Introduction to Robotics

Lab 8

Shaaf Farooque, Mysha Zulfiqar

```
clc; clear all; close all;
arb = Arbotix('port', 'COM4', 'nservos', 5)
```

```
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

arb =
Arbotix chain on serPort COM4 (open)
 5 servos in chain
```

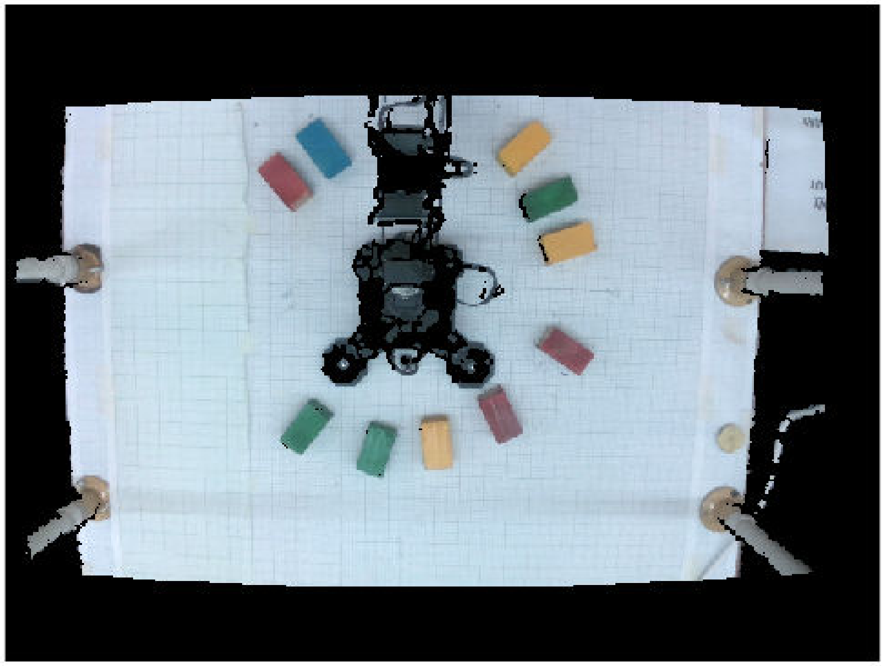
```
determineIntrinsics()
```

```
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]
depth_intrinsics = struct with fields:
     width: 640
    height: 480
       ppx: 308.9356
       ppy: 245.4494
        fx: 474.8976
        fy: 474.8976
     model: 2
    coeffs: [0.1457 0.0512 0.0038 0.0014 0.0663]
```

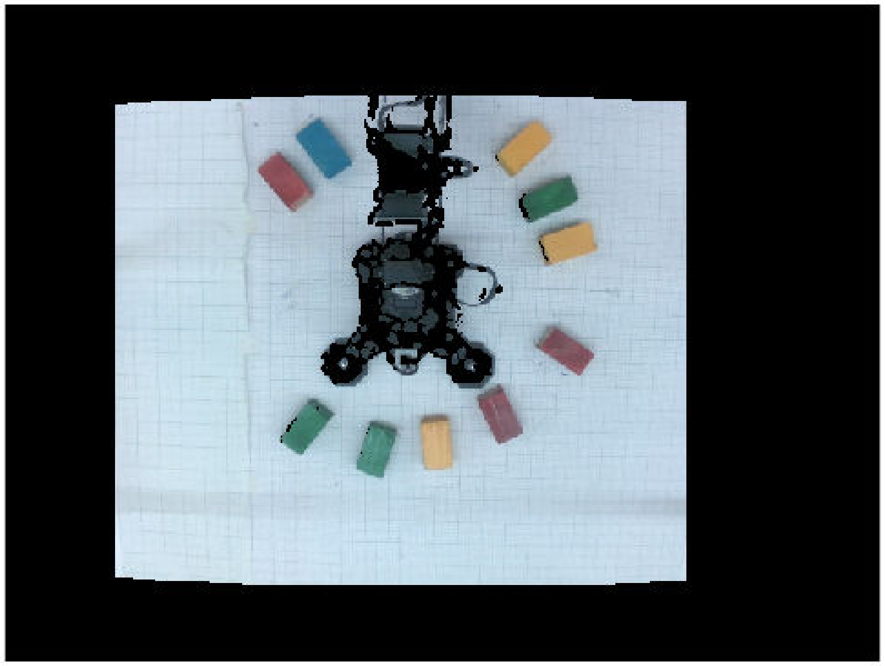
```
determineExtrinsics()
```

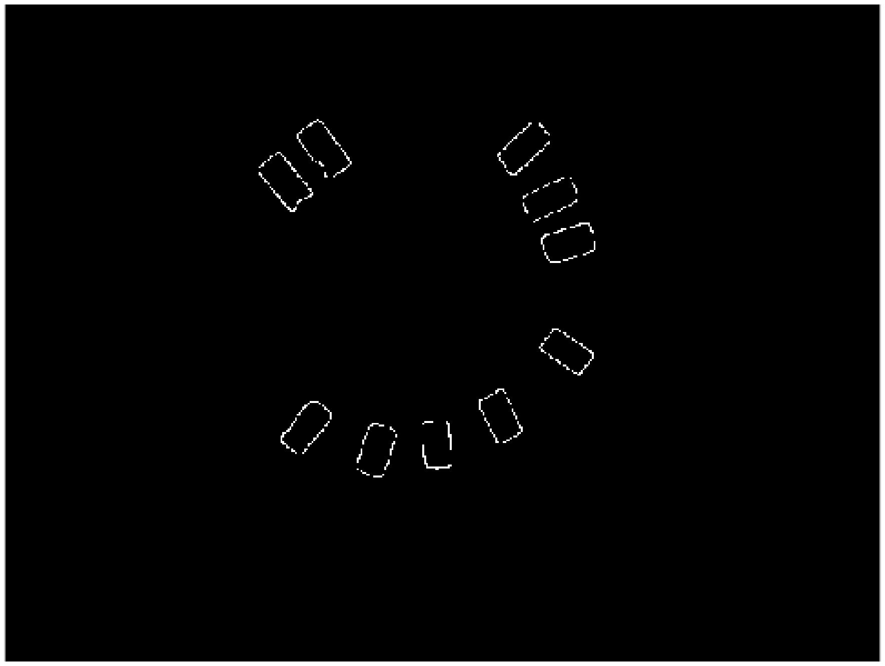
```
Tdc = struct with fields:
       rotation: [1.0000 -7.3491e-04 0.0013 7.4430e-04 1.0000 -0.0074 -0.0013 0.0074 1.0000]
    translation: [0.0257 -7.3326e-04 0.0044]
```

```
[img,ig] = depth_example();
imshow(img)
```



```
% Main robot loop
armDemo(arb);
```





```
x = 10.2568
y = -12.3575
z = 5.4069
p_dest = 1×3
   10.2568  -12.3575    9.4069
angles = 4×4
    0.6928    1.9628   -1.3346    2.5133
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.6928    1.9628   -1.3346    2.5133
    0.6928    0.6431    1.3346    1.1639
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.6928    1.9628   -1.3346    2.5133
    0.6928    0.6431    1.3346    1.1639
    3.8344   -1.9628   -1.3346   -2.5133
         0         0         0         0
angles = 4×4
    0.6928    1.9628   -1.3346    2.5133
    0.6928    0.6431    1.3346    1.1639
    3.8344   -1.9628   -1.3346   -2.5133
    3.8344   -0.6431   -1.3346   -1.1639
solution = 1×4
    0.6928    0.6431    1.3346    1.1639
jointAngles = 1×4
    0.9828    0.6431    1.3346    1.1639
angles = 4×4
    0.6928    2.4654   -1.3643    2.0405
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.6928    2.4654   -1.3643    2.0405
    0.6928    1.1164    1.3643    0.6609
```

```
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.6928    2.4654   -1.3643    2.0405
    0.6928    1.1164    1.3643    0.6609
    3.8344   -2.4654   -1.3643   -2.0405
         0         0         0         0
angles = 4×4
    0.6928    2.4654   -1.3643    2.0405
    0.6928    1.1164    1.3643    0.6609
    3.8344   -2.4654   -1.3643   -2.0405
    3.8344   -1.1164   -1.3643   -0.6609
solution = 1×4
    0.6928    1.1164    1.3643    0.6609
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.7042    2.3152   -1.4123    2.2386
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7042    2.3152   -1.4123    2.2386
    1.7042    0.9191    1.4123    0.8103
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7042    2.3152   -1.4123    2.2386
    1.7042    0.9191    1.4123    0.8103
   -1.4374   -2.3152   -1.4123   -2.2386
         0         0         0         0
angles = 4×4
    1.7042    2.3152   -1.4123    2.2386
    1.7042    0.9191    1.4123    0.8103
   -1.4374   -2.3152   -1.4123   -2.2386
   -1.4374   -0.9191   -1.4123   -0.8103
solution = 1×4
    1.7042    0.9191    1.4123    0.8103
x = 8.4197
y = 12.3929
z = 3.7321
p_dest = 1×3
    8.4197   12.3929    7.7321
angles = 4×4
```

```
    2.5448    2.1507   -1.5388    2.5297
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    2.5448    2.1507   -1.5388    2.5297
    2.5448    0.6302    1.5388    0.9727
         0         0         0         0
         0         0         0         0
angles = 4×4
    2.5448    2.1507   -1.5388    2.5297
    2.5448    0.6302    1.5388    0.9727
   -0.5968   -2.1507   -1.5388   -2.5297
         0         0         0         0
angles = 4×4
    2.5448    2.1507   -1.5388    2.5297
    2.5448    0.6302    1.5388    0.9727
   -0.5968   -2.1507   -1.5388   -2.5297
   -0.5968   -0.6302   -1.5388   -0.9727
solution = 1×4
    2.5448    0.6302    1.5388    0.9727
jointAngles = 1×4
    2.4948    0.6302    1.5388    0.9727
angles = 4×4
    2.5448    2.6246   -1.4482    1.9651
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    2.5448    2.6246   -1.4482    1.9651
    2.5448    1.1932    1.4482    0.5003
         0         0         0         0
         0         0         0         0
angles = 4×4
    2.5448    2.6246   -1.4482    1.9651
    2.5448    1.1932    1.4482    0.5003
   -0.5968   -2.6246   -1.4482   -1.9651
         0         0         0         0
angles = 4×4
    2.5448    2.6246   -1.4482    1.9651
    2.5448    1.1932    1.4482    0.5003
   -0.5968   -2.6246   -1.4482   -1.9651
   -0.5968   -1.1932   -1.4482   -0.5003
solution = 1×4
    2.5448    1.1932    1.4482    0.5003
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.6997    2.3110   -1.4037    2.2343
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6997    2.3110   -1.4037    2.2343
    1.6997    0.9233    1.4037    0.8147
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6997    2.3110   -1.4037    2.2343
    1.6997    0.9233    1.4037    0.8147
   -1.4419   -2.3110   -1.4037   -2.2343
         0         0         0         0
angles = 4×4
    1.6997    2.3110   -1.4037    2.2343
    1.6997    0.9233    1.4037    0.8147
   -1.4419   -2.3110   -1.4037   -2.2343
   -1.4419   -0.9233   -1.4037   -0.8147
solution = 1×4
    1.6997    0.9233    1.4037    0.8147
x = 6.2423
y = -15.5140
z = 5.8069
p_dest = 1×3
    6.2423  -15.5140    9.8069
angles = 4×4
    0.3825    1.8924   -1.2162    2.4654
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.3825    1.8924   -1.2162    2.4654
    0.3825    0.6894    1.2162    1.2360
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.3825    1.8924   -1.2162    2.4654
    0.3825    0.6894    1.2162    1.2360
    3.5241   -1.8924   -1.2162   -2.4654
         0         0         0         0
angles = 4×4
    0.3825    1.8924   -1.2162    2.4654
    0.3825    0.6894    1.2162    1.2360
    3.5241   -1.8924   -1.2162   -2.4654
    3.5241   -0.6894   -1.2162   -1.2360
solution = 1×4
    0.3825    0.6894    1.2162    1.2360
jointAngles = 1×4
    0.6725    0.6894    1.2162    1.2360
angles = 4×4
    0.3825    2.3909   -1.2768    2.0275
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    0.3825    2.3909   -1.2768    2.0275
    0.3825    1.1281    1.2768    0.7367
         0         0         0         0
         0         0         0         0
```

```
angles = 4×4
    0.3825    2.3909   -1.2768    2.0275
    0.3825    1.1281    1.2768    0.7367
    3.5241   -2.3909   -1.2768   -2.0275
         0         0         0         0
angles = 4×4
    0.3825    2.3909   -1.2768    2.0275
    0.3825    1.1281    1.2768    0.7367
    3.5241   -2.3909   -1.2768   -2.0275
    3.5241   -1.1281   -1.2768   -0.7367
solution = 1×4
    0.3825    1.1281    1.2768    0.7367
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.7059    2.2818   -1.3847    2.2446
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7059    2.2818   -1.3847    2.2446
    1.7059    0.9127    1.3847    0.8442
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7059    2.2818   -1.3847    2.2446
    1.7059    0.9127    1.3847    0.8442
   -1.4357   -2.2818   -1.3847   -2.2446
         0         0         0         0
angles = 4×4
    1.7059    2.2818   -1.3847    2.2446
    1.7059    0.9127    1.3847    0.8442
   -1.4357   -2.2818   -1.3847   -2.2446
   -1.4357   -0.9127   -1.3847   -0.8442
solution = 1×4
    1.7059    0.9127    1.3847    0.8442
x = 1.3507
y = 14.8629
z = 3.2321
p_dest = 1×3
    1.3507   14.8629    7.2321
angles = 4×4
    3.0510    2.1918   -1.5576    2.5074
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.0510    2.1918   -1.5576    2.5074
    3.0510    0.6529    1.5576    0.9312
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.0510    2.1918   -1.5576    2.5074
    3.0510    0.6529    1.5576    0.9312
   -0.0906   -2.1918   -1.5576   -2.5074
         0         0         0         0
angles = 4×4
    3.0510    2.1918   -1.5576    2.5074
    3.0510    0.6529    1.5576    0.9312
   -0.0906   -2.1918   -1.5576   -2.5074
   -0.0906   -0.6529   -1.5576   -0.9312
solution = 1×4
   -0.0906   -0.6529   -1.5576   -0.9312
jointAngles = 1×4
   -0.1406   -0.6529   -1.5576   -0.9312
angles = 4×4
    3.0510    2.6470   -1.4312    1.9258
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.0510    2.6470   -1.4312    1.9258
    3.0510    1.2322    1.4312    0.4782
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.0510    2.6470   -1.4312    1.9258
    3.0510    1.2322    1.4312    0.4782
   -0.0906   -2.6470   -1.4312   -1.9258
         0         0         0         0
angles = 4×4
    3.0510    2.6470   -1.4312    1.9258
    3.0510    1.2322    1.4312    0.4782
   -0.0906   -2.6470   -1.4312   -1.9258
   -0.0906   -1.2322   -1.4312   -0.4782
solution = 1×4
   -0.0906   -1.2322   -1.4312   -0.4782
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
```

```
   -1.2490   -0.5994   -1.3452   -1.1970
angles = 4×4
    1.6950    2.3265   -1.4330    2.2481
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6950    2.3265   -1.4330    2.2481
    1.6950    0.9100    1.4330    0.7986
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6950    2.3265   -1.4330    2.2481
    1.6950    0.9100    1.4330    0.7986
   -1.4466   -2.3265   -1.4330   -2.2481
         0         0         0         0
angles = 4×4
    1.6950    2.3265   -1.4330    2.2481
    1.6950    0.9100    1.4330    0.7986
   -1.4466   -2.3265   -1.4330   -2.2481
   -1.4466   -0.9100   -1.4330   -0.7986
solution = 1×4
   -1.4466   -0.9100   -1.4330   -0.7986
x = -4.9760
y = 14.3099
z = 3.1821
p_dest = 1×3
   -4.9760   14.3099    7.1821
angles = 4×4
    3.4762    2.1828   -1.5283    2.4870
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.4762    2.1828   -1.5283    2.4870
    3.4762    0.6727    1.5283    0.9407
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.4762    2.1828   -1.5283    2.4870
    3.4762    0.6727    1.5283    0.9407
    0.3347   -2.1828   -1.5283   -2.4870
         0         0         0         0
angles = 4×4
    3.4762    2.1828   -1.5283    2.4870
    3.4762    0.6727    1.5283    0.9407
    0.3347   -2.1828   -1.5283   -2.4870
    0.3347   -0.6727   -1.5283   -0.9407
solution = 1×4
    0.3347   -0.6727   -1.5283   -0.9407
jointAngles = 1×4
    0.0547   -0.6727   -1.5283   -0.9407
angles = 4×4
    3.4762    2.6285   -1.3980    1.9111
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.4762    2.6285   -1.3980    1.9111
    3.4762    1.2464    1.3980    0.4972
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.4762    2.6285   -1.3980    1.9111
    3.4762    1.2464    1.3980    0.4972
    0.3347   -2.6285   -1.3980   -1.9111
         0         0         0         0
angles = 4×4
    3.4762    2.6285   -1.3980    1.9111
    3.4762    1.2464    1.3980    0.4972
    0.3347   -2.6285   -1.3980   -1.9111
    0.3347   -1.2464   -1.3980   -0.4972
solution = 1×4
    0.3347   -1.2464   -1.3980   -0.4972
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
   -1.2490   -0.5994   -1.3452   -1.1970
angles = 4×4
    1.6946    2.3278   -1.4395    2.2533
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6946    2.3278   -1.4395    2.2533
    1.6946    0.9049    1.4395    0.7973
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6946    2.3278   -1.4395    2.2533
    1.6946    0.9049    1.4395    0.7973
   -1.4470   -2.3278   -1.4395   -2.2533
         0         0         0         0
angles = 4×4
    1.6946    2.3278   -1.4395    2.2533
    1.6946    0.9049    1.4395    0.7973
   -1.4470   -2.3278   -1.4395   -2.2533
   -1.4470   -0.9049   -1.4395   -0.7973
solution = 1×4
   -1.4470   -0.9049   -1.4395   -0.7973
x = -11.4217
y = 11.3159
z = 3.4571
p_dest = 1×3
  -11.4217   11.3159    7.4571
angles = 4×4
    3.9316    2.1076   -1.3927    2.4267
         0         0         0         0
         0         0         0         0
         0         0         0         0
```

```
angles = 4×4
    3.9316    2.1076   -1.3927    2.4267
    3.9316    0.7307    1.3927    1.0182
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.9316    2.1076   -1.3927    2.4267
    3.9316    0.7307    1.3927    1.0182
    0.7901   -2.1076   -1.3927   -2.4267
         0         0         0         0
angles = 4×4
    3.9316    2.1076   -1.3927    2.4267
    3.9316    0.7307    1.3927    1.0182
    0.7901   -2.1076   -1.3927   -2.4267
    0.7901   -0.7307   -1.3927   -1.0182
solution = 1×4
    0.7901   -0.7307   -1.3927   -1.0182
jointAngles = 1×4
    0.5101   -0.7307   -1.3927   -1.0182
angles = 4×4
    3.9316    2.5356   -1.2797    1.8857
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.9316    2.5356   -1.2797    1.8857
    3.9316    1.2700    1.2797    0.5919
         0         0         0         0
         0         0         0         0
angles = 4×4
    3.9316    2.5356   -1.2797    1.8857
    3.9316    1.2700    1.2797    0.5919
    0.7901   -2.5356   -1.2797   -1.8857
         0         0         0         0
angles = 4×4
    3.9316    2.5356   -1.2797    1.8857
    3.9316    1.2700    1.2797    0.5919
    0.7901   -2.5356   -1.2797   -1.8857
    0.7901   -1.2700   -1.2797   -0.5919
solution = 1×4
    0.7901   -1.2700   -1.2797   -0.5919
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
   -1.2490   -0.5994   -1.3452   -1.1970
angles = 4×4
```

```
    1.7050    2.3215   -1.4328    2.2529
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7050    2.3215   -1.4328    2.2529
    1.7050    0.9051    1.4328    0.8037
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7050    2.3215   -1.4328    2.2529
    1.7050    0.9051    1.4328    0.8037
   -1.4366   -2.3215   -1.4328   -2.2529
         0         0         0         0
angles = 4×4
    1.7050    2.3215   -1.4328    2.2529
    1.7050    0.9051    1.4328    0.8037
   -1.4366   -2.3215   -1.4328   -2.2529
   -1.4366   -0.9051   -1.4328   -0.8037
solution = 1×4
   -1.4366   -0.9051   -1.4328   -0.8037
x = -13.5392
y = -15.6084
z = 5.4569
p_dest = 1×3
  -13.5392  -15.6084    9.4569
angles = 4×4
   -0.7145    1.4344   -0.1536    1.8608
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.7145    1.4344   -0.1536    1.8608
   -0.7145    1.2823    0.1536    1.7057
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.7145    1.4344   -0.1536    1.8608
   -0.7145    1.2823    0.1536    1.7057
    2.4271   -1.4344   -0.1536   -1.8608
         0         0         0         0
angles = 4×4
   -0.7145    1.4344   -0.1536    1.8608
   -0.7145    1.2823    0.1536    1.7057
    2.4271   -1.4344   -0.1536   -1.8608
    2.4271   -1.2823   -0.1536   -1.7057
solution = 1×4
   -0.7145    1.2823    0.1536    1.7057
jointAngles = 1×4
   -0.9045    1.2823    0.1536    1.7057
angles = 4×4
   -0.7145    1.8886   -0.2988    1.5518
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.7145    1.8886   -0.2988    1.5518
   -0.7145    1.5926    0.2988    1.2502
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.7145    1.8886   -0.2988    1.5518
   -0.7145    1.5926    0.2988    1.2502
    2.4271   -1.8886   -0.2988   -1.5518
```

```
         0         0         0         0
angles = 4×4
   -0.7145    1.8886   -0.2988    1.5518
   -0.7145    1.5926    0.2988    1.2502
    2.4271   -1.8886   -0.2988   -1.5518
    2.4271   -1.5926   -0.2988   -1.2502
solution = 1×4
   -0.7145    1.5926    0.2988    1.2502
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.6996    2.3057   -1.4069    2.2428
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6996    2.3057   -1.4069    2.2428
    1.6996    0.9148    1.4069    0.8199
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6996    2.3057   -1.4069    2.2428
    1.6996    0.9148    1.4069    0.8199
   -1.4420   -2.3057   -1.4069   -2.2428
         0         0         0         0
angles = 4×4
    1.6996    2.3057   -1.4069    2.2428
    1.6996    0.9148    1.4069    0.8199
   -1.4420   -2.3057   -1.4069   -2.2428
   -1.4420   -0.9148   -1.4069   -0.8199
solution = 1×4
    1.6996    0.9148    1.4069    0.8199
x = -16.1473
y = -10.6085
z = 5.3069
p_dest = 1×3
  -16.1473  -10.6085    9.3069
angles = 4×4
   -0.9895    1.7140   -0.7324    2.1599
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.9895    1.7140   -0.7324    2.1599
   -0.9895    0.9889    0.7324    1.4203
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.9895    1.7140   -0.7324    2.1599
   -0.9895    0.9889    0.7324    1.4203
    2.1521   -1.7140   -0.7324   -2.1599
         0         0         0         0
angles = 4×4
   -0.9895    1.7140   -0.7324    2.1599
   -0.9895    0.9889    0.7324    1.4203
    2.1521   -1.7140   -0.7324   -2.1599
    2.1521   -0.9889   -0.7324   -1.4203
solution = 1×4
   -0.9895    0.9889    0.7324    1.4203
jointAngles = 1×4
   -1.1795    0.9889    0.7324    1.4203
angles = 4×4
   -0.9895    2.1381   -0.7645    1.7680
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.9895    2.1381   -0.7645    1.7680
   -0.9895    1.3812    0.7645    0.9959
         0         0         0         0
         0         0         0         0
angles = 4×4
   -0.9895    2.1381   -0.7645    1.7680
   -0.9895    1.3812    0.7645    0.9959
    2.1521   -2.1381   -0.7645   -1.7680
         0         0         0         0
angles = 4×4
   -0.9895    2.1381   -0.7645    1.7680
   -0.9895    1.3812    0.7645    0.9959
    2.1521   -2.1381   -0.7645   -1.7680
    2.1521   -1.3812   -0.7645   -0.9959
solution = 1×4
   -0.9895    1.3812    0.7645    0.9959
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.7001    2.2977   -1.3970    2.2409
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7001    2.2977   -1.3970    2.2409
    1.7001    0.9166    1.3970    0.8280
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7001    2.2977   -1.3970    2.2409
    1.7001    0.9166    1.3970    0.8280
   -1.4414   -2.2977   -1.3970   -2.2409
         0         0         0         0
angles = 4×4
    1.7001    2.2977   -1.3970    2.2409
    1.7001    0.9166    1.3970    0.8280
   -1.4414   -2.2977   -1.3970   -2.2409
   -1.4414   -0.9166   -1.3970   -0.8280
solution = 1×4
    1.7001    0.9166    1.3970    0.8280
x = -18.1902
y = 4.5696
z = 3.9446
p_dest = 1×3
  -18.1902    4.5696    7.9446
angles = 4×4
    4.4663    1.8716   -0.9224    2.1924
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    4.4663    1.8716   -0.9224    2.1924
    4.4663    0.9586    0.9224    1.2606
         0         0         0         0
         0         0         0         0
angles = 4×4
    4.4663    1.8716   -0.9224    2.1924
    4.4663    0.9586    0.9224    1.2606
    1.3247   -1.8716   -0.9224   -2.1924
         0         0         0         0
angles = 4×4
    4.4663    1.8716   -0.9224    2.1924
    4.4663    0.9586    0.9224    1.2606
    1.3247   -1.8716   -0.9224   -2.1924
    1.3247   -0.9586   -0.9224   -1.2606
solution = 1×4
    1.3247   -0.9586   -0.9224   -1.2606
jointAngles = 1×4
    1.0447   -0.9586   -0.9224   -1.2606
angles = 4×4
    4.4663    2.2421   -0.8243    1.7238
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    4.4663    2.2421   -0.8243    1.7238
    4.4663    1.4260    0.8243    0.8912
         0         0         0         0
         0         0         0         0
angles = 4×4
    4.4663    2.2421   -0.8243    1.7238
    4.4663    1.4260    0.8243    0.8912
    1.3247   -2.2421   -0.8243   -1.7238
         0         0         0         0
angles = 4×4
```

```
    4.4663    2.2421   -0.8243    1.7238
    4.4663    1.4260    0.8243    0.8912
    1.3247   -2.2421   -0.8243   -1.7238
    1.3247   -1.4260   -0.8243   -0.8912
solution = 1×4
    1.3247   -1.4260   -0.8243   -0.8912
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
   -1.2490   -0.5994   -1.3452   -1.1970
angles = 4×4
    1.6903    2.3269   -1.4274    2.2421
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6903    2.3269   -1.4274    2.2421
    1.6903    0.9158    1.4274    0.7984
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.6903    2.3269   -1.4274    2.2421
    1.6903    0.9158    1.4274    0.7984
   -1.4513   -2.3269   -1.4274   -2.2421
         0         0         0         0
angles = 4×4
    1.6903    2.3269   -1.4274    2.2421
    1.6903    0.9158    1.4274    0.7984
   -1.4513   -2.3269   -1.4274   -2.2421
   -1.4513   -0.9158   -1.4274   -0.7984
solution = 1×4
   -1.4513   -0.9158   -1.4274   -0.7984
x = -17.9706
y = -6.2819
z = 5.0444
p_dest = 1×3
  -17.9706   -6.2819    9.0444
angles = 4×4
   -1.2345    1.7652   -0.8156    2.1920
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -1.2345    1.7652   -0.8156    2.1920
   -1.2345    0.9578    0.8156    1.3683
         0         0         0         0
         0         0         0         0
angles = 4×4
   -1.2345    1.7652   -0.8156    2.1920
   -1.2345    0.9578    0.8156    1.3683
    1.9071   -1.7652   -0.8156   -2.1920
         0         0         0         0
angles = 4×4
   -1.2345    1.7652   -0.8156    2.1920
   -1.2345    0.9578    0.8156    1.3683
    1.9071   -1.7652   -0.8156   -2.1920
    1.9071   -0.9578   -0.8156   -1.3683
solution = 1×4
   -1.2345    0.9578    0.8156    1.3683
jointAngles = 1×4
   -1.4245    0.9578    0.8156    1.3683
angles = 4×4
   -1.2345    2.1815   -0.8199    1.7800
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
   -1.2345    2.1815   -0.8199    1.7800
   -1.2345    1.3698    0.8199    0.9519
         0         0         0         0
         0         0         0         0
angles = 4×4
   -1.2345    2.1815   -0.8199    1.7800
   -1.2345    1.3698    0.8199    0.9519
    1.9071   -2.1815   -0.8199   -1.7800
         0         0         0         0
angles = 4×4
   -1.2345    2.1815   -0.8199    1.7800
   -1.2345    1.3698    0.8199    0.9519
    1.9071   -2.1815   -0.8199   -1.7800
    1.9071   -1.3698   -0.8199   -0.9519
solution = 1×4
   -1.2345    1.3698    0.8199    0.9519
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
         0         0         0         0
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
         0         0         0         0
angles = 4×4
    1.8925    1.9296   -1.3452    2.5572
    1.8925    0.5994    1.3452    1.1970
   -1.2490   -1.9296   -1.3452   -2.5572
   -1.2490   -0.5994   -1.3452   -1.1970
solution = 1×4
    1.8925    0.5994    1.3452    1.1970
angles = 4×4
    1.7010    2.2809   -1.3831    2.2438
         0         0         0         0
         0         0         0         0
         0         0         0         0
```

```
angles = 4×4
    1.7010    2.2809   -1.3831    2.2438
    1.7010    0.9134    1.3831    0.8450
         0         0         0         0
         0         0         0         0
angles = 4×4
    1.7010    2.2809   -1.3831    2.2438
    1.7010    0.9134    1.3831    0.8450
   -1.4406   -2.2809   -1.3831   -2.2438
         0         0         0         0
angles = 4×4
    1.7010    2.2809   -1.3831    2.2438
    1.7010    0.9134    1.3831    0.8450
   -1.4406   -2.2809   -1.3831   -2.2438
   -1.4406   -0.9134   -1.3831   -0.8450
solution = 1×4
    1.7010    0.9134    1.3831    0.8450
```

```
function armDemo(arb)
    % Main demo function to control a robotic arm to detect, pick, and place
colored blocks
    placed = 1;      % Counter for how many objects have been placed
    STATE = 1;       % State machine variable
    global focalLength principalPoint
    focalLength = [1.4053e+03, 1.4053e+03];
    principalPoint = [951.5109, 526.6981]; % Camera intrinsic parameters

    % State machine loop
    while STATE ~= 8
        switch STATE
            case 1  % Go to home position
                homePosition(arb)
                STATE = STATE + 1;
                pause(4)

            case 2  % Detect all target objects
                [targets, howMany] = findTargets();
                STATE = STATE + 1;

            case 3  % Go to the next target
                if placed <= howMany
                    prev = goToTarget(targets(placed), arb);
                    STATE = STATE + 1;
                    pause(5)
                end

            case 4  % Pick the object
                pickTarget(arb, prev);
                STATE = STATE + 1;
                pause(4)

            case 5  % Move to place location
                goToPlace(arb);
                STATE = STATE + 1;
                pause(6)

            case 6  % Place the object
                place(arb);
                placed = placed + 1;
                STATE = STATE + 1;
                pause(3)

            case 7  % Check if all objects are placed
                if placed > howMany
                    STATE = 8;
                    homePosition(arb);
                else
                    homePosition(arb);
                    STATE = 3;
                    pause(3)
                end

            otherwise
                STATE = 8;
        end
    end
end

function homePosition(arb)
    % Move arm to default home position
    arb.setpos([0, pi/3, pi/12, pi/2, 0], [100 100 100 100 100]);
end

function [targets, howMany] = findTargets()
    % Detect colored objects in the image
    global img ig BW
    [img, ig] = depth_example();  % Get image and depth info
    % Mask out unnecessary parts of the image
    img(:, 1:80, :) = 0;
    img(:, 500:640, :) = 0;
    img(425:480, :, :) = 0;
    img(1:65, :, :) = 0;

    figure; imshow(img);  % Show cropped image

    % Create masks for different colors
    [blueMask, ~] = createBlueMask(img);
    [yellowMask, ~] = createYellowMask(img);
    [redMask, ~] = createRedMask(img);
    [greenMask, ~] = createGreenMask(img);

    % Combine masks and extract edges
    BW = blueMask | redMask | yellowMask | greenMask;
    BW = edge(BW, 'Canny');
    figure; imshow(BW)

    % Find bounding boxes of detected regions
    targets = regionprops(BW, "BoundingBox");
    howMany = length(targets);
end

function prev = goToTarget(target, arb)
    % Move the robot to hover above the detected object
    global BW ig principalPoint pose_obj
    % Create a mask for current target bounding box
    cube_mask = false(size(BW));
    cube_mask(round(target.BoundingBox(2)):round(target.BoundingBox(2) +
target.BoundingBox(4)), ...
              round(target.BoundingBox(1)):round(target.BoundingBox(1) +
target.BoundingBox(3))) = true;
    cubes = cube_mask & BW;

    % Get object center and depth
    stats = regionprops(cubes, 'Centroid');
    object_center = stats.Centroid;
    row = round(object_center(2));
    col = round(object_center(1));
    depth = ig(row, col);

    % Convert 2D image coordinates to 3D camera coordinates
    [x, y, z] = pixelToCameraCoords(col, row, depth, principalPoint(1),
principalPoint(2));

    % Calibrate and shift to real-world coordinates
    x = (-x - 6) + 1;
    y = (y + 1.9) - 0.1;
    z = 70 - z;

    pose_obj = [x, y, z];
    p_dest = pose_obj + [0 0 4];  % Hover above the object

    currentPos = arb.getpos();
    jointAngles = findOptimalSolution(p_dest, currentPos(1:4));
    prev = jointAngles;

    % Move the arm to the target position
    arb.setpos([jointAngles 0], [60 60 60 60 60]);
end

function pickTarget(arb, prev)
    % Move arm down and close the gripper to pick object
    global pose_obj
    p_dest = pose_obj - [0 0 4];  % Move down 4 cm
    currentPos = arb.getpos();
    jointAngles = findOptimalSolution(p_dest, currentPos(1:4));
    arb.setpos([jointAngles 0], [60 60 60 60 60]);
    pause(3)

    % Adjust wrist if object is in a specific quadrant
    x = p_dest(1); y = p_dest(2);
    if x < 0 && y < 0
        jointAngles = jointAngles - [0 0 0 -0.3];
        arb.setpos([jointAngles 0], [60 60 60 60 60]);
        pause(1)
    end

    % Close gripper to grab the object
    temp = arb.getpos();
    arb.setpos([temp(1:4), 1.2], [60 60 60 60 60]);
    pause(2)

    % Lift back up
    arb.setpos([prev 1.2], [60 60 60 60 60]);
end

function goToPlace(arb)
    % Move to fixed placement location
    p_dest = [15 5 10];
    currentPos = arb.getpos();
    jointAngles = findOptimalSolution(p_dest, currentPos(1:4));
    arb.setpos([jointAngles 1.2], [60 60 60 60 60]);
    pause(3)
end

function place(arb)
    % Place the object and open the gripper
    currentPos = arb.getpos();
    [x, y, z, ~] = pincherFK(currentPos);
    pose_obj = [x, y, z];
    p_dest = pose_obj - [-1 2 5];  % Slight offset for drop
    jointAngles = findOptimalSolution(p_dest, currentPos(1:4));
    arb.setpos([jointAngles currentPos(5)], [60 60 60 60 60]);
    pause(3)

    % Open the gripper
    temp = arb.getpos();
    arb.setpos([temp(1:4), 0], [60 60 60 60 60]);
    pause(3)
    arb.setpos([currentPos(1:4) 0], [60 60 60 60 60]);
end

function [X_cm, Y_cm, Z_cm] = pixelToCameraCoords(u, v, Z_m, cx, cy)
    % Convert pixel coordinates to 3D world coordinates (in cm)
    u0 = cx * (640/1920);  % Adjust for actual resolution
    v0 = cy * (480/1080);

    X = ((u - u0) * Z_m) / 474.8976;
    Y = ((v - v0) * Z_m) / 474.8976;

    X_cm = X * 100;
    Y_cm = Y * 100;
    Z_cm = Z_m * 100;
end

%% === Kinematics Helper Functions ===

function vs = skew(v)
    % Returns skew-symmetric matrix of a 3D vector
    vs = [0 -v(3) v(2);
          v(3) 0 -v(1);
         -v(2) v(1) 0];
end

function T = exp(S, theta)
    % Matrix exponential using screw axis S and angle theta
    w_skew = skew(S(1:3)/norm(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 [S1, S2, S3, S4, Tsb] = getTsb()
    % Define screw axes and forward kinematics chain
    l_1=10; l_2=4.5; l_3=10.7; l_4=10.5;
    w1 = [0;0;1]; q1 = [0;0;l_1];
    S1 = [w1;cross(-w1,q1)];

    w2 = [1;0;0]; q2 = [0;0;l_1+l_2];
    S2 = [w2;cross(-w2,q2)];

    w3 = [1;0;0]; q3 = [0;0;l_1+l_2+l_3];
    S3 = [w3;cross(-w3,q3)];

    w4 = [1;0;0]; q4 = [0;0;l_1+l_2+l_3+l_4];
    S4 = [w4;cross(-w4,q4)];

    % Symbolic transformation matrix
    syms theta_1 theta_2 theta_3 theta_4
    Tsb = exp(S1,theta_1)*exp(S2,theta_2)*exp(S3,theta_3)*exp(S4,theta_4);
end

function zeroConfig= getZeroConf()
```

```
    % Get zero configuration transformation matrix M
    l_1=10; l_2=4.5; l_3=10.7; l_4=10.5; l_5=9.5;
    zeroConfig = [1 0 0 0;
                  0 1 0 0;
                  0 0 1 l_1 + l_2 + l_3 + l_4 + l_5;
                  0 0 0 1];
end

function [x,y,z,R] = pincherFK(jointAngles)
    % Forward Kinematics to compute end-effector position and rotation
    M = getZeroConf();
    syms theta_1 theta_2 theta_3 theta_4
    [~, ~, ~, ~, Tsb] = getTsb();
    T = double(subs(Tsb, [theta_1 theta_2 theta_3 theta_4], jointAngles(1:4)) * M);
    x = T(1,4); y = T(2,4); z = T(3,4);
    R = T(1:3, 1:3);  % Rotation matrix
end

function angles = findJointAngles(x,y,z,phi)
    % Initialize a 4x4 matrix to store all possible joint angle solutions
    angles = zeros(4,4);

    % Link lengths of the robotic arm (in appropriate units)
    l_1 = 10;
    l_2 = 4.5;
    l_3 = 10.7;
    l_4 = 10.5;
    l_5 = 9.5;

    % Calculate two possible solutions for theta1 (base rotation)
    theta1_1 = mod(atan2(y,x) + pi, 2*pi) - pi;
    theta1_2 = mod(atan2(y,x) + pi + pi, 2*pi) - pi;

    % Distance in the horizontal plane from base to target
    r = sqrt(x^2 + y^2);
    % Height difference from base to target
    s = z - (l_1 + l_2);

    % Adjust target to account for orientation angle phi and wrist length l_5
    r_ = r - l_5 * cos(phi);
    s_ = s - l_5 * sin(phi);

    % Compute the cosine of angle between l3 and l4 using law of cosines
    D = ((r_ * r_) + (s_ * s_) - l_3^2 - l_4^2) / (2 * l_3 * l_4);

    % Calculate the sine component (numerator) for inverse kinematics
    num = sqrt(1 - D * D);

    % Two possible solutions for elbow joint angle (theta3)
    theta3_1 = mod(atan2(num, D) + pi, 2*pi) - pi;
    theta3_2 = mod(atan2(-num, D) + pi, 2*pi) - pi;

    % Compute two corresponding shoulder angles (theta2) using geometry
    theta2_1 = mod(atan2(s_, r_) - atan2(l_4 * sin(theta3_1), l_3 + l_4 *
cos(theta3_1)) + pi, 2*pi) - pi;
    theta2_2 = mod(atan2(s_, r_) - atan2(l_4 * sin(theta3_2), l_3 + l_4 *
cos(theta3_2)) + pi, 2*pi) - pi;

    % Compute wrist joint angles to satisfy total orientation phi
    theta4_1 = mod((phi - theta2_1 - theta3_1) + pi, 2*pi) - pi;
    theta4_2 = mod((phi - theta2_2 - theta3_2) + pi, 2*pi) - pi;

    % Combine all possible sets of solutions for the 4 joints
    angles(1,:) = [theta1_1 + pi/2, -theta2_1 + pi/2, -theta3_1, -theta4_1];
    angles(2,:) = [theta1_1 + pi/2, -theta2_2 + pi/2, -theta3_2, -theta4_2];
    angles(3,:) = [theta1_2 + pi/2, -(-theta2_1 + pi) + pi/2, -theta3_1, theta4_1];
    angles(4,:) = [theta1_2 + pi/2, -(-theta2_2 + pi) + pi/2, theta3_2, theta4_2];

end

function solution = findOptimalSolution(desiredPos, currentPos)
    % Extract target x, y, z coordinates
    x = desiredPos(1);
    y = desiredPos(2);
    z = desiredPos(3);

    % Desired end-effector orientation angle (constant here)
    phi = -pi/2;

    % Find all possible inverse kinematics solutions
    solutions = findJointAngles(x, y, z, phi);

    % Check each solution against joint limits
    B = cellfun(@checkJointLimits, num2cell(solutions, 2), 'UniformOutput', false);
    B = cell2mat(B);  % Convert logical results into matrix form

    % Filter valid solutions (where all joint angles are within limits)
    valid_solutions_mask = all(B, 2);
    valid_solutions = solutions(valid_solutions_mask, :);

    % If no valid solutions are found, return empty
    if isempty(valid_solutions)
        solution = [];
        return;
    end

    % Calculate absolute difference between each solution and current position
    delta = abs(valid_solutions - currentPos(1, 1:4));

    % Sum of differences for each solution (used for scoring)
    s = sum(delta, 2);

    % Select the solution with the minimum error (closest to current position)
    [~, idx] = min(s);
    solution = valid_solutions(idx, :);

    % Empirical bias correction based on workspace quadrant
    if x < 0 && y > 0
        solution = solution - [0.28 0 0 0];
    elseif x > 0 && y > 0
        solution = solution - [0.05 0 0 0];
    elseif x > 0 && y < 0
        solution = solution + [0.29 0 0 0];
    elseif x < 0 && y < 0
        solution = solution - [0.19 0 0 0];
    end
end

function isValid = checkJointLimits(jointAngles)
    % Joint angle limits in radians (-150 to 150 degrees)
    thetaLimits = [-150*pi/180, 150*pi/180];

    % Return true if all joint angles are within limits
    isValid = all(jointAngles >= thetaLimits(1) & jointAngles <= thetaLimits(2));
end
```

```
% Mask creation for each cube;

% YELLOW MASK
function [BW,maskedRGBImage] = createMask(RGB)
%createMask  Threshold RGB image using auto-generated code from colorThresholder
app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 12-May-2025
%------------------------------------------------------



% Convert RGB image to chosen color space
I = rgb2lab(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 50.389;
channel1Max = 87.324;
```

```
% Define thresholds for channel 2 based on histogram settings
channel2Min = -1.696;
channel2Max = 10.460;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 16.667;
channel3Max = 34.858;

% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW = sliderBW;
% Combine both masks
BW = imfill(BW,"holes");
BW = bwpropfilt(BW, 'Area', [350 2000]);
se = strel('rectangle',[3 2]);
% BW = imopen(BW, se);

if BW ~= 0
    BW = activecontour(RGB, BW,20,"Chan-vese", 'SmoothFactor',1.5);
end
BW = bwpropfilt(BW, 'Area', [350 2000]);
% BW = imopen(BW, se);
BW = imfill(BW,"holes");

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

% REDMASK
function [BW,maskedRGBImage] = createMask(RGB)
%createMask  Threshold RGB image using auto-generated code from colorThresholder
app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 12-May-2025
%-------------------------------------------------------


% Convert RGB image to chosen color space
I = rgb2lab(RGB);
```

```
% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.000;
channel1Max = 54.172;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 6.445;
channel2Max = 26.799;

% Define thresholds for channel 3 based on histogram settings
channel3Min = -6.813;
channel3Max = 12.653;

% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW = sliderBW;
% Combine both masks
BW = imfill(BW,"holes");
BW = bwpropfilt(BW, 'Area', [350 2000]);
se = strel('rectangle',[3 2]);
% BW = imopen(BW, se);

if BW ~= 0
    BW = activecontour(RGB, BW,20,"Chan-vese", 'SmoothFactor',1.5);
end
BW = bwpropfilt(BW, 'Area', [350 2000]);
% BW = imopen(BW, se);
BW = imfill(BW,"holes");

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

% GREEN MASK
function [BW,maskedRGBImage] = createMask(RGB)
%createMask  Threshold RGB image using auto-generated code from colorThresholder
app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 06-May-2025
```

```matlab
%-------------------------------------------------------


% Convert RGB image to chosen color space
I = rgb2lab(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.000;
channel1Max = 98.864;

% Define thresholds for channel 2 based on histogram settings
channel2Min = -24.840;
channel2Max = 20.097;

% Define thresholds for channel 3 based on histogram settings
channel3Min = -27.470;
channel3Max = 34.436;

% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Create mask based on selected regions of interest on point cloud projection
I = double(I);
[m,n,~] = size(I);
polyBW = false([m,n]);
I = reshape(I,[m*n 3]);
temp = I(:,1);
I(:,1) = I(:,2);
I(:,2) = I(:,3);
I(:,3) = temp;
clear temp

% Project 3D data into 2D projected view from current camera view point within app
J = rotateColorSpace(I);

% Apply polygons drawn on point cloud in app
polyBW = applyPolygons(J,polyBW);

% Combine both masks
BW = sliderBW & polyBW;
BW = imfill(BW,"holes");
BW = bwpropfilt(BW, 'Area', [350 2000]);
se = strel('rectangle',[3 2]);
% BW = imopen(BW, se);

if BW ~= 0
    BW = activecontour(RGB, BW,20,"Chan-vese", 'SmoothFactor',1.5);
end
```

```matlab
BW = bwpropfilt(BW, 'Area', [350 2000]);
% BW = imopen(BW, se);
BW = imfill(BW,"holes");

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

function J = rotateColorSpace(I)

% Translate the data to the mean of the current image within app
shiftVec = [-3.184231 -6.352806 41.636915];
I = I - shiftVec;
I = [I ones(size(I,1),1)]';

% Apply transformation matrix
tMat = [-0.021608 -0.000019 -0.000000 0.500775;
    -0.000002 0.000828 0.008965 -0.532555;
    0.000033 -0.012233 0.000606 9.124592;
    0.000000 0.000000 0.000000 1.000000];

J = (tMat*I)';
end

function polyBW = applyPolygons(J,polyBW)

% Define each manually generated ROI
hPoints(1).data = [0.701340 -0.276163;
    0.624691 -0.773047;
    1.001550 -0.816773;
    1.001550 -0.451066];

% Iteratively apply each ROI
for ii = 1:length(hPoints)
    if size(hPoints(ii).data,1) > 2
        in = inpolygon(J(:,1),J(:,2),hPoints(ii).data(:,1),hPoints(ii).data(:,2));
        in = reshape(in,size(polyBW));
        polyBW = polyBW | in;
    end
end

end


% BLUEMASK
function [BW,maskedRGBImage] = createMask(RGB)
```

```matlab
%createMask  Threshold RGB image using auto-generated code from colorThresholder
app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 05-May-2025
%------------------------------------------------------


% Convert RGB image to chosen color space
I = rgb2lab(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.000;
channel1Max = 46.991;

% Define thresholds for channel 2 based on histogram settings
channel2Min = -22.812;
channel2Max = 17.679;

% Define thresholds for channel 3 based on histogram settings
channel3Min = -26.058;
channel3Max = -18.877;

% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Create mask based on selected regions of interest on point cloud projection
I = double(I);
[m,n,~] = size(I);
polyBW = false([m,n]);
I = reshape(I,[m*n 3]);
temp = I(:,1);
I(:,1) = I(:,2);
I(:,2) = I(:,3);
I(:,3) = temp;
clear temp

% Project 3D data into 2D projected view from current camera view point within app
J = rotateColorSpace(I);

% Apply polygons drawn on point cloud in app
polyBW = applyPolygons(J,polyBW);

% Combine both masks
```

```
BW = sliderBW & polyBW;
BW = imfill(BW,"holes");
BW = bwpropfilt(BW, 'Area', [350 2000]);
se = strel('rectangle',[3 2]);
% BW = imopen(BW, se);
% BW = imfill(BW,"holes");
if BW ~= 0
    BW = activecontour(RGB, BW,20,"Chan-vese", 'SmoothFactor',1.5);
end
BW = bwpropfilt(BW, 'Area', [350 2000]);
% BW = imopen(BW, se);
BW = imfill(BW,"holes");

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end

function J = rotateColorSpace(I)

% Translate the data to the mean of the current image within app
shiftVec = [-3.348622 -5.784846 41.274362];
I = I - shiftVec;
I = [I ones(size(I,1),1)]';

% Apply transformation matrix
tMat = [-0.003974 -0.011834 0.000000 0.572981;
    -0.000067 0.000005 0.008679 -0.498882;
    0.024737 -0.001901 0.000023 8.244551;
    0.000000 0.000000 0.000000 1.000000];

J = (tMat*I)';
end

function polyBW = applyPolygons(J,polyBW)

% Define each manually generated ROI
hPoints(1).data = [0.530592 -0.085298;
    0.691377 -0.721582;
    0.983715 -0.646725;
    0.784925 -0.047870];

% Iteratively apply each ROI
for ii = 1:length(hPoints)
    if size(hPoints(ii).data,1) > 2
        in = inpolygon(J(:,1),J(:,2),hPoints(ii).data(:,1),hPoints(ii).data(:,2));
        in = reshape(in,size(polyBW));
```

```
            polyBW = polyBW | in;
        end
    end

    end
```

STATE machine logic: The robot operates in a loop with defined states from 1 to 8, executing a sequence:

1. **Home Position** – Arm resets.
2. **Find Targets** – Uses computer vision to detect colored objects.
3. **Move to Target** – Navigates to detected object's location.
4. **Pick Object** – Gripper moves to pick.
5. **Move to Placement Zone** – Moves to predefined place zone.
6. **Place Object** – Places the object down.
7. **Loop Check** – Repeats if more objects are present.
8. **End** – Ends operation after all objects handled.

Function	Description
`homePosition`	Resets the arm to initial position.
`findTargets`	Filters image masks for colors (red, blue, yellow, green), extracts bounding boxes.
`goToTarget`	Converts image pixel to camera coordinates, then IK to joint angles.
`pickTarget`	Moves end-effector down to pick the object, closes gripper.
`goToPlace`	Moves to a fixed placement location.
`place`	Lowers object and opens gripper.
`pixelToCameraCoords`	Converts image space to real-world space using camera intrinsics.
`findOptimalSolution`	(Assumed) inverse kinematics solver to calculate joint angles.
`pincherFK`	Forward kinematics based on screw theory and exponential coordinates.

`pixelToCameraCoords:`

Converts 2D pixel coordinates `(u, v)` from a camera image into real-world 3D coordinates `(X_cm, Y_cm, Z_cm)` in **centimeters.**

Step 1 – Scale Principal Point to Working Resolution (640×480)

```
u0 = cx * (640/1920);  % Adjust for actual resolution
v0 = cy * (480/1080);
```

Step 2 – Convert Pixel Coordinates to Real-World Coordinates

Here, `fx = fy = 474.8976` (focal length in pixels)

```
X = ((u - u0) * Z_m) / 474.8976;
Y = ((v - v0) * Z_m) / 474.8976;
```

Step 3 – Convert Meters to Centimeters

X_cm = X * 100;

Y_cm = Y * 100;

Z_cm = Z_m * 100;

Algorithm Overview

1. **Inverse Kinematics Calculation**:The function `findJointAngles(x, y, z, phi)` computes all four possible inverse kinematics (IK) solutions for the robotic arm. It:

- Computes the base rotation angle theta1 using the `atan2` of the planar target.
- Adjusts the target position by subtracting the wrist offset.
- Solves for the elbow angle theta3 using the cosine law and considers both positive and negative elbow configurations.
- Computes the shoulder angle theta2 using the geometric relationship between the links.
- Deduces the wrist joint angle theta4 to align with the desired end-effector orientation ϕ\phi.
- Produces four full joint configurations considering symmetrical solutions.

1. **Joint Limit Filtering**:The `checkJointLimits` function validates whether each joint angle in a solution lies within physical limits (±150°). This ensures mechanical safety and avoids invalid configurations.
2. **Optimal Solution Selection**:The `findOptimalSolution(desiredPos, currentPos)` function:

- Evaluates all valid IK solutions and selects the one closest to the current joint state using L1-norm (sum of absolute differences).
- Applies quadrant-based angle corrections for fine-tuned positioning accuracy. These are empirical offsets observed to improve real-world grasp alignment in different workspace quadrants.

Performance Results

The algorithm demonstrates grasping performance with the following observed characteristics:

- **Accuracy**:The use of quadrant-based offset corrections significantly improved grasping precision, especially in regions where inverse kinematics produced slightly misaligned configurations.
- **Responsiveness**:The computational load of evaluating four IK solutions and filtering them is minimal, enabling real-time performance during grasping tasks.
- **Reliability**:The joint limit checking prevented mechanical overextension, however, the edge cases showed poor performance

- **Versatility**:The algorithm correctly handles multiple inverse kinematics configurations and dynamically selects the most suitable one, allowing flexible grasping from multiple approach angles however, slight misalignment in the configurations was observed

Observed Errors

1. **Inaccurate Centroid Detection:** The most significant source of error was the **perception system's incorrect estimation of object centroids**. These errors led to incorrect coordinate values being passed to the inverse kinematics module, which resulted in misplaced grasping. Most likely, the area for the surface of the cuboid was not sufficient enough to correctly determine the centroid.
2. **Axis Boundary Misclassification:** At edge cases where x=0 or y=0, the robot occasionally applied **incorrect quadrant-based offsets**. These offsets, meant to fine-tune the joint angles, unintentionally pushed the placement coordinates into adjacent quadrants due to the absence of special handling for axis-aligned positions.
3. **Place Position:** Unlike the grasping targets, which were dynamically observed from the camera feed, the **placement position was static** in the system. This inflexibility introduced a mismatch between real-time environmental conditions and the robot's actuation, especially if the object's dimensions or orientation changed slightly after grasping. Since the place position was static, it showed the affect of offsets which is why the place psoition was not at the intended position i.e. too much offset misaligned the place position.

Strategies to improve the observed errors:

1. **Improve the perception**; the masks can be further improved to give more accurate centroids.
2. **Improve the camera calibration**; we are using the in built fx and fy for the RGB camera provided by the determineExtrinsics() function. This could possibly have led to error in the coorinates and the need for offsets. The mitigation startegy could be to analytically determine the fx and fy and that origin of the camera frame matches with the origin of the base frame of the robot.

Our inverse kinematics function was working fine however, since the perception was not correctly giving the centroid, offsets had to be introduced in the joint angles.

Test Run video:

https://www.canva.com/design/DAGnQKTh40Q/wICm5PPqqv-w6KXObQaHzw/edit?utm_content=DAGnQKTh40Q&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton