# Lab 08 Final Report: Vision-Guided Pick and Place with FSM

Authors: Muhammad Aqeel Mehdi & Ehzem

Instructor: Prof. Basit Memon

Lab Assistant: Khuzaima Ali Khan

Task 8.1 - Picking and Placing a Cube (70 points):
The robot must detect a single cube placed randomly in the workspace, compute its coordinates using image processing and depth sensing, and then move to grasp and place it in a predefined target location.

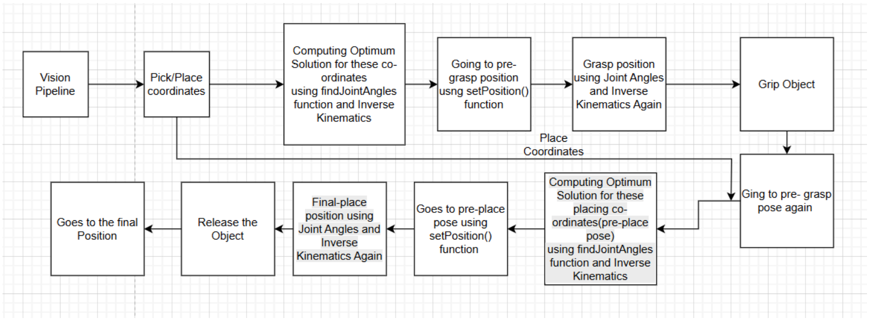
Task 8.2 - Bin It! (30 points):
Ten cubes (in two distinct colors) are randomly scattered. The robot must autonomously detect all cubes of a specific color and move them to the placement zone, earning points for each successfully placed cube.

## Methodology / FSM Description

We implemented a Finite State Machine (FSM) to manage the sequence of operations for pick-and-place. The pipeline includes:

1. Vision Pipeline – RGB image segmentation using LAB thresholding.
2. Perception Module – Converts 2D centroid + depth to 3D camera coordinates.
3. Coordinate Mapping – Uses camera intrinsics and extrinsics to convert to world coordinates.
4. IK & Motion Planning – Computes joint angles using `findJointAngles` and `findOptimalSolution`.
5. Pre-Grasp/Grasp – Robot first moves to a safe pre-grasp pose, then to final grasp pose.
6. Pick & Place – Gripper closes; the object is lifted and moved to the placement location.

The FSM diagram is shown below:


Vision Pipeline
Pick/Place coordinates
Computing Optimum Solution for these co-ordinates using findJointAngles function and Inverse Kinematics
Going to pre-grasp position usng setPosition() function
Grasp position using Joint Angles and Inverse Kinematics Again
Grip Object
Place Coordinates
Ging to pre- grasp pose again
Computing Optimum Solution for these placing co-ordinates(pre-place pose) using findJointAngles function and Inverse Kinematics
Goes to pre-place pose using setPosition() function
Final-place position using Joint Angles and Inverse Kinematics Again
Release the Object
Goes to the final Position

## Step-by-Step Output:

- Detected Centroid: (297.56, 146.76)
- Camera to World Mapping: (x, y, z) = (-2.77, 12.76, 3)
- Pick Joint Angles: [0.2138 -0.9974 -1.7410 -0.4031]
- Pre-Grasp Pose Success: setPosition(...) → Success
- Gripper Command: positionJaw(34) → Confirmed
- Place Location: (-13, -13, 8)
- Place Joint Angles: [-0.7854 1.1683 1.0217 0.9516]
- Gripper Release: Jaw Open

## Limitations and Assumptions

- There was some accuracy problem if we place a cube very closer to the robot like next to the base frame
- Thresholding assumes constant lighting.

- Camera Adjustment should be very precised

## Conclusion

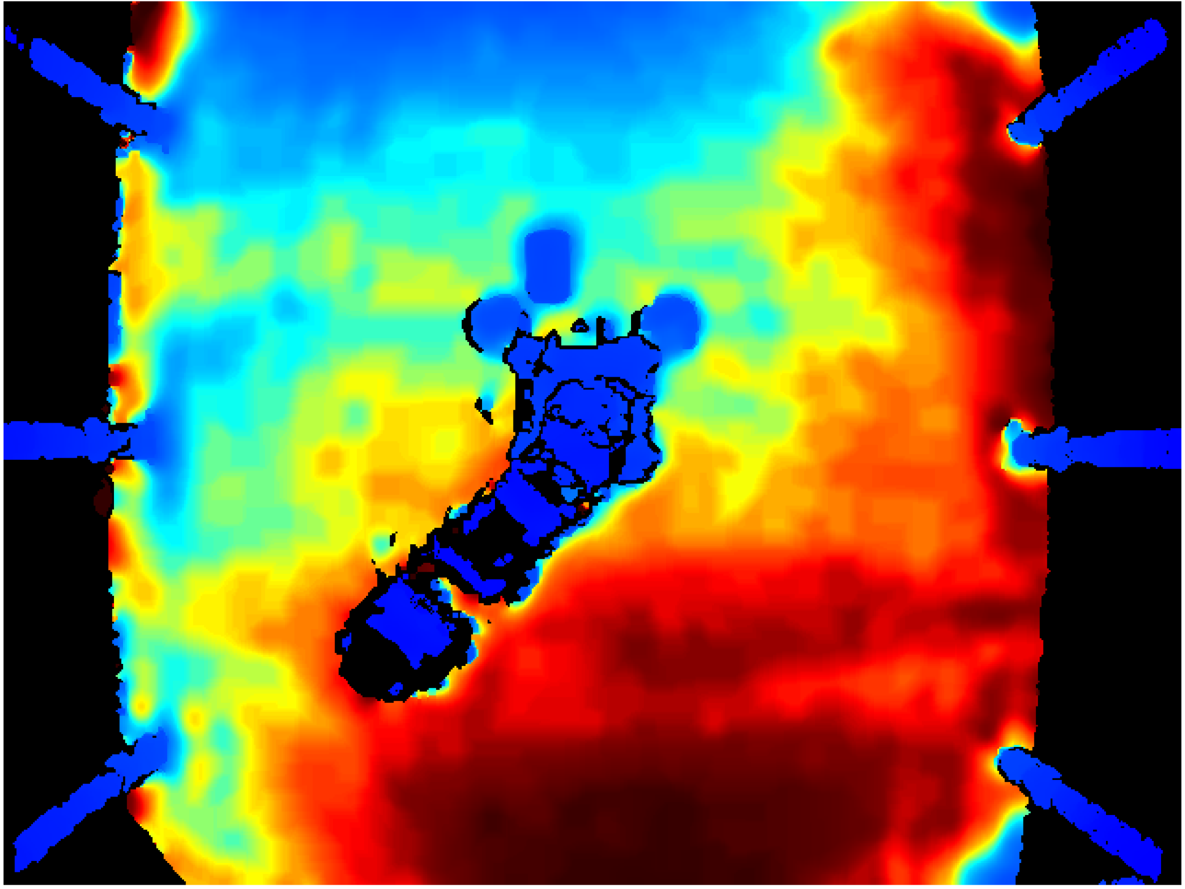
This lab successfully demonstrated a complete pick-and-place pipeline using perception, inverse kinematics, and motion planning. The FSM-based control structure provided clarity and reliability. While hardware and sensing limitations existed, the system achieved its objectives under controlled conditions.

**Code is attached below**

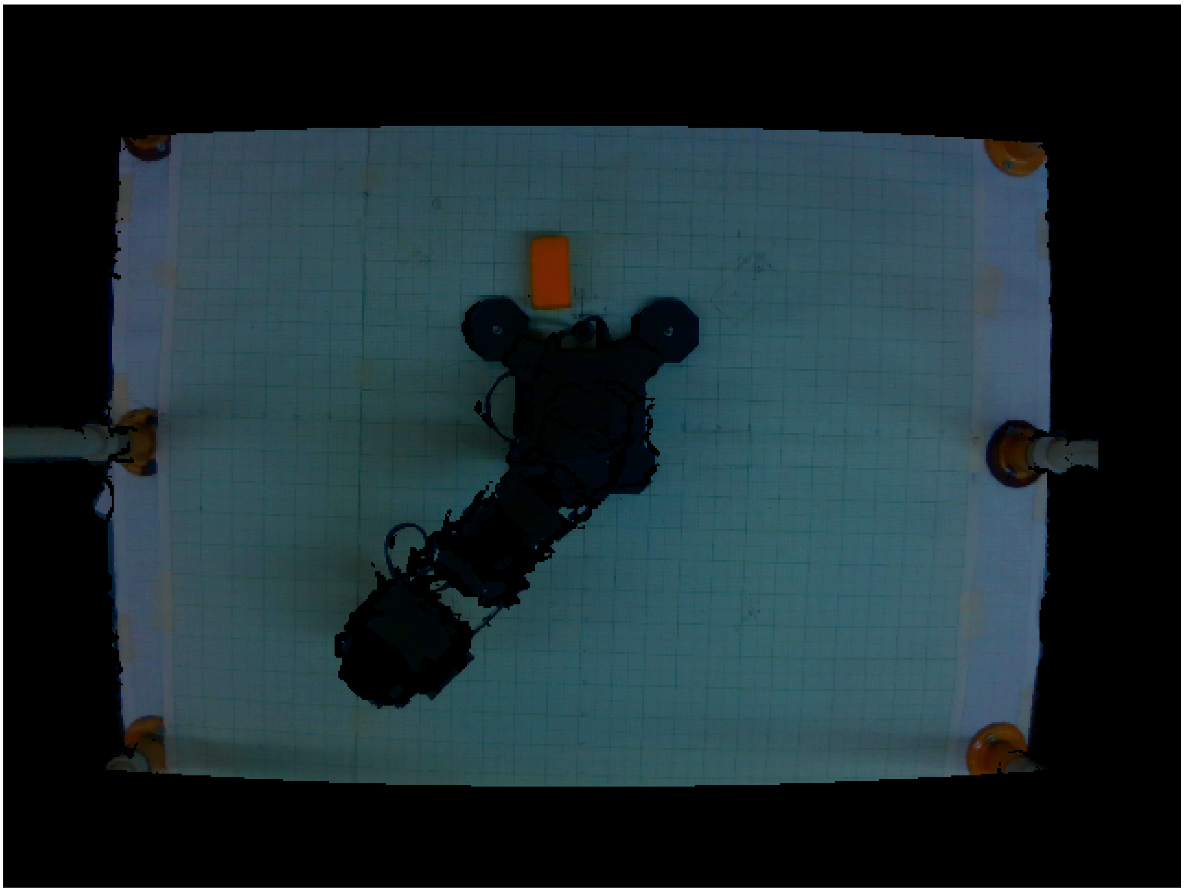
```
% arb = Arbotix('port', 'COM4', 'nservos', 5);
locations = [0 0 0];
positions = preception();
```

```
color_intrinsics = struct with fields:
     width: 1920
    height: 1080
       ppx: 950.7278
       ppy: 533.0599
        fx: 1.4089e+03
        fy: 1.4089e+03
     model: 0
    coeffs: [0 0 0 0 0]
depth_intrinsics = struct with fields:
     width: 640
    height: 480
       ppx: 305.3088
       ppy: 245.3343
        fx: 474.7335
        fy: 474.7335
     model: 2
    coeffs: [0.1101 0.1787 0.0041 0.0021 -0.0799]
Tdc = struct with fields:
       rotation: [1.0000 -0.0033 0.0013 0.0033 1.0000 -0.0025 -0.0013 0.0025 1.0000]
    translation: [0.0257 0.0013 0.0041]
Tdc = struct with fields:
       rotation: [1.0000 -0.0033 0.0013 0.0033 1.0000 -0.0025 -0.0013 0.0025 1.0000]
    translation: [0.0257 0.0013 0.0041]
```

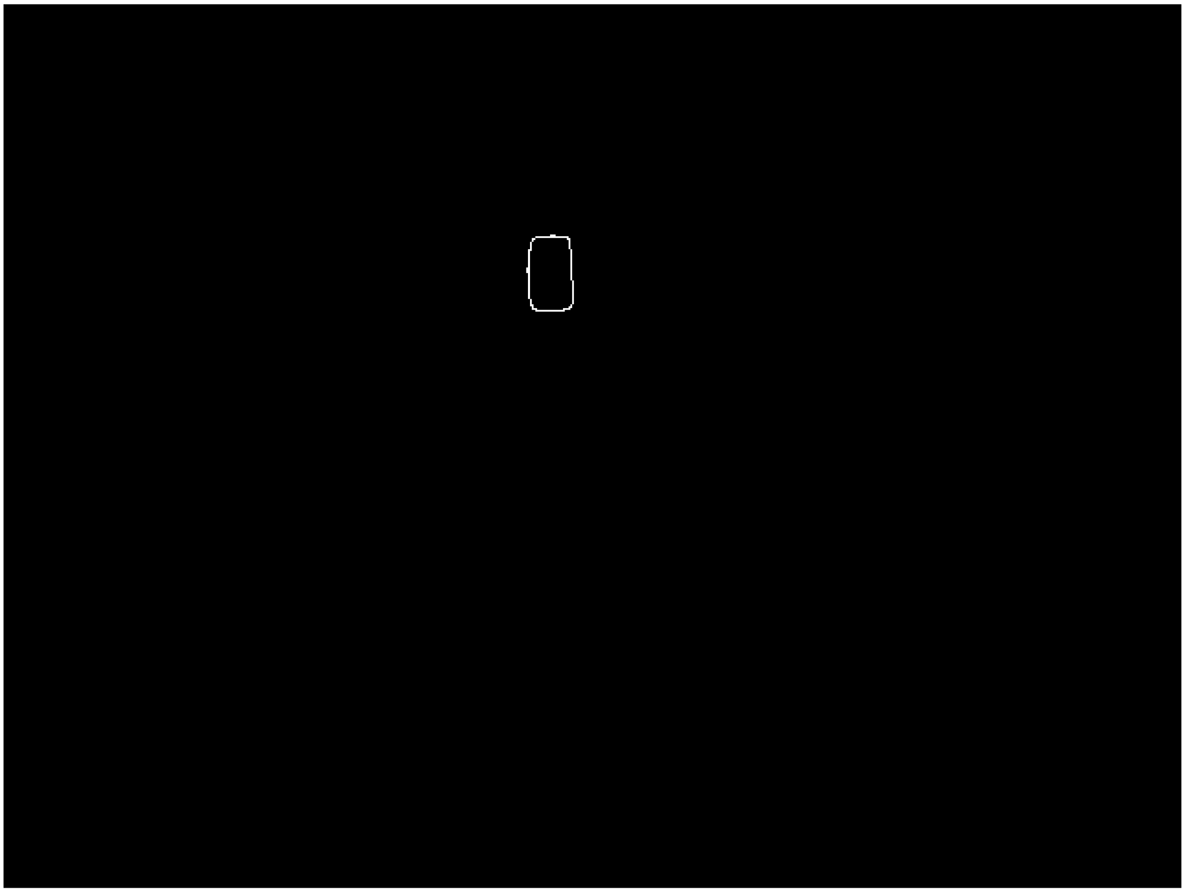
Colorized depth frame from Intel RealSense SR305

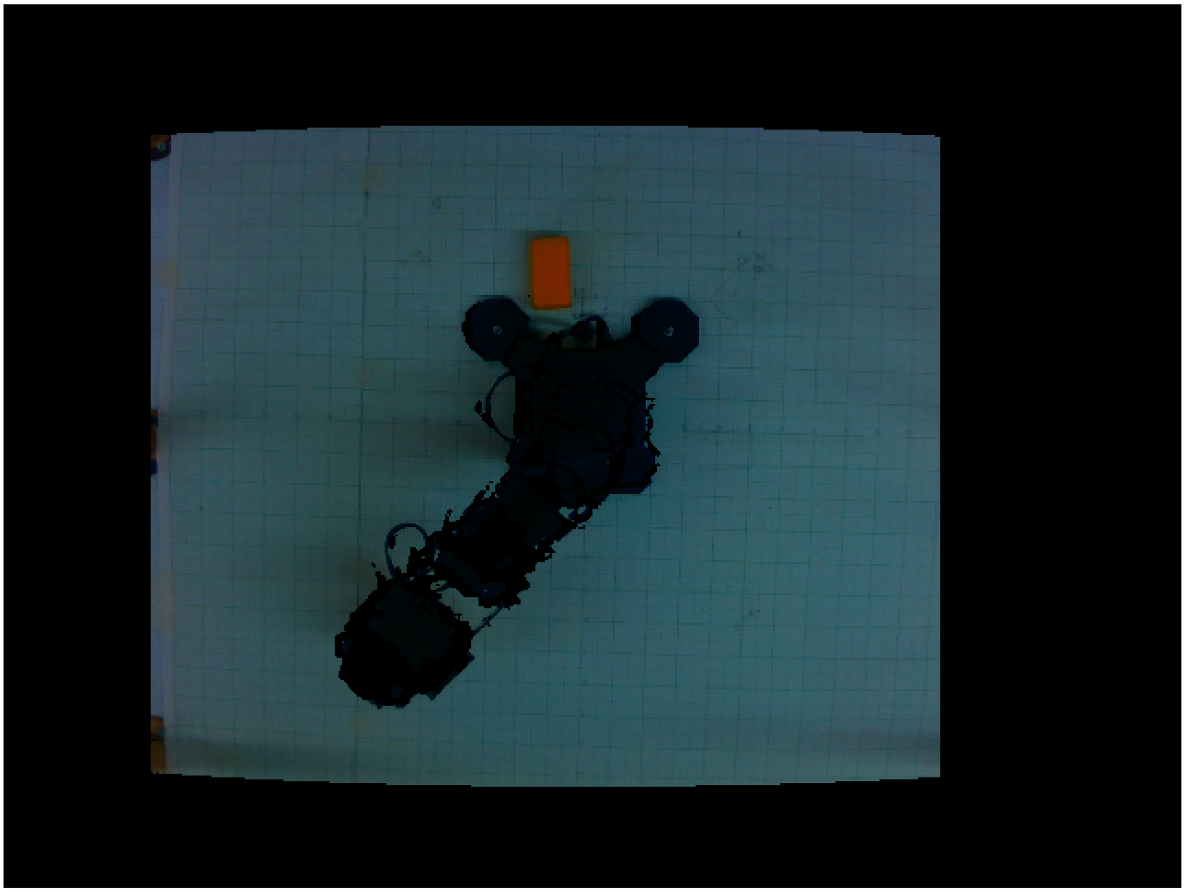


Color RGB frame from Intel RealSense SR305



```
image
```





```
object_center = 1×2
  297.5625  146.7656
         0         0
  297.5625  146.7656
x = 2
```

```
topickup = [locations; positions];
for i = 2:size(topickup, 1)
    x = topickup(i, 1)
    y = topickup(i, 2)
    z = topickup(i, 3)
    lab(x, y, z, arb);
end
```

```
x = -2.7715
y = 12.7657
z = 3
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
```

```
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
Successfully set jaw
ans =


     []
solutions = 4×4
    0.2138   -2.7385    1.7410   -2.1442
    0.2138   -0.9974   -1.7410   -0.4031
   -2.9278    2.7385   -1.7410    2.1442
   -2.9278    0.9974    1.7410    0.4031
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
   -0.0051   -0.0102   -0.0102   -0.0102   -0.0051
pick_jointAngles = 1×4
    0.2138   -0.9974   -1.7410   -0.4031
p_dest = 1×3
   -2.7715   12.7657    8.0000
solutions = 4×4
    0.2138   -2.3990    1.8092   -2.5518
    0.2138   -0.5898   -1.8092   -0.7426
   -2.9278    2.3990   -1.8092    2.5518
   -2.9278    0.5898    1.8092    0.7426
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
   -0.0051   -0.0102   -0.0102   -0.0102   -0.0051
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
errorCode = 0
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
```

```
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
Successfully set jaw
ans = 0.1278
errorGrip = logical
   1
p_dest = 1×3
   -2.7715   12.7657    8.0000
solutions = 4×4
    0.2138   -2.3990    1.8092   -2.5518
    0.2138   -0.5898   -1.8092   -0.7426
   -2.9278    2.3990   -1.8092    2.5518
   -2.9278    0.5898    1.8092    0.7426
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
    0.1994   -0.9664   -1.7181   -0.3988    1.1505
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
phi2 = -1.5708
pre_z = 15
solutions = 4×4
   -3.9270   -1.4913    0.6618   -2.3121
   -3.9270   -0.8295   -0.6618   -1.6503
   -0.7854    1.4913   -0.6618    2.3121
   -0.7854    0.8295    0.6618    1.6503
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
    0.1994   -0.6136   -1.8050   -0.7312    1.1505
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
solutions = 4×4
   -3.9270   -2.1900    1.0217   -1.9733
   -3.9270   -1.1683   -1.0217   -0.9516
   -0.7854    2.1900   -1.0217    1.9733
   -0.7854    1.1683    1.0217    0.9516
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
```

```
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
   -0.7977    0.8744   -0.2863    0.7568    1.1505
place_jointAngles = 1×4
   -0.7854    1.1683    1.0217    0.9516
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
Successfully set jaw
ans =

     []
p_dest = 1×3
   -13   -13    10
solutions = 4×4
   -3.9270   -1.9068    0.9954   -2.2303
   -3.9270   -0.9113   -0.9954   -1.2348
   -0.7854    1.9068   -0.9954    2.2303
   -0.7854    0.9113    0.9954    1.2348
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
currentAngles = 1×5
   -0.7977    1.1965    1.0431    0.9460   -0.0051
Warning: instrfind will be removed in a future release. For serialport, tcpclient, tcpserver, udpport,
visadev, aardvark, and ni845x objects, use serialportfind, tcpclientfind, tcpserverfind, udpportfind,
visadevfind, aardvarkfind, and ni845xfind instead.
serPort COM4is in use.   Closing it.
Warning: serial will be removed in a future release. Use serialport instead.
If you are using serial with icdevice, continue using serial in this MATLAB release.
i = 4
[INFO] Joint angles sent successfully.
```

```matlab
function lab(x, y, z, arb)
    arb = Arbotix('port', 'COM4', 'nservos', 5);
    arb.setpos([0, 0, 0, 0, 0], [50, 50, 50, 50, 50]);
    position = 34;

    success = positionJaw(position);
    % Create an Picking position
    %x = 11.79; y = -17.3; z = 7; phi = -pi/2; % Example target pose
    phi = -pi/2; % Example target pose
    %x = -13; y = 13; z = 3; phi = -pi/2;
    %x = 5; y = 0; z = 30; phi = pi/4; % Example target pose(working fine)
    % Example initial joint angles

    pick_jointAngles = findOptimalSolution( x, y, z, phi)
    pickObject([x, y, z], arb);
    errorCode = setPosition(pick_jointAngles)
    errorGrip = gripObject()
    % %After picking the object we need to go to the pre-grasp pose
    pickObject([x, y, z], arb);
    %now we need to go to the place location
    x2=-13;
    y2=-13;
    z2=5;
    phi2=-pi/2
    %going to pre-place pose
    pre_z=z2+10
    jointAnglesUp = findOptimalSolution(x2, y2, pre_z, phi2);
    errorUp = setPosition(jointAnglesUp);
    place_jointAngles= findOptimalSolution(x2, y2, z2, phi2)
    if isempty(place_jointAngles)
        error('No valid IK solution for pick location.');
    end
    errorPlace = setPosition(place_jointAngles);
    errorRelease = positionJaw(34);
    errorPickObj = pickObject([x2, y2, z2] ,arb);
end



function success = positionJaw(position)
    l1 = 8.68;
    l2 = 25.91;
    d_min = 21;
    d_max = 34;
    if position < d_min || position > d_max
        fprintf('Error: Jaw position outside of range');
        success = false;
        return;
    end

    initial_guess = 0;
    options = optimoptions('fsolve', 'Display', 'off');
    theta = fsolve(@(tg) (l1 * cos(tg) + l2 * cos(tg + (asin(-l1 * sin(tg) / l2) -
tg)) - position), initial_guess, options);

    motorLimit = [deg2rad(0), deg2rad(115)];
    if theta < motorLimit(1) || theta > motorLimit(2)
        fprintf('Error: Gripper exceeds limits [-150°, 150°]');
        success = false;
        return;
    end



    arb = Arbotix('port', 'COM4', 'nservos', 5);


    speed = 70; % consistent with lab settings
    arb.setpos(5, theta, speed);


    % Success
    fprintf('Successfully set jaw');


    arb.getpos(5)
    success = true;
end

function solution = findOptimalSolution(x, y, z, phi)
    % Get all IK solutions
    solutions = findJointAngles(x, y, z, phi)
    if y < 0
    % If y is negative, keep only the 4th row
        solutions = solutions(4, :);
   else
    % If y is 0 or positive, keep only the first 3 rows
        solutions = solutions(1:3, :);
    end

    % Filter out non-realizable solutions based on joint limits
    isValid = cellfun(@checkJointLimits, num2cell(solutions, 2)); % Get logical
array
    validSolutions = solutions(isValid, :); % Keep only valid solutions

    % Check if there are valid solutions
    if isempty(validSolutions)
        error('No valid solutions found.');
    end

    % Get current joint angles
    currentJointAngles = getCurrentPose();

    % Compute the absolute errors for all solutions
    delta = abs(validSolutions - currentJointAngles(1, 1:4));

    % Compute total error for each solution
    totalError = sum(delta, 2);
    % Find the optimal solution with the minimum error
    [~, idx] = min(totalError);
    solution = validSolutions(idx, :);
end

function theta_solutions = findJointAngles(x, y, z, phi)
 % Define link lengths (example values, replace with actual robot parameters)
 a2 = 10.6; % Length of link 2
 a3 = 10.6; % Length of link 3
 d1 = 14.8; % Base height
 a4=7.8;
 % Compute r and s (wrist center coordinates)
 r = sqrt(x^2 + y^2);
 s = z - d1;

 u=(r-a4*(cos(phi)));
 v=(s-a4*(sin(phi)));
 % Compute possible theta1 solutions
 theta1_1 = mod(atan2(y, x)+pi,2*pi)-pi;
 theta1_2 = mod(pi+atan2(y, x)+pi,2*pi)-pi;
 % Compute possible theta3 solutions using the cosine rule
 cos_theta3 = real((u^2 + v^2 - a2^2 - a3^2) / (2 * a2 * a3));

 % if abs(cos_theta3) > 1
 % error('No valid solution for theta3 (cosine rule constraint violated)');
 %end

 theta3_1 = mod((atan2(real(sqrt(1 - cos_theta3.^2)), cos_theta3))+pi,2*pi)-pi; %
First possible theta3

 theta3_2 = (mod((atan2(real(-sqrt(1 - cos_theta3.^2)), cos_theta3))+pi,2*pi)-pi);
% Second possible theta3


 % Compute possible theta2 solutions
 %cosbeta1 = (a3^2 - (u^2+v^2)-(a2^2))/(-2*sqrt(u^2+v^2)*(a2));
% beta1 = acos(cosbeta1);
 theta2_1 = mod(atan2(v, u) - atan2(a3 * sin(theta3_1), a2 + a3 * cos(theta3_1))
+pi,2*pi)-pi;
 theta2_2 = mod(atan2(v, u) - atan2(a3 * sin(theta3_2), a2 + a3 * cos(theta3_2))
+pi,2*pi)-pi;

 %theta2_1 = mod(atan2(v, u) - beta1+pi,2*pi)-pi;
 %theta2_2 = mod(atan2(v, u) + beta1+pi,2*pi)-pi;

 % Compute theta4 for each solution
 theta4_1 = mod((phi - theta2_1 - theta3_1)+pi,2*pi)-pi;
 theta4_2 = mod((phi - theta2_2- theta3_2)+pi,2*pi)-pi;
 % Construct the N × 4 solution matrix
%   if y < 0
%        theta_solutions=[theta1_2-pi/2 -pi/2-(theta2_2-pi) -theta3_2 -theta4_2;];
%   else
%        theta_solutions = [theta1_1-pi/2 -pi/2+theta2_1 theta3_1 theta4_1;
% theta1_1-pi/2 -pi/2+theta2_2 theta3_2 theta4_2;
% theta1_2-pi/2 -pi/2-(theta2_1)-pi -theta3_1 2*pi-theta4_1;];
%  end
theta_solutions = [theta1_1-pi/2 -pi/2+theta2_1 theta3_1 theta4_1;
theta1_1-pi/2 -pi/2+theta2_2 theta3_2 theta4_2;
theta1_2-pi/2 -pi/2-(theta2_1-pi) -theta3_1 -theta4_1;
theta1_2-pi/2 -pi/2-(theta2_2-pi) -theta3_2 -theta4_2;];
% % -----------------
% theta_solutions = [theta1_1+pi/2 -theta2_1+pi/2 -theta3_1 -theta4_1;
%     theta1_1+pi/2 (-theta2_2)+pi/2 -theta3_2 -theta4_2;];
 %theta1_2+pi/2 theta2_1-pi/2 -theta3_1 -theta4_1;
%theta1_2+pi/2 theta2_2-pi/2 -theta3_2 -theta4_2; ];
% ---------
% theta_solutions = [theta1_1-pi/2 theta2_1+pi/2 theta3_1 theta4_1;
%     theta1_1-pi/2 theta2_2-pi/2 theta3_2 theta4_2;
%     theta1_2-pi/2 theta2_1+pi/2 theta3_1 theta4_1;
%     theta1_2-pi/2 theta2_2-pi/2 theta3_2 theta4_2; ]

 % theta_solutions = mod(theta_solutions+pi,2*pi)-pi

end

function isValid = checkJointLimits(theta)
 jointLimits = [-150, 150]; % Joint limits in degrees
 isValid = all(theta >= jointLimits(1) & theta <= jointLimits(2));
end

function currentAngles = getCurrentPose()
 arb = Arbotix('port', 'COM4', 'nservos', 5);
 currentAngles = arb.getpos()
end

function errorCode = setPosition(jointAngles)

     % Constants
     maxAngleDeg = 150; % Maximum servo limit in degrees
     minAngleDeg = -150;

     % Initialize errorCode
     errorCode = 0;
     % Convert radians to degrees for limit check
     anglesDeg = rad2deg(jointAngles);
     % Check limits
     if any(anglesDeg > maxAngleDeg) || any(anglesDeg < minAngleDeg)
```

```
         fprintf('[ERROR] One or more joint angles are outside servo limits
(±150°).\n');
         errorCode = 1;
         return;
     end

     try
     % Send joint angles to servos via Arbotix
         arb = Arbotix('port', 'COM4', 'nservos', 5);
         speed=40;
         arb.setpos(1,jointAngles(1),speed); % Ensure arb object is already
initialized in your workspace
         arb.setpos(2,jointAngles(2),speed);
         arb.setpos(3,jointAngles(3),speed);
         arb.setpos(4,jointAngles(4),speed);
         fprintf('[INFO] Joint angles sent successfully.\n');
     catch
         fprintf('[ERROR] Failed to communicate with Arbotix controller.\n');
         errorCode = 2;
     end
end

function vs = skew(v)
 vs = [0 -v(3) v(2);
 v(3) 0 -v(1);
 -v(2) v(1) 0];
end
function T = exponential(S, theta)
 w_skew = skew(S(1:3));
 exp_w = eye(3) + w_skew*sin(theta) + w_skew^2*(1-cos(theta));
 G = eye(3)*theta + (1-cos(theta))*w_skew+(theta-sin(theta))*w_skew^2;
 T = [exp_w, G*S(4:6);
 zeros(1,3), 1];
end
function Ad_T = adjoint(T)

 R = T(1:3, 1:3);
 p = T(1:3, 4);
 p_hat = skew(p);

 Ad_T = [R, zeros(3,3); p_hat * R, R];
end
function [x, y, z, R] = pincherFK(jointAngles)
 % pincherFK - Computes forward kinematics using POE for Phantom X Pincher
 % INPUT:
 % jointAngles - [theta1, theta2, theta3, theta4] in radians
 % OUTPUT:
 % x, y, z - End-effector position
 % R - Orientation matrix (3x3)
 % Link lengths (in cm)
 l_0 = 10; l_1 = 4.8; l_2 = 10.6; l_3 = 10.6; l_4 = 7.8;
 % Zero configuration (home position)
 R_home = eye(3); % Identity rotation
 p_home = [0; 0; l_0 + l_1 + l_2 + l_3 + l_4];
 M = [R_home, p_home; 0 0 0 1];
 % Screw Axes
 % S1 - Rotation about z-axis at base
 omega1 = [0; 0; 1];
 q1 = [0; 0; l_0];
 v1 = -cross(omega1, q1);
 S1 = [omega1; v1];
 % S2 - Rotation about x-axis at shoulder
 omega2 = [1; 0; 0];
 q2 = [0; 0; l_0 + l_1];
 v2 = -cross(omega2, q2);
 S2 = [omega2; v2];
 % S3 - Rotation about x-axis at elbow
 q3 = [0; 0; l_0 + l_1 + l_2];
 v3 = -cross(omega2, q3);
 S3 = [omega2; v3];
 % S4 - Rotation about x-axis at wrist
 q4 = [0; 0; l_0 + l_1 + l_2 + l_3];
 v4 = -cross(omega2, q4);
 S4 = [omega2; v4];
 % Compute transformation using POE formula
 T = exponential(S1, jointAngles(1)) * ...
 exponential(S2, jointAngles(2)) * ...
 exponential(S3, jointAngles(3)) * ...
 exponential(S4, jointAngles(4)) * M;
 % Extract position and orientation
 position = T(1:3, 4);
 R = T(1:3, 1:3);
 % Output as separate variables
 x = double(position(1));
 y = double(position(2));
 z = double(position(3));
 R=double(R)
 x = round(x, 4);
 y = round(y, 4);
 z = round(z, 4);
 R = round(R, 4);
end


function errorCode = pickObject(pose_obj, arb)
   p_dest = pose_obj + [0 0 5] %pre grasp pose 5cm above final pose
   IK = findOptimalSolution(p_dest(1), p_dest(2), p_dest(3), -pi/2); % Manually
getting destination joint angles for now
   jointAngles = IK(1:4);
   errorCode = setPosition(jointAngles);
   if errorCode ~= 0
       disp("Error.");
       return;
   end
end



function success = gripObject()
    multiplier = 96;
    angleIncrement = 0.29;
    success = positionJaw(multiplier*angleIncrement);
end


function positions = preception()
    % Camera intrinciscs
    [color_intrinsics, depth_intrinsics] = determineIntrinsics()
    Tdc = determineExtrinsics()
    focalLength = [color_intrinsics.fx color_intrinsics.fy];
    principalPoint = [color_intrinsics.ppx color_intrinsics.ppy];
    imagesize = [double(color_intrinsics.width) double(color_intrinsics.height)];
    [ig, colour_img] = depth_example();

    camerawidth = imagesize(1);
    cameralength = imagesize(2);

    imagewidth = depth_intrinsics.width;
    imagelength = depth_intrinsics.height;



    % Read Image
    imwrite(colour_img, 'pic_Color.png');
    img = imread("pic_Color.png");
    % img2 = img;
    % imshow(img)

    rect = [0 0 640 480];
    img(:, 1:80, :) = 0;
    img(:, 510:640, :) = 0;
    % img(425:480, :, :) = 0;
    % img(1:65, :, :) = 0;
    disp('image')
    imshow(img);
    % Convert to LAB color space
    lab_img = rgb2lab(img);
    [L, a, b] = imsplit(lab_img);
    % Thresholding to isolate object
    Mask = b > 7 | a > 7 ;
    rm2 = bwareaopen(Mask, 300);
    rm3 = imfill(rm2, 'holes');
    imshow(rm3)
    % Enhance Edge Detection
    BW = edge(rm3, 'Canny'); % Using Canny edge detector

    imshow(BW)

    stats2 = regionprops(BW, "BoundingBox");
    figure;
    imshow(img), hold on;


    place_z_stack = 3.2;

    centers = [0 0];

    for k = 1:length(stats2)
        % Get the BoundingBox
        segmented_img = zeros(size(BW), 'uint8');
        cube_mask = false(size(BW)); % Create an empty mask
        cube_mask(round(stats2(k).BoundingBox(2)):round(stats2(k).BoundingBox(2) +
stats2(k).BoundingBox(4)), ...
        round(stats2(k).BoundingBox(1)):round(stats2(k).BoundingBox(1) +
stats2(k).BoundingBox(3))) = true;
        cubes = cube_mask & BW;
        [H, theta, rho] = hough(cubes);
        P = houghpeaks(H, 5, 'NHoodSize', [55 11]); % Ensure NHoodSize is odd
        lines = houghlines(cubes, theta, rho, P, 'FillGap', 5, 'MinLength', 10);
        max_len = 0;
        for j = 1:length(lines)
            xy = [lines(j).point1; lines(j).point2];
            len = norm(lines(j).point1 - lines(j).point2);
            if len > max_len
                max_len = len;
                xy_long = xy;
            end
        end
        stats = regionprops(cubes, 'Centroid');
        object_center = stats.Centroid % [x, y] format
        centers = [centers; object_center];

        row = round(object_center(2));
        col = round(object_center(1));
        depth = ig(row, col);
    end

    disp(centers);

    positions = [];
    depth = 0.667;
    % fx = focalLength(1);
    % fy = focalLength(2);
    cx = principalPoint(1);
    cy = principalPoint(2);

    x = length(centers)

    for i = 2:length(centers)
        u = centers(i, 1);
        v = centers(i, 2);
        Z_m = depth;

        [X_cm, Y_cm, Z_cm] = pixelToCameraCoords(u, v, Z_m, cx, cy, camerawidth,
cameralength, imagewidth, imagelength);

        pos = [X_cm, Y_cm, Z_cm 1]';

        % some adjustments
        % X_cm = X_cm*20;
        % Y_cm = Y_cm*20;
        Z_cm = 3;
        positions = [positions; X_cm Y_cm Z_cm];
    end

end




function [X_cm, Y_cm, Z_cm] = pixelToCameraCoords(u, v, Z_m, cx, cy, camerawidth,
cameralength, imagewidth, imagelength)
    % pixelToCameraCoords - Converts 2D pixel coordinates and depth to 3D camera
coordinates
    %
    % Inputs:
    %   u, v   - Pixel coordinates
    %   Z_m    - Depth value in meters
    %   fx, fy - Focal lengths in pixels
    %   cx, cy - Principal point (usually image center)
    %
    % Outputs:
    %   X_cm, Y_cm, Z_cm - 3D coordinates in centimeters

    % Compute X, Y in meters


    u0 = (cx) * (640/1920);
    v0 = (cy) * (480/1080);

    fx = 320/(tan(deg2rad(34.5)));
    fy = 240/(tan(deg2rad(27)));

    X = (((u - u0) * Z_m) / fx); %340 %475.1561%360.6;
    Y = (((v - v0) * Z_m) / fy); %250 %475.1562%600 %471.03;

    % Convert to centimeters
    X_cm = X * 100;
    Y_cm = -Y * 100;
    Z_cm = Z_m * 100;
end

function [color_intrinsics, depth_intrinsics] = determineIntrinsics()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();
    % Start streaming on an arbitrary camera with default settings
    profile = pipe.start();
    % Extract the color stream
    color_stream =
profile.get_stream(realsense.stream.color).as('video_stream_profile');
    depth_stream =
profile.get_stream(realsense.stream.depth).as('video_stream_profile');
    % Get and display the intrinsics
    color_intrinsics = color_stream.get_intrinsics();
    depth_intrinsics = depth_stream.get_intrinsics();
end

function [ig, colour_img] = depth_example()
    %% Create all objects to be used in this file
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();
    % Make Colorizer object to prettify depth output
    colorizer = realsense.colorizer();
    % Create a config object to specify configuration of pipeline
    cfg = realsense.config();


    %% Set configuration and start streaming with configuration
    % Stream options are in stream.m; These options tap into the various
    % sensors included in the camera
    streamType = realsense.stream('depth');
    % Data format options are in format.m
    formatType = realsense.format('Distance');
    % Enable default depth
    cfg.enable_stream(streamType,formatType);
    % Enable color stream
    streamType = realsense.stream('color');
    formatType = realsense.format('rgb8');
    cfg.enable_stream(streamType,formatType);

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

    %% Acquire and Set device parameters
    % Get streaming device's name
    dev = profile.get_device();
    name = dev.get_info(realsense.camera_info.name);

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

    % Access RGB Sensor
    rgb_sensor = dev.first('roi_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();

    % Set the control parameters for the depth sensor
    % See the option.m file for different settable options that are visible
    % to you in the viewer.
    optionType = realsense.option('visual_preset');
    % Set parameters to the midrange preset. See for options:
    % https://intelrealsense.github.io/librealsense/doxygen/
rs__option_8h.html#a07402b9eb861d1defe57dbab8befa3ad
    depth_sensor.set_option(optionType,9);

    % Set autoexposure for RGB sensor
    optionType = realsense.option('enable_auto_exposure');
    rgb_sensor.set_option(optionType,1);
    optionType = realsense.option('enable_auto_white_balance');
    rgb_sensor.set_option(optionType,1);

    %% Align the color frame to the depth frame 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
    align_to_depth = realsense.align(realsense.stream.depth);
    fs = align_to_depth.process(fs);

    % Stop streaming
    pipe.stop();

    %% Depth Post-processing
    % Select depth frame
    depth = fs.get_depth_frame();
    width = depth.get_width();
    height = depth.get_height();

    % Decimation filter of magnitude 2
%     dec = realsense.decimation_filter(2);
%     depth = dec.process(depth);

    % Spatial Filtering
    % spatial_filter(smooth_alpha, smooth_delta, magnitude, hole_fill)
    spatial = realsense.spatial_filter(.5,20,2,0);
    depth_p = spatial.process(depth);

    % Temporal Filtering
    % temporal_filter(smooth_alpha, smooth_delta, persistence_control)
    temporal = realsense.temporal_filter(.13,20,3);
    depth_p = temporal.process(depth_p);

    %% Color Post-processing
    % Select color frame
    color = fs.get_color_frame();

    %% Colorize and display depth frame
    % Colorize depth frame
    depth_color = colorizer.colorize(depth_p);

    % Get actual data and convert into a format imshow can use
    % (Color data arrives as [R, G, B, R, G, B, ...] vector)fs
    data = depth_color.get_data();
    img = permute(reshape(data',
[3,depth_color.get_width(),depth_color.get_height()]),[3 2 1]);

    % Display image
    imshow(img);
    title(sprintf("Colorized depth frame from %s", name));

    %% Display RGB frame
    % Get actual data and convert into a format imshow can use
    % (Color data arrives as [R, G, B, R, G, B, ...] vector)fs
    data2 = color.get_data();
    im = permute(reshape(data2',[3,color.get_width(),color.get_height()]),[3 2 1]);

    % Display image
    figure;
    imshow(im);
    title(sprintf("Color RGB frame from %s", name));

    %% Depth frame without colorizing
    % Convert depth values to meters
    data3 = depth_scaling * double(depth_p.get_data());

    %Arrange data in the right image format
    ig = permute(reshape(data3',[width,height]),[2 1]);

    % Scale depth values to [0 1] for display
    figure;
    imshow(mat2gray(ig));
    colour_img = im;
end

function Tdc = determineExtrinsics()
    % Make Pipeline object to manage streaming
    pipe = realsense.pipeline();
    % Start streaming on an arbitrary camera with default settings
    profile = pipe.start();
    % Extract the color and depth streams
    color_stream =
profile.get_stream(realsense.stream.color).as('video_stream_profile');
    depth_stream =
profile.get_stream(realsense.stream.depth).as('video_stream_profile');
    % Get and display the intrinsics
    Tdc = depth_stream.get_extrinsics_to(color_stream)
end
```