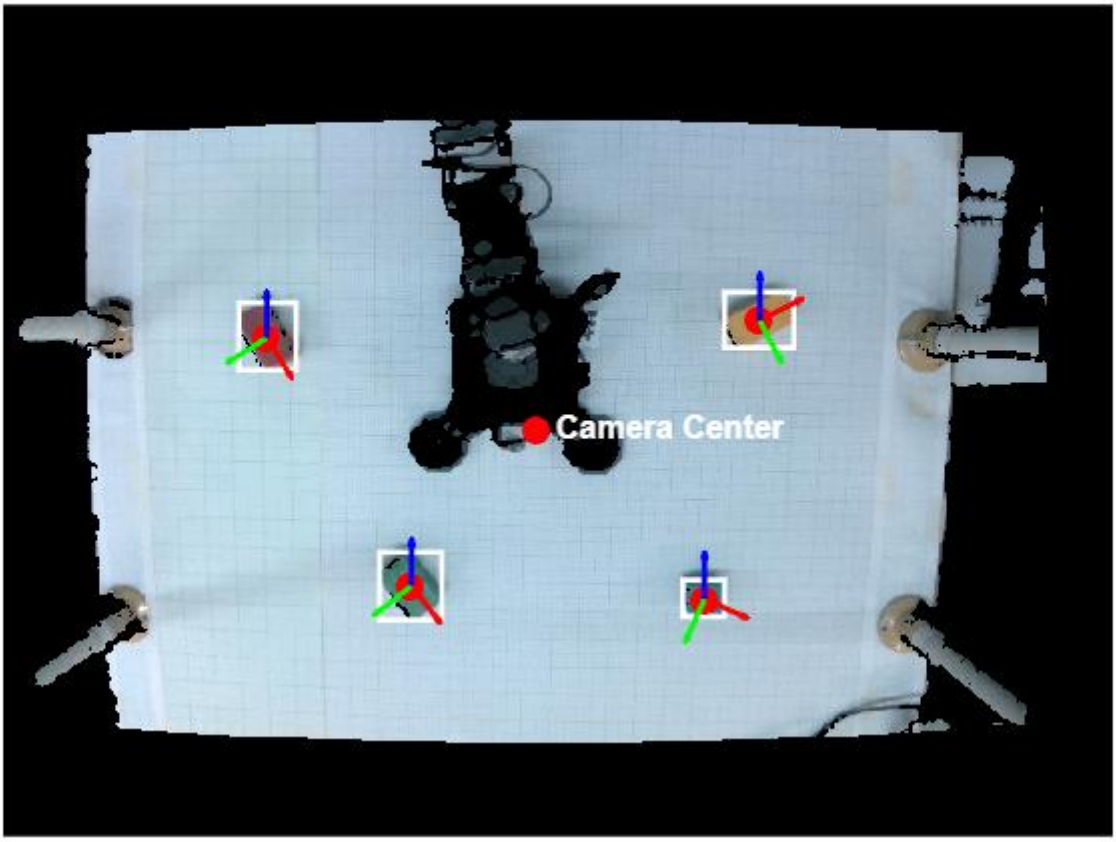
Red Cube: 2D Centroid (251.21, 329.87) → 3D Coords (X: -0.08, Y: 0.12, Z: 0.68), Orientation -37.16°
Green Cube: 2D Centroid (445.50, 288.03) → 3D Coords (X: 0.19, Y: 0.06, Z: 0.67), Orientation 39.06°
Yellow Cube: 2D Centroid (421.49, 129.75) → 3D Coords (X: 0.15, Y: -0.16, Z: 0.65), Orientation 89.12°



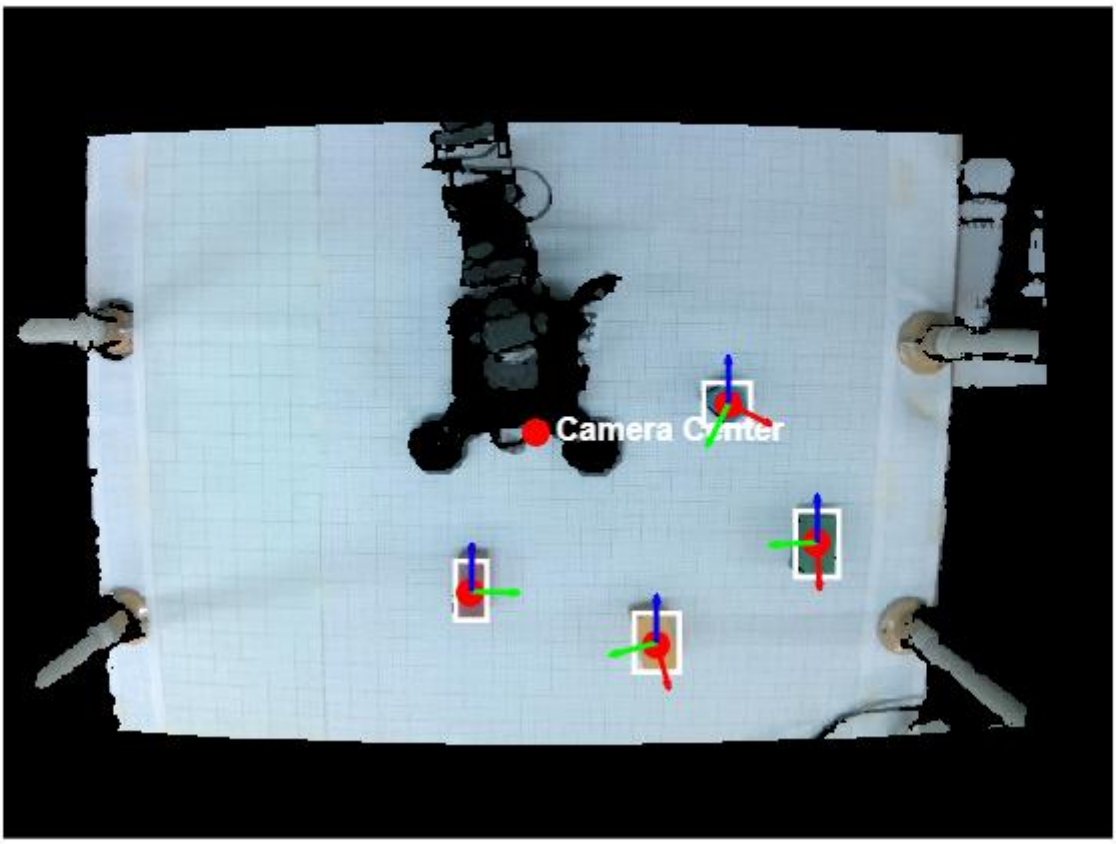
Red Cube: 2D Centroid (436.03, 117.97) → 3D Coords (X: 0.17, Y: -0.18, Z: 0.65), Orientation 41.05°
Green Cube: 2D Centroid (282.14, 329.21) → 3D Coords (X: -0.04, Y: 0.12, Z: 0.68), Orientation -21.56°
Yellow Cube: 2D Centroid (445.54, 310.88) → 3D Coords (X: 0.20, Y: 0.09, Z: 0.68), Orientation 63.88°



Red Cube: 2D Centroid (152.48, 192.86) → 3D Coords (X: -0.22, Y: -0.07, Z: 0.67), Orientation -57.78°
Green Cube: 2D Centroid (236.10, 335.90) → 3D Coords (X: -0.11, Y: 0.13, Z: 0.69), Orientation -52.14°
Blue Cube: 2D Centroid (405.02, 343.45) → 3D Coords (X: 0.14, Y: 0.14, Z: 0.68), Orientation -24.01°
Yellow Cube: 2D Centroid (436.98, 182.68) → 3D Coords (X: 0.18, Y: -0.09, Z: 0.66), Orientation 27.06°



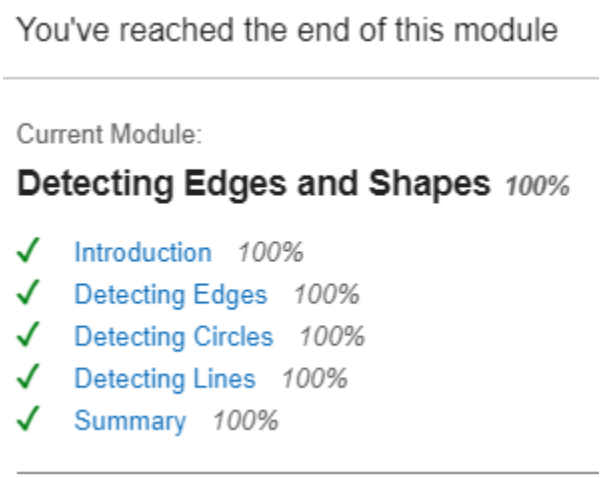
```
Red Cube: 2D Centroid (270.62, 337.99) → 3D Coords (X: -0.06, Y: 0.13, Z: 0.69), Orientation 88.45°
Green Cube: 2D Centroid (470.06, 309.67) → 3D Coords (X: 0.23, Y: 0.09, Z: 0.67), Orientation -86.96°
Blue Cube: 2D Centroid (418.81, 229.48) → 3D Coords (X: 0.15, Y: -0.02, Z: 0.66), Orientation -26.47°
Yellow Cube: 2D Centroid (377.34, 367.59) → 3D Coords (X: 0.10, Y: 0.18, Z: 0.69), Orientation -74.84°
```



The real world coordinates are very accurate. The real world x and y values are given in meters, and by measuring them physically I can confirm that the values are close to correct.

Task 2.15 Detecting Features (10 points)

Complete module 'Detecting Edges and Shapes' of the course 'Image Processing with MATLAB'.



Task 2.18 Perception Pipeline (20 points)

In this task, you'll combine `depth_example` and your previously developed algorithms to develop an automated perception pipeline that acquires an RGB image and a depth image of your scene and output poses of all the cubes of your chosen color in the scene.

I have already combined everything in the previous task.