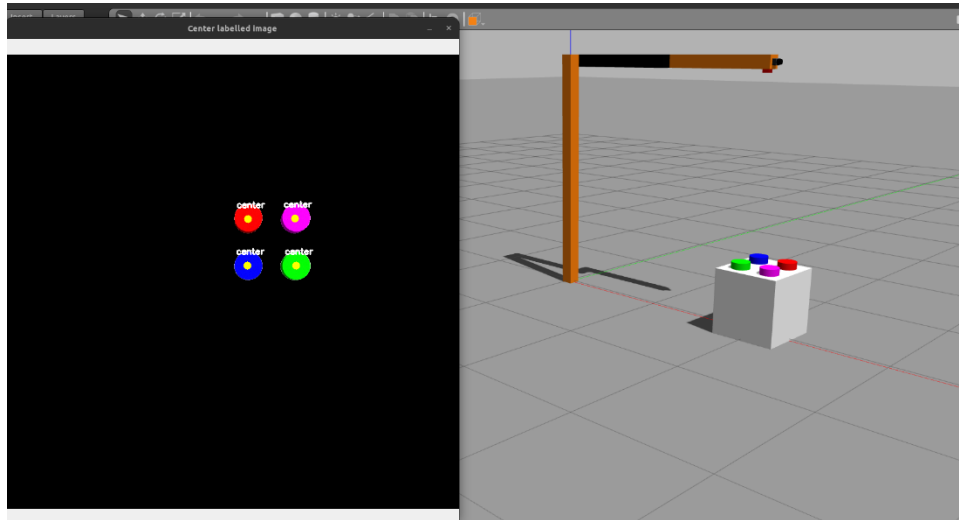


## RBE 450X - Vision Based Manipulation

### HW - 04

#### Step 1: Initial Spawn



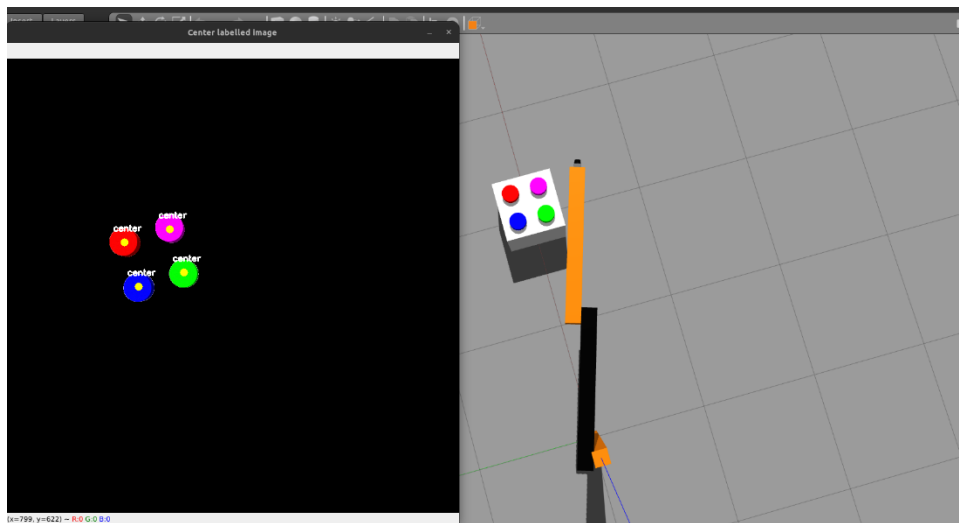
Green Center coordinate: (512, 371)

Blue Center coordinate: (426, 371)

Meg Center coordinate: (510, 288)

Red Center coordinate: (427, 289)

Trajectory start position:

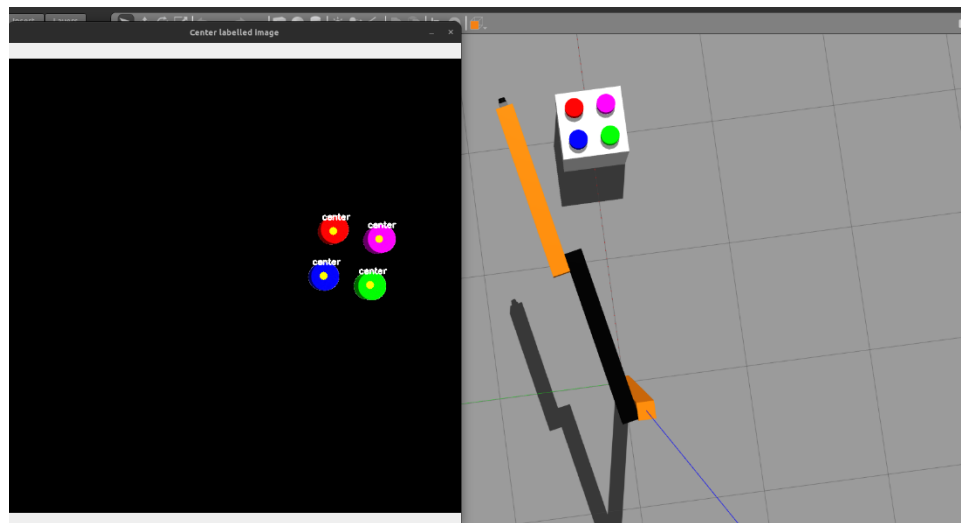


Green Center coordinate: (312, 376)

Blue Center coordinate: (234, 400)

Meg Center coordinate: (289, 299)

Red Center coordinate: (210, 323)



Green Center coordinate: (639, 398)

Blue Center coordinate: (557, 382)

Meg Center coordinate: (655, 317)

Red Center coordinate: (574, 303)

```

RBE450X_ROS2_Control.md RBE450X-Gazebo.md vserve.py x
home > gs > rbe450x-ros2 > src > visualserve > vserve.py > ImageSubscriber > listener_callback
171 # Find the center of the circle
172 def findCenters(self, img, disp):
173     # Finding contour centers
174     cnts, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
175     centers = []
176     for c in cnts:
177         if cv2.contourArea(c) <= 25:
178             continue
179         cx = int(average([x for x in c[:,0]]))
180         cy = int(average([y for y in c[:,1]]))
181         center = (cx, cy)
182         centers.append(center)
183     return centers
184
185
186
187
188 def masks(self, mask, img):
189     imask = mask>0
190     color = np.zeros_like(img, np.uint8)
191     color[imask] = img[imask]
192     return self.findCenters(mask, color)
193

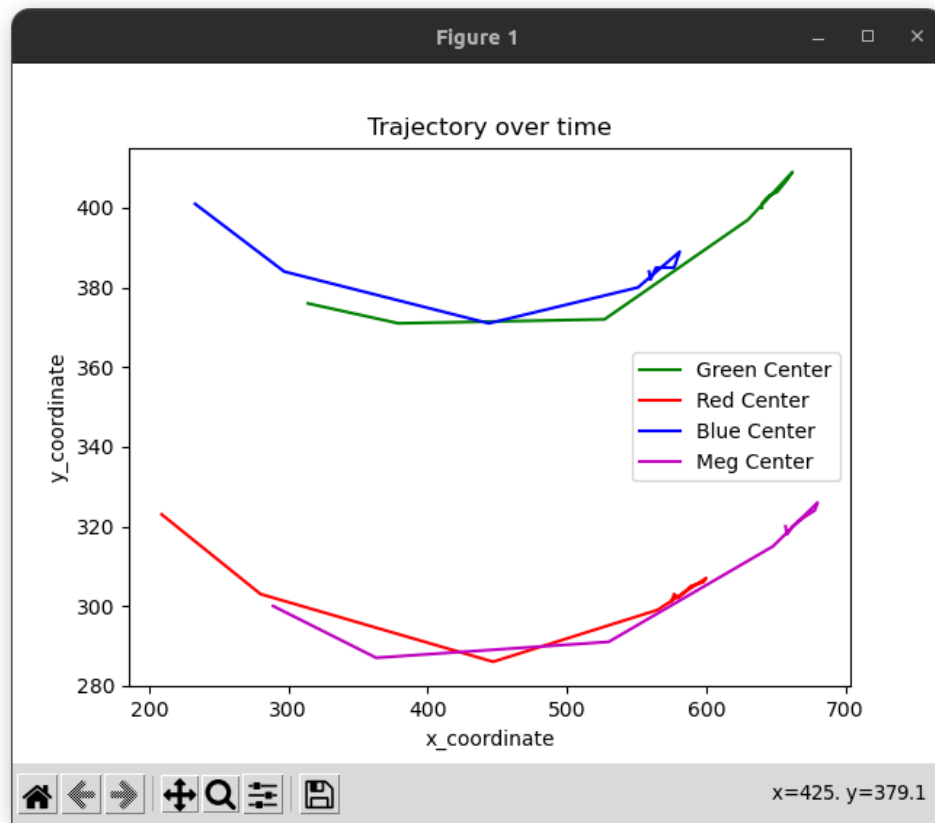
```

```

RBE450X_ROS2_Control.md RBE450X-Gazebo.md vserve.py x
home > gs > rbe450x-ros2 > src > visualserve > vserve.py > ImageSubscriber > listener_callback
114 def colorThreshold(self, img):
115     # Color thresholding
116     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
117     # Green
118     gmask = cv2.inRange(hsv, (30, 25, 25), (70, 255, 255))
119     green_centers=self.masks(gmask, img)
120     self.gcx.append(green_centers[0][0])
121     self.gcy.append(green_centers[0][1])
122     # Blue
123     bmask = cv2.inRange(hsv, (95, 25, 25), (135, 255, 255))
124     blue_centers=self.masks(bmask, img)
125     self.bcx.append(blue_centers[0][0])
126     self.bcy.append(blue_centers[0][1])
127     # Red
128     rmask = cv2.inRange(hsv, (0, 180, 25), (15, 255, 255))
129     red_centers=self.masks(rmask, img)
130     self.rcx.append(red_centers[0][0])
131     self.rcy.append(red_centers[0][1])
132     #Meg
133     mmask = cv2.inRange(hsv, (140, 100, 20), (170, 255, 255))
134     meg_centers=self.masks(mmask, img)
135     self.mcx.append(meg_centers[0][0])
136     self.mcy.append(meg_centers[0][1])
137     #current features and error calculation
138     f_c = np.array([green_centers, blue_centers, red_centers, meg_centers])
139     f_c = f_c.flatten()# 4 x 1
140     error = [0,0,0,0,0,0,0]
141     error = f_c - self.f_r
142     print("error", error)
143     self.find(error)
144

```

The coordinates of the centers over time are plotted as follows:



```

72 def findCenters(self, img, disp):
73     # Finding contour centers
74     cnts, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
75     centers = []
76     for c in cnts:
77         if cv2.contourArea(c) <= 25:
78             continue
79         cx = int(average(x for x in c[:,0]))
80         cy = int(average(y for y in c[:,1]))
81         center = (cx, cy)
82         centers.append(center)
83     return centers
84
85 def checkConvergence(self, err):
86     x_sum, y_sum = 0, 0
87     x_sum = sum(abs(i) for i in err[::2])
88     y_sum = sum(abs(i) for i in err[1::2])
89     if x_sum < 10 and y_sum < 10:
90         return True
91     else:
92         return False
93
94 def plot(self, convergence_val):
95     if convergence:
96         plt.plot(self.gcx, self.gcy, color='g', label='Green Center')
97         plt.plot(self.rcx, self.rcy, color='r', label='Red Center')
98         plt.plot(self.bcx, self.bcy, color='b', label='Blue Center')
99         plt.plot(self.mcx, self.mcy, color='m', label='Meg Center')
100     plt.xlabel('x coordinate')
101     plt.ylabel('y coordinate')
102     plt.legend()
103     plt.title('Trajectory of centers over time')
104     plt.show()
105     rospy.shutdown()
106     self.velocity_publisher.publish(val)
107
108 def mask(self, mask, img):
109     mask = mask>0
110     color = np.zeros_like(img, np.uint8)
111     color[mask] = img[mask]
112     return self.findCenters(mask, color)

```

```

RBE430X_ROS2_Control.md  RBE430X-Gazebo.md  vserve.py x
home > gs > rbe430x-ros2 > src > visualserve > visualserve > vserve.py > ImageSubscriber > listener_callback
114
115 def colorThreshold(self,img):
116     # Color thresholding
117     hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
118     # Green
119     gmask = cv2.inRange(hsv, (30, 25, 25), (70, 255, 255))
120     green_centers=self.masks(gmask,img)
121     self.gcx.append(green_centers[0][0])
122     self.gcy.append(green_centers[0][1])
123     # Blue
124     bmask = cv2.inRange(hsv, (90, 25, 25), (135, 255, 255))
125     blue_centers=self.masks(bmask,img)
126     self.bcx.append(blue_centers[0][0])
127     self.bcy.append(blue_centers[0][1])
128     # Red
129     rmask = cv2.inRange(hsv, (0, 100, 25), (15, 255, 255))
130     red_centers=self.masks(rmask,img)
131     self.rcx.append(red_centers[0][0])
132     self.rcy.append(red_centers[0][1])
133     # Mag
134     mmask = cv2.inRange(hsv, (140,100,20), (170,255,255))
135     mag_centers=self.masks(mmask,img)
136     self.mcx.append(mag_centers[0][0])
137     self.mcy.append(mag_centers[0][1])
138     # Feature extraction and error calculation
139     f_c = np.array([green_centers,blue_centers,red_centers,mag_centers])
140     f_c = f_c.flatten()# 0 x 1
141     error = [0,0,0,0,0,0,0]
142     error = f_c - self.f_r
143     print("error",error)
144     self.find(error)

```

```

RBE430X_ROS2_Control.md  RBE430X-Gazebo.md  vserve.py x
home > gs > rbe430x-ros2 > src > visualserve > visualserve > vserve.py > ImageSubscriber > listener_callback
145 def find(self,error,lbda = -0.00225):
146     l_e_l = -1*np.eye(2)
147     l_e = np.vstack((l_e_l,l_e_l,l_e_l,l_e_l))
148     l_e_inv = np.linalg.pinv(l_e)
149     v_c = -lbda*l_e_inv @ error
150     v_c = np.concatenate((v_c,[0,1])).reshape((-1))
151
152     try:
153         now = rospy.time.Time()
154         trans = self.tf_buffer.lookup_transform("link1","camera_link",now)
155     except TransformException as ex:
156         self.get_logger().info(
157             f'Could not transform ("link1") to ("camera_link"): {ex}')
158         return
159     Rotation matrix = quaternion_matrix([trans.transform.rotation.x,
160                                           trans.transform.rotation.y,
161                                           trans.transform.rotation.z,
162                                           trans.transform.rotation.w])
163     v_w = Rotation matrix @ v_c # 4 x 1 # link1 frame velocities
164     trans_link1_camera = [trans.transform.translation.x,
165                           trans.transform.translation.y,
166                           trans.transform.translation.z]
167
168     try:
169         now = rospy.time.Time()
170         trans = self.tf_buffer.lookup_transform("link2","camera_link",now)
171     except TransformException as ex:
172         self.get_logger().info(
173             f'Could not transform ("link2") to ("camera_link"): {ex}')
174         return
175     trans_link2_camera = [trans.transform.translation.x,
176                           trans.transform.translation.y,
177                           trans.transform.translation.z]
178
179     J11 = np.cross(np.array([0,0,1]),trans_link1_camera)
180     J12 = np.cross(np.array([0,0,1]),trans_link2_camera)
181     J21 = np.array([0,0,1])
182     J22 = J21
183     J = np.hstack((np.vstack((J11,J12)),np.vstack((J21,J22)))) # 6 x 2
184     J_inv = np.linalg.pinv(J) # 2 x 6
185     j_v = J_inv @ v_c # 2 x 2 @ 2 x 1
186
187     # publish
188     val = Float64MultiArray()
189     val.data = np.array([j_v[0],j_v[1]]).astype(np.float64).tolist()
190     # self.velocity_publisher.publish(val)
191     self.plot(self.checkConvergence(error),val)
192

```