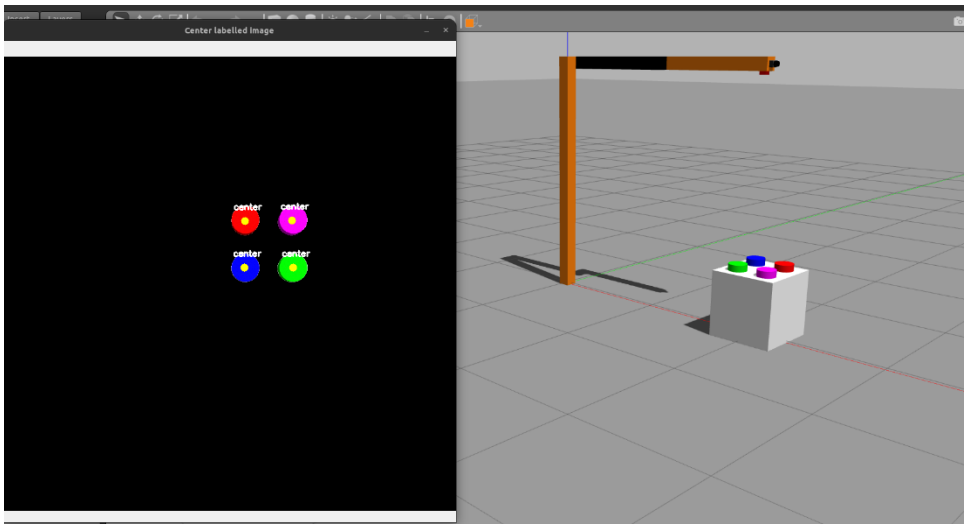
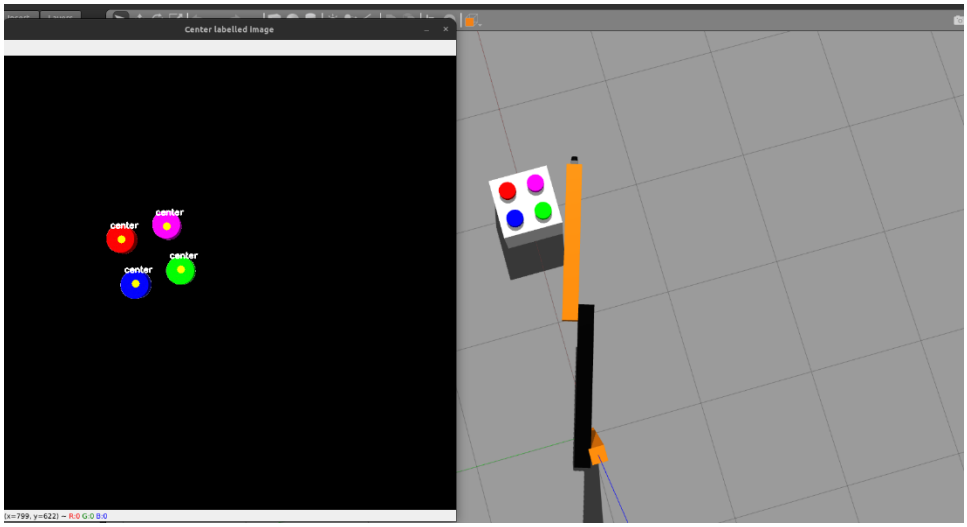
Gokul Srinivasan 500594698

RBE 450X - Vision Based Manipulation
HW - 04

Step 1: Initial Spawn



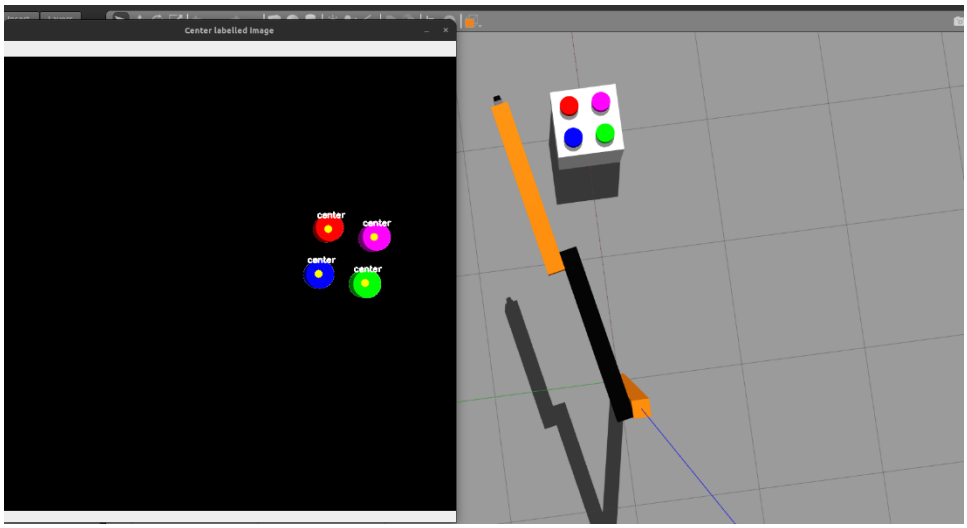
Green Center coordinate: (512, 371)
Blue Center coordinate: (426, 371)
Meg Center coordinate: (510, 288)
Red Center coordinate: (427, 289)
Trajectory start position:



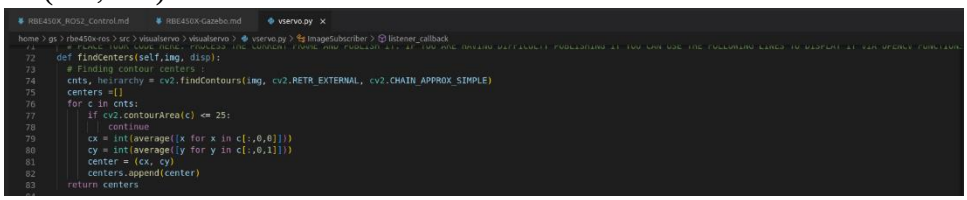
Green Center coordinate: (312, 376)
Blue Center coordinate: (234, 400)
Meg Center coordinate: (289, 299)
Red Center coordinate: (210, 323)

Gokul Srinivasan 500594698

Step 3: Trajectory end position

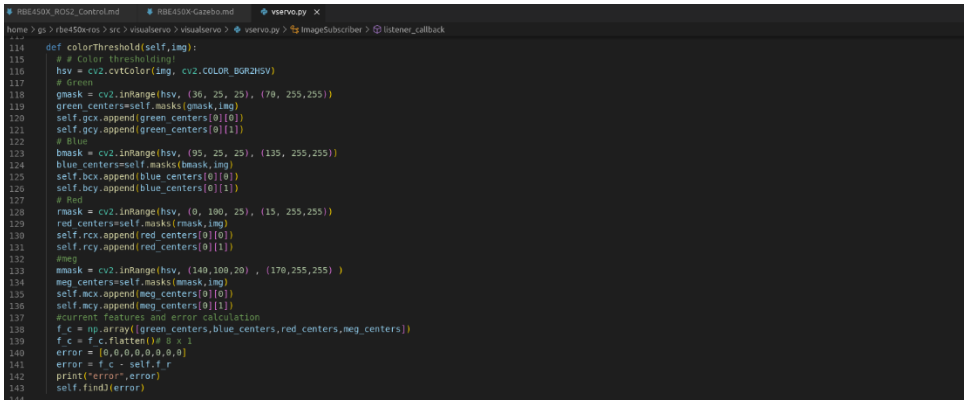


Green Center coordinate: (639, 398)
Blue Center coordinate: (557, 382)
Meg Center coordinate: (655, 317)
Red Center coordinate: (574, 303)



```
def findCenters(self,img, disp):
    # Finding contour centers
    cnts, heirarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    centers =[]
    for c in cnts:
        if cv2.contourArea(c) <= 25:
            continue
        cx = int(average([x for x in c[:,0,0]]))
        cy = int(average([y for y in c[:,0,1]]))
        center = (cx, cy)
        centers.append(center)
    return centers
```

```
def masks(self,mask, img):
    imask = mask>0
    color = np.zeros_like(img, np.uint8)
    color[imask] = img[imask]
    return self.findCenters(mask, color)
```

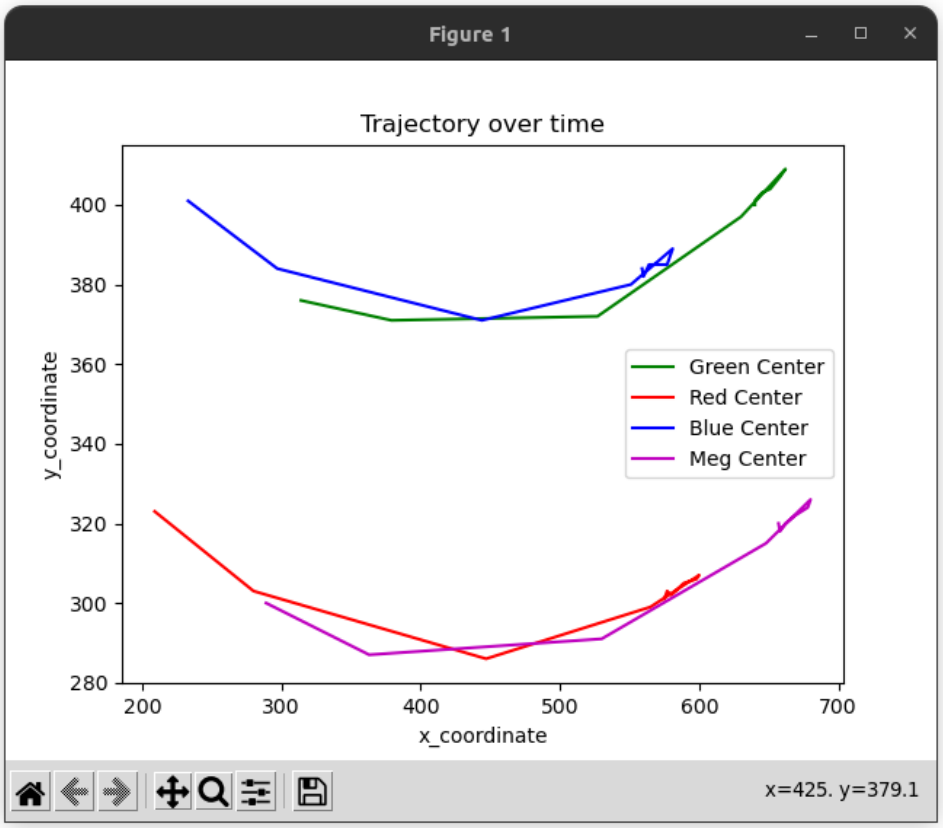


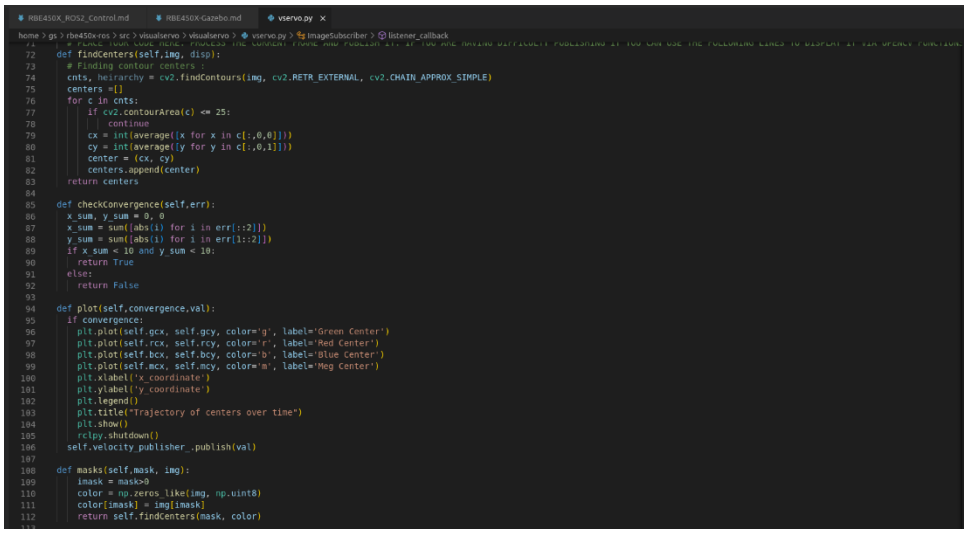
```
def colorThreshold(self,img):
    # # Color thresholding
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # Green
    gmask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))
    green_centers=self.masks(gmask,img)
    self.gcx.append(green_centers[0][0])
    self.gcy.append(green_centers[0][1])
    # Blue
    bmask = cv2.inRange(hsv, (95, 25, 25), (135, 255,255))
    blue_centers=self.masks(bmask,img)
    self.bcx.append(blue_centers[0][0])
    self.bcy.append(blue_centers[0][1])
    # Red
    rmask = cv2.inRange(hsv, (0, 100, 25), (15, 255,255))
    red_centers=self.masks(rmask,img)
    self.rcx.append(red_centers[0][0])
    self.rcy.append(red_centers[0][1])
    #meg
    mmask = cv2.inRange(hsv, (140,100,20) , (170,255,255) )
    meg_centers=self.masks(mmask,img)
    self.mcx.append(meg_centers[0][0])
    self.mcy.append(meg_centers[0][1])
    #current features and error calculation
    f_c = np.array([green_centers,blue_centers,red_centers,meg_centers])
    f_c = f_c.flatten()# 8 x 1
    error = [0,0,0,0,0,0,0,0]
    error = f_c - self.f_r
    print("error",error)
    self.findJ(error)
```

Gokul Srinivasan 500594698

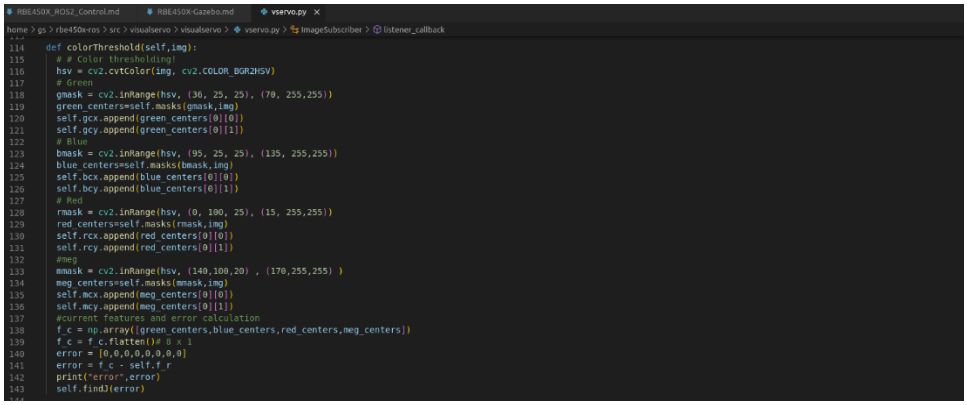
Step 4:

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

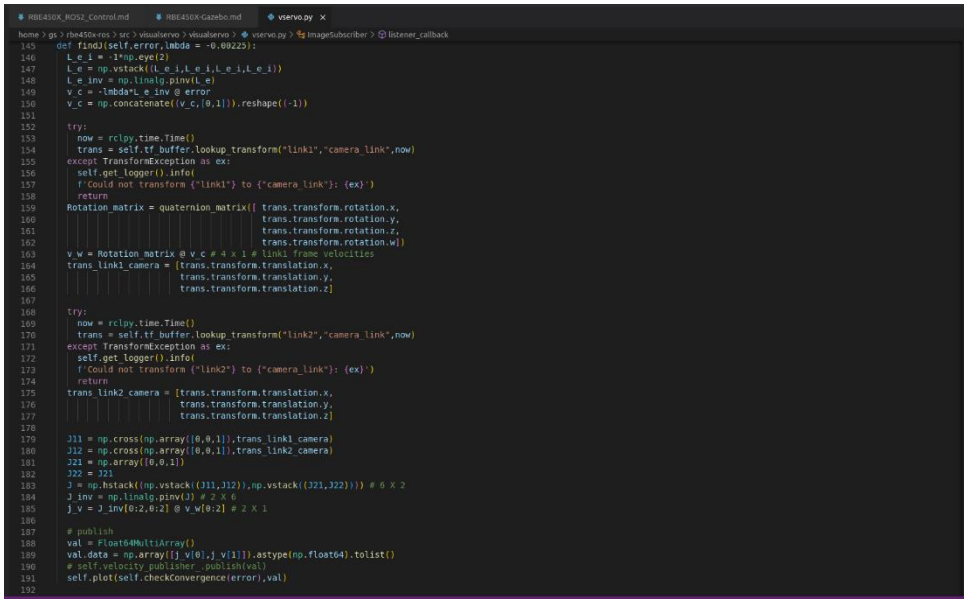




```
  def findCenters(self,img, disp):
    # Finding contour centers :
    cnts, heirarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    centers =[]
    for c in cnts:
        if cv2.contourArea(c) <= 25:
            continue
        cx = int(average([x for x in c[:,0,0]]))
        cy = int(average([y for y in c[:,0,1]]))
        center = (cx, cy)
        centers.append(center)
    return centers

  def checkConvergence(self,err):
    x_sum, y_sum = 0, 0
    x_sum = sum([abs(i) for i in err[::2]])
    y_sum = sum([abs(i) for i in err[1::2]])
    if x_sum < 10 and y_sum < 10:
      return True
    else:
      return False

  def plot(self,convergence,val):
    if convergence:
      plt.plot(self.gcx, self.gcy, color='g', label='Green Center')
      plt.plot(self.rcx, self.rcy, color='r', label='Red Center')
      plt.plot(self.bcx, self.bcy, color='b', label='Blue Center')
      plt.plot(self.mcx, self.mcy, color='m', label='Meg Center')
      plt.xlabel('x_coordinate')
      plt.ylabel('y_coordinate')
      plt.legend()
      plt.title("Trajectory of centers over time")
      plt.show()
      rclpy.shutdown()
    self.velocity_publisher_.publish(val)

  def masks(self,mask, img):
      imask = mask>0
      color = np.zeros_like(img, np.uint8)
      color[imask] = img[imask]
      return self.findCenters(mask, color)
```



```
def colorThreshold(self,img):
  # # Color thresholding!
  hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
  # Green
  gmask = cv2.inRange(hsv, (36, 25, 25), (70, 255,255))
  green_centers=self.masks(gmask,img)
  self.gcx.append(green_centers[0][0])
  self.gcy.append(green_centers[0][1])
  # Blue
  bmask = cv2.inRange(hsv, (95, 25, 25), (135, 255,255))
  blue_centers=self.masks(bmask,img)
  self.bcx.append(blue_centers[0][0])
  self.bcy.append(blue_centers[0][1])
  # Red
  rmask = cv2.inRange(hsv, (0, 100, 25), (15, 255,255))
  red_centers=self.masks(rmask,img)
  self.rcx.append(red_centers[0][0])
  self.rcy.append(red_centers[0][1])
  #meg
  mmask = cv2.inRange(hsv, (140,100,20) , (170,255,255) )
  meg_centers=self.masks(mmask,img)
  self.mcx.append(meg_centers[0][0])
  self.mcy.append(meg_centers[0][1])
  #current features and error calculation
  f_c = np.array([green_centers,blue_centers,red_centers,meg_centers])
  f_c = f_c.flatten()# 8 x 1
  error = [0,0,0,0,0,0,0,0]
  error = f_c - self.f_r
  print("error",error)
  self.findJ(error)
```



```
def findJ(self,error,lmbda = -0.00225):
  L_e_i = -1*np.eye(2)
  L_e = np.vstack((L_e_i,L_e_i,L_e_i,L_e_i))
  L_e_inv = np.linalg.pinv(L_e)
  v_c = -lmbda*L_e_inv @ error
  v_c = np.concatenate((v_c,[0,1])).reshape((-1))

  try:
    now = rclpy.time.Time()
    trans = self.tf_buffer.lookup_transform("link1","camera_link",now)
  except TransformException as ex:
    self.get_logger().info(
    f'Could not transform {"link1"} to {"camera_link"}: {ex}')
    return
  Rotation_matrix = quaternion_matrix([ trans.transform.rotation.x,
                                        trans.transform.rotation.y,
                                        trans.transform.rotation.z,
                                        trans.transform.rotation.w])
  v_w = Rotation_matrix @ v_c # 4 x 1 # link1 frame velocities
  trans_link1_camera = [trans.transform.translation.x,
                        trans.transform.translation.y,
                        trans.transform.translation.z]

  try:
    now = rclpy.time.Time()
    trans = self.tf_buffer.lookup_transform("link2","camera_link",now)
  except TransformException as ex:
    self.get_logger().info(
    f'Could not transform {"link2"} to {"camera_link"}: {ex}')
    return
  trans_link2_camera = [trans.transform.translation.x,
                        trans.transform.translation.y,
                        trans.transform.translation.z]

  J11 = np.cross(np.array([0,0,1]),trans_link1_camera)
  J12 = np.cross(np.array([0,0,1]),trans_link2_camera)
  J21 = np.array([0,0,1])
  J22 = J21
  J = np.hstack((np.vstack((J11,J12)),np.vstack((J21,J22)))) # 6 X 2
  J_inv = np.linalg.pinv(J) # 2 X 6
  j_v = J_inv[0:2,0:2] @ v_w[0:2] # 2 X 1

  # publish
  val = Float64MultiArray()
  val.data = np.array([j_v[0],j_v[1]]).astype(np.float64).tolist()
  # self.velocity_publisher_.publish(val)
  self.plot(self.checkConvergence(error),val)
```