Homework 4 - Report
Bharath Kumar Ramesh Babu

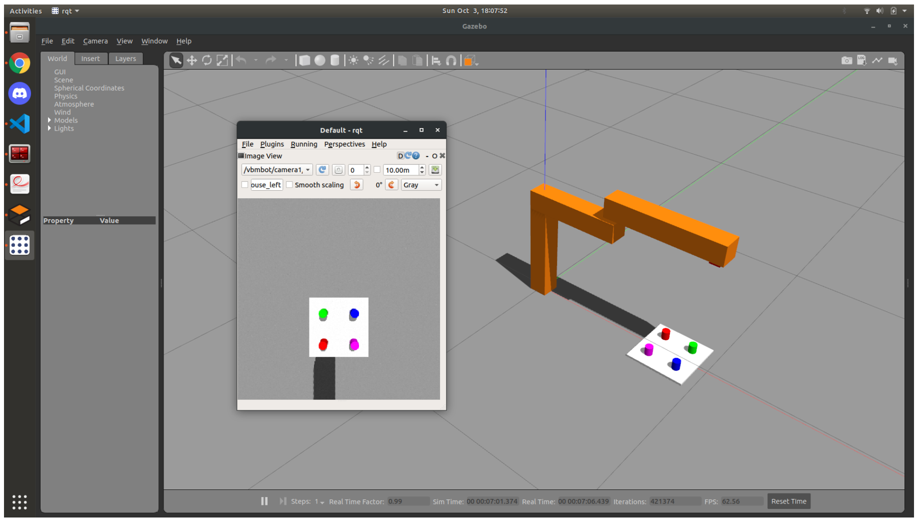
Problem Statement:
Implement a visual servoing algorithm for a 2 DOF robot.

1. Spawn the object you have created in the previous HW on the ground within the robot's workspace.
2. Move the robot via the position controller so that the whole object is visible in the image. Take an image, get the coordinates of the 4 circle centers.
3. Move the robot to a different location using the position controller. In the new location, the whole object should still be visible by the virtual camera. Take an image, get the coordinates of the 4 circle centers.
4. Implement a visual servoing algorithm that uses these four point features (the centers of the circles) and servos the robot from one image configuration to the other.

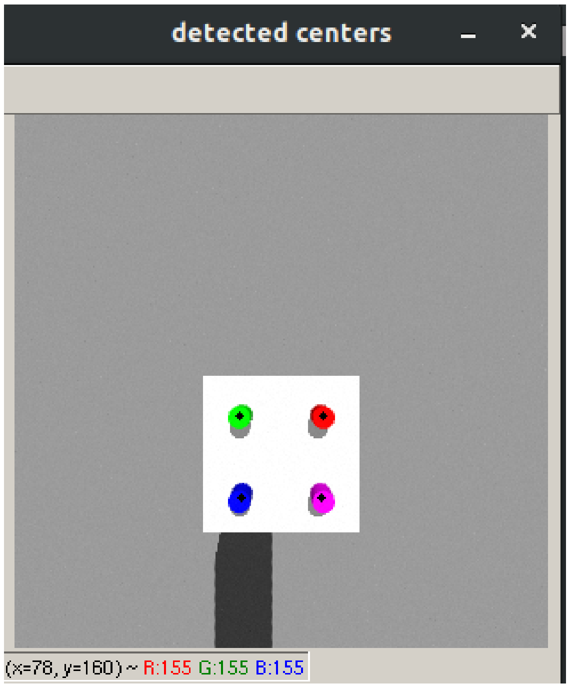
List of files I created for this assignment:
1. scripts/hw4_solution.py - This file contains the code for object center detection, and visual servoing logic
2. scripts/switch_controllers.py - Running this file with an argument ('position' or 'velocity') will switch the controller mentioned in the argument
3. urdf/robot_env.urdf - The urdf of the custom spawned object
4. bags/feat1-2.bag - Contains the record of object center data when feature 2 (as mentioned in the code) is given as setpoint to the controller
5. bags/feat2-1.bag - Contains the record of object center data when feature 1 (as mentioned in the code) is given as setpoint to the controller

Deliverables:
The code/workspace folder to this homework is submitted along with the report

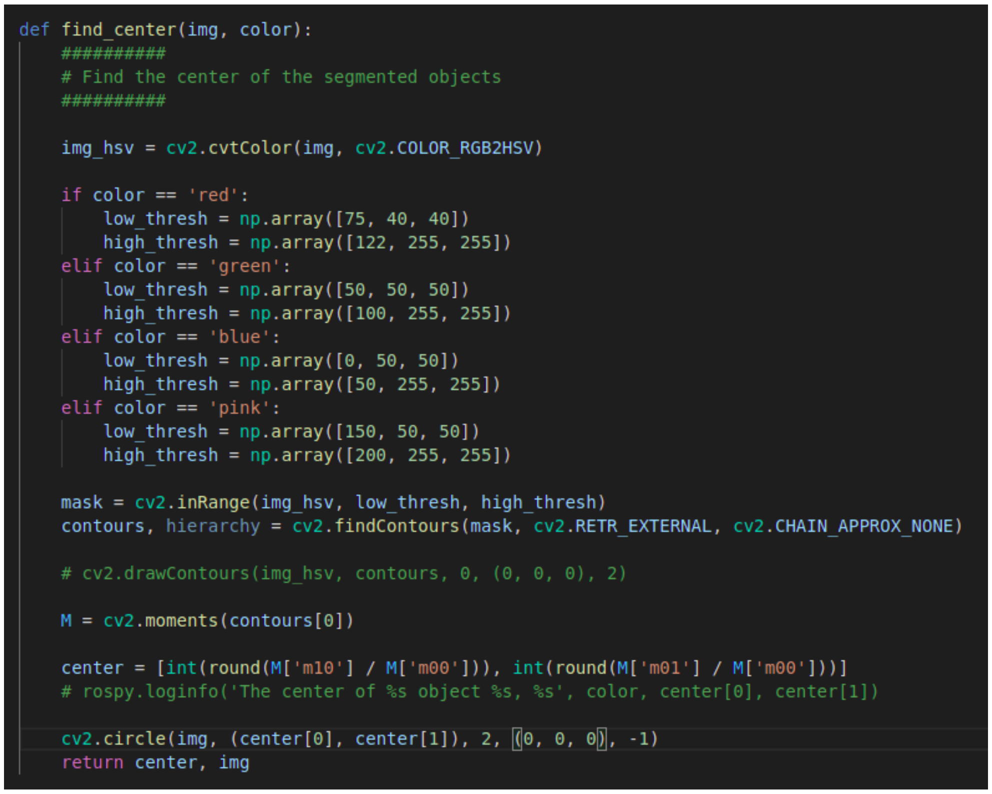
1. Screenshot of the robot and the object in Gazebo



2. Image taken from the virtual camera, the detected circle centers, and a snapshot of the related part of the code

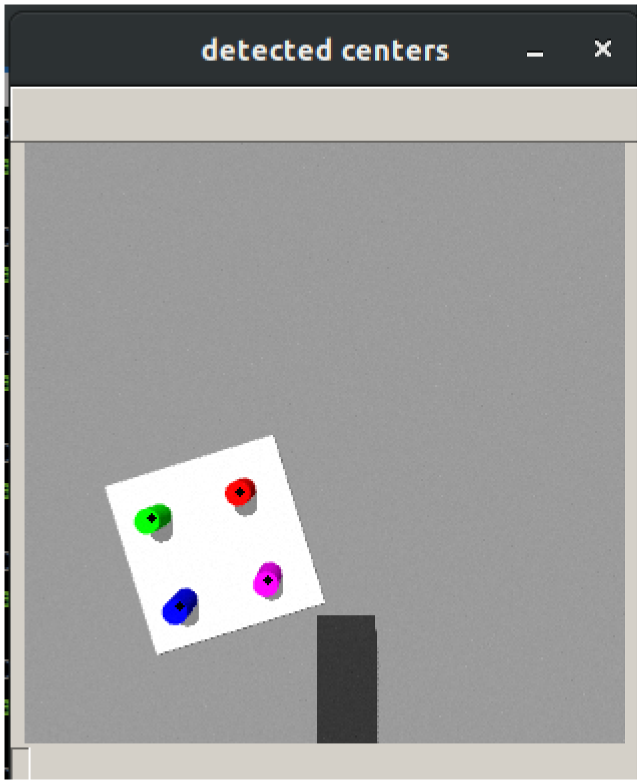


The image is converted to hsv initially. The threshold values for the different colored objects are found. A mask is created for each object and contours of the masks are found. The center of the objects are found using the contour moments. This proved to be significantly faster than the averaging approach in the previous homework. The code snippet for the logic is as follows.

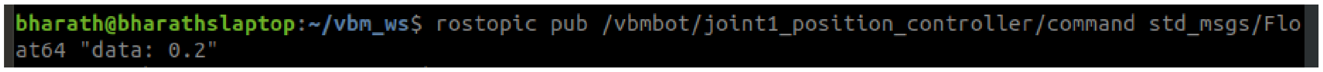


```
def find_center(img, color):
    ##########
    # Find the center of the segmented objects
    ##########

    img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

    if color == 'red':
        low_thresh = np.array([75, 40, 40])
        high_thresh = np.array([122, 255, 255])
    elif color == 'green':
        low_thresh = np.array([50, 50, 50])
        high_thresh = np.array([100, 255, 255])
    elif color == 'blue':
        low_thresh = np.array([0, 50, 50])
        high_thresh = np.array([50, 255, 255])
    elif color == 'pink':
        low_thresh = np.array([150, 50, 50])
        high_thresh = np.array([200, 255, 255])

    mask = cv2.inRange(img_hsv, low_thresh, high_thresh)
    contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    # cv2.drawContours(img_hsv, contours, 0, (0, 0, 0), 2)

    M = cv2.moments(contours[0])

    center = [int(round(M['m10'] / M['m00'])), int(round(M['m01'] / M['m00']))]
    # rospy.loginfo('The center of %s object %s, %s', color, center[0], center[1])

    cv2.circle(img, (center[0], center[1]), 2, (0, 0, 0), -1)
    return center, img
```

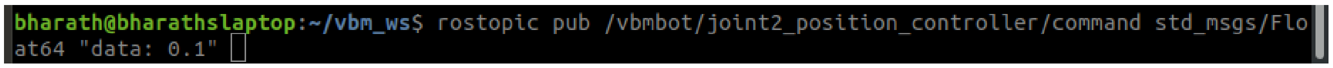
3. Image taken from the virtual camera, the detected circle centers, and a snapshot of the related part of the code



The joint 1 is set to 0.2 radians and the joint 2 is set to 0.2 radians using the joint position controller. The position controller publisher command is given below. The black dots depict the center of each object.

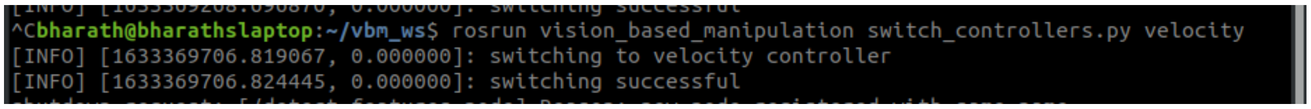


```
bharath@bharathslaptop:~/vbm_ws$ rostopic pub /vbmbot/joint1_position_controller/command std_msgs/Flo
at64 "data: 0.2"
```



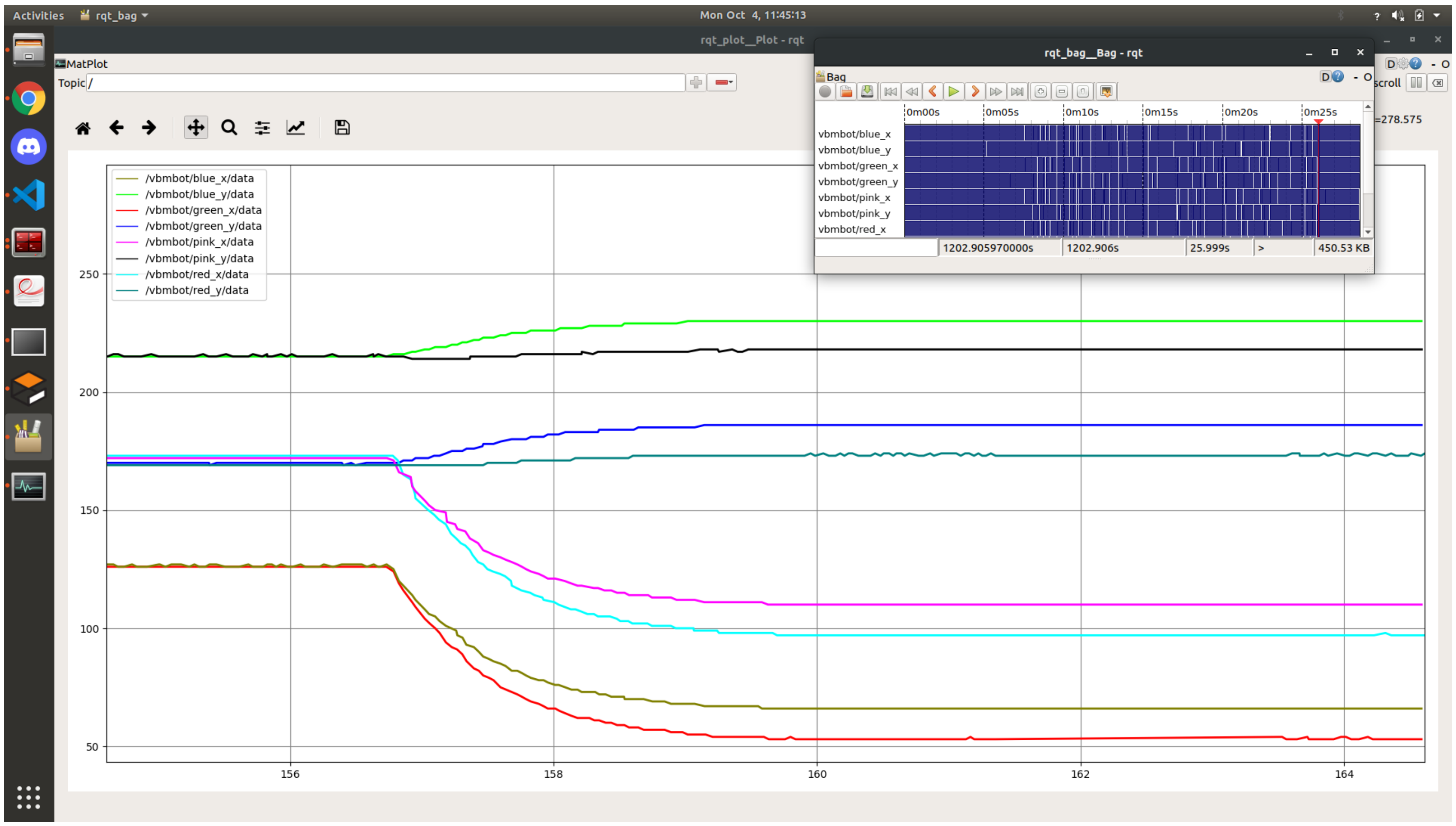
```
bharath@bharathslaptop:~/vbm_ws$ rostopic pub /vbmbot/joint2_position_controller/command std_msgs/Flo
at64 "data: 0.1"
```

4. Visual servoing algorithm that uses these four point features (the centers of the circles) and servos the robot from one image configuration to the other.

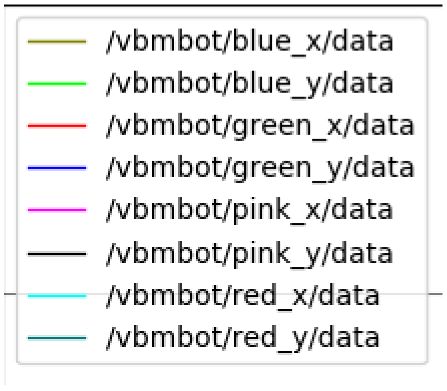


```
^Cbharath@bharathslaptop:~/vbm_ws$ rosrun vision_based_manipulation switch_controllers.py velocity
[INFO] [1633369706.819067, 0.000000]: switching to velocity controller
[INFO] [1633369706.824445, 0.000000]: switching successful
```

The controller is changed from joint position controller to joint velocity controller using the following command

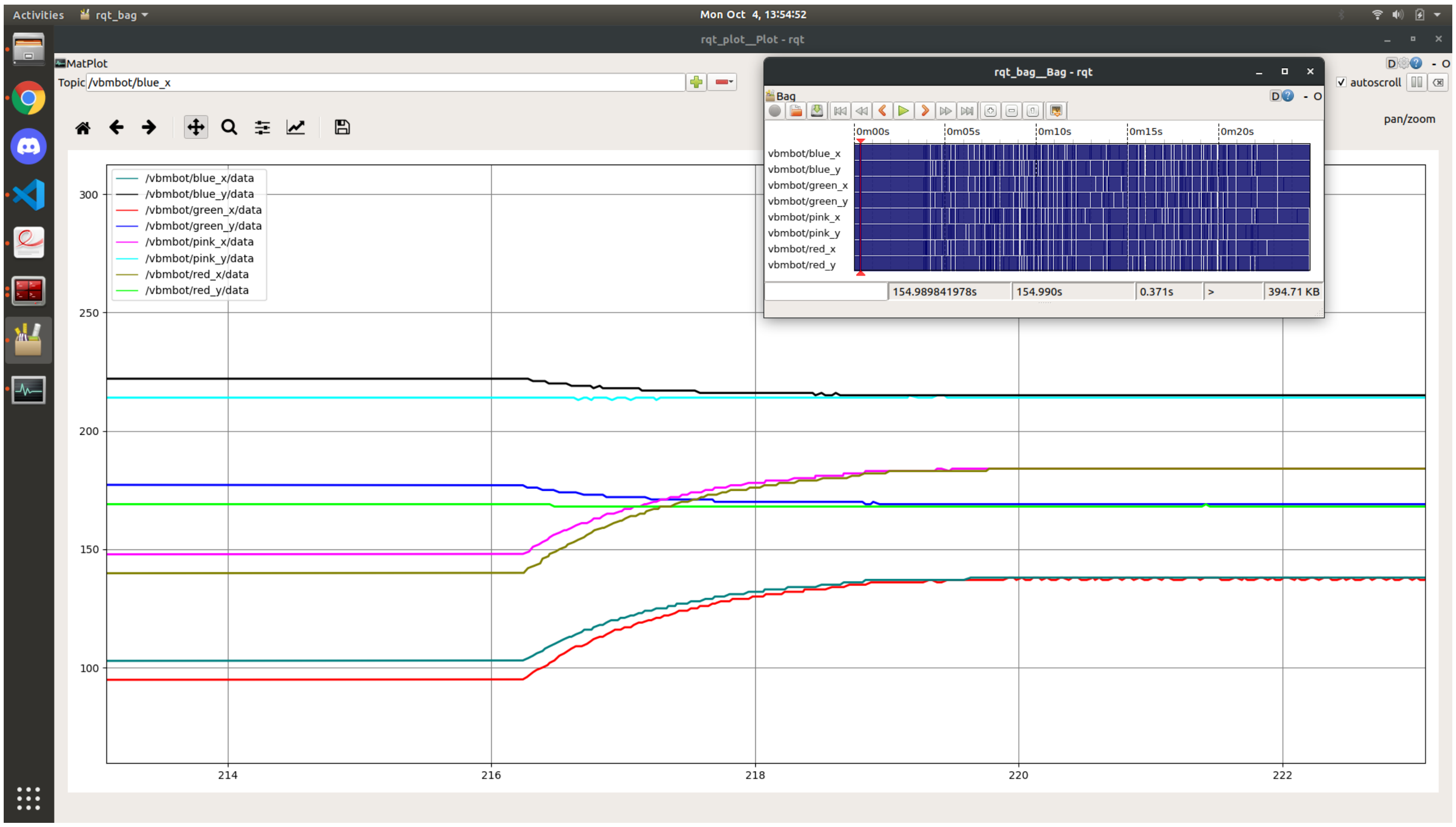


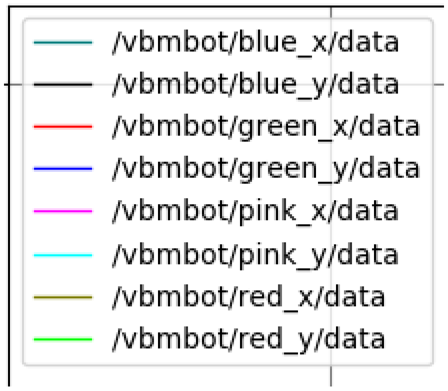
The above graph is the response of the object centers for the setpoint [[63, 187], [121, 218], [107, 174], [77, 231]]. They are [green, pink, red, blue] respectively.



The controller receives the current object centers and the setpoint object centers. It computes the error. An image jacobian is calculated with f and z as 1. The error is multiplied with the pseudo inverse of the image jacobian and in turn is multiplied by a control gain. The negative of the controller output is multiplied with the pseudo inverse of the robot jacobian to get the desired joint velocities.

Once the setpoints are given, the controller controls the joint velocities to make them reach the checkpoint exponentially. A control gain of 0.01 is used for the controller. The convergence takes 2-4 seconds. Higher control gain will result in faster convergence.



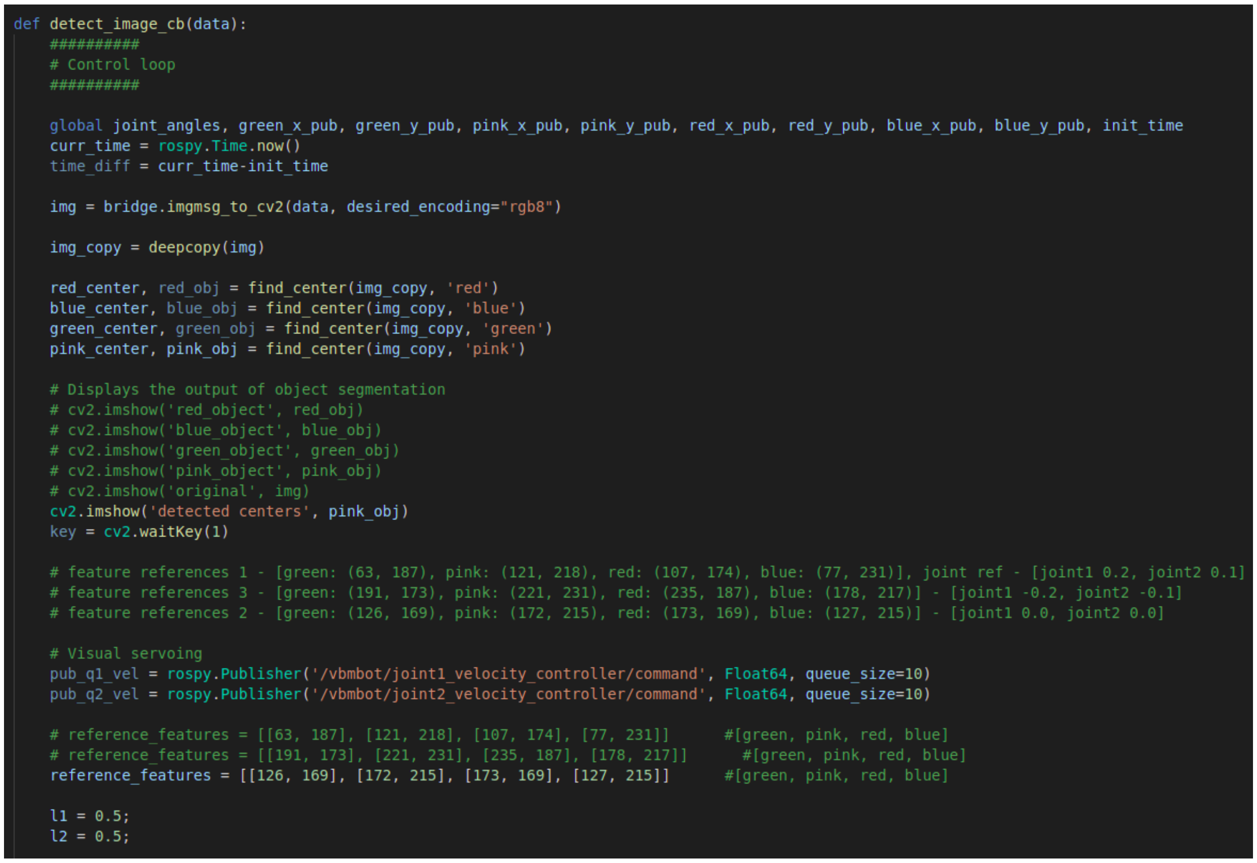


The above graph is the response of the object centers for the setpoint [[126, 169], [172, 215], [173, 169], [127, 215]].
They are [green, pink, red, blue] respectively.

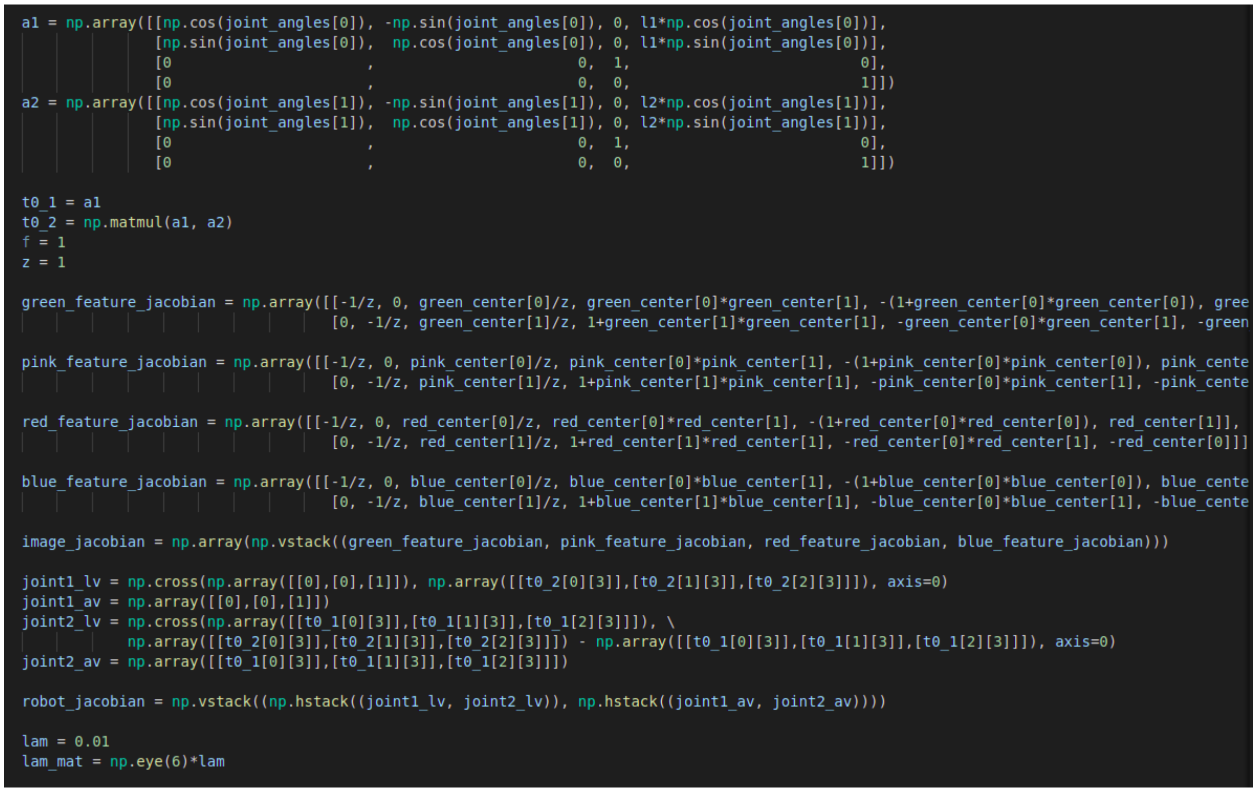
The data that correspond to the graph line colors are shown at the right.

The center values are recorded in a bag. The bags can be found in the bags folder inside the package. The plot is visualized using rqt_plot.

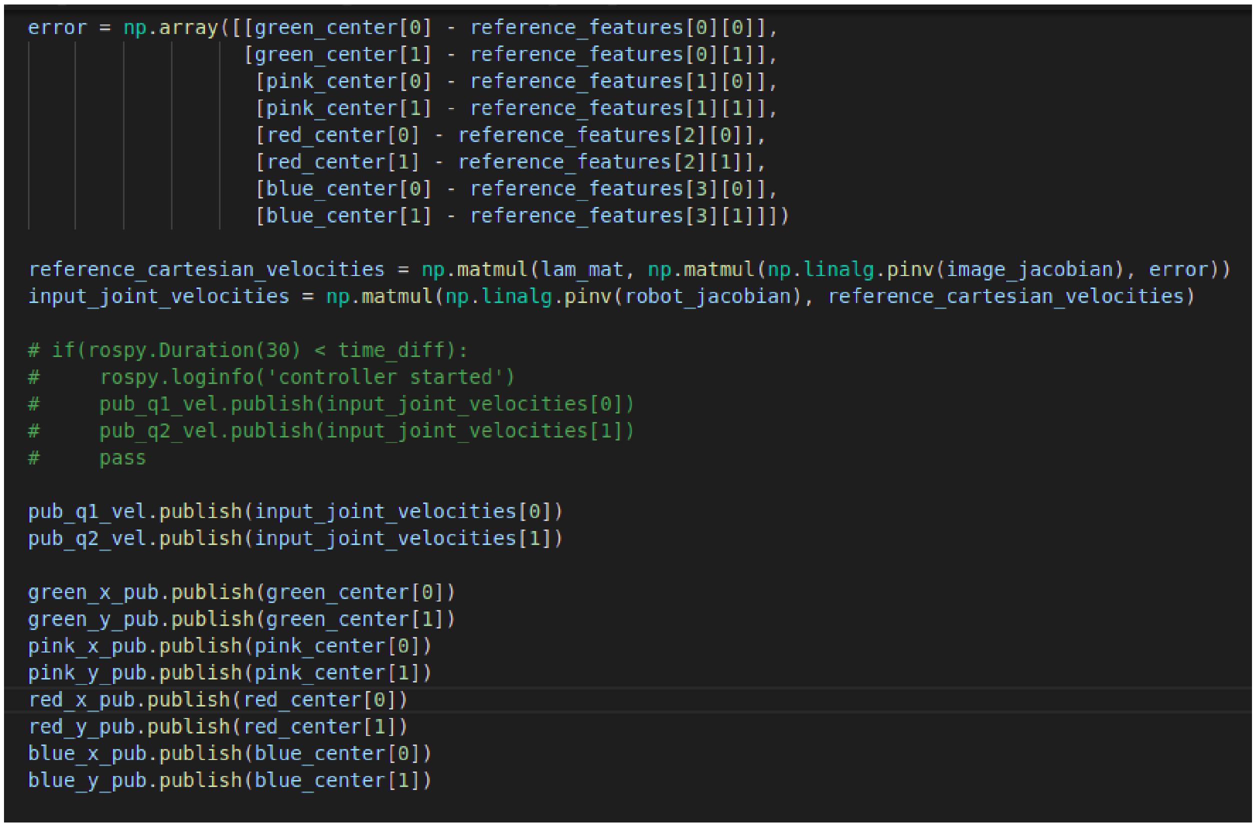
The code for the controller is as below



```python
def detect_image_cb(data):
    ##########
    # Control loop
    ##########

    global joint_angles, green_x_pub, green_y_pub, pink_x_pub, pink_y_pub, red_x_pub, red_y_pub, blue_x_pub, blue_y_pub, init_time
    curr_time = rospy.Time.now()
    time_diff = curr_time-init_time

    img = bridge.imgmsg_to_cv2(data, desired_encoding="rgb8")

    img_copy = deepcopy(img)

    red_center, red_obj = find_center(img_copy, 'red')
    blue_center, blue_obj = find_center(img_copy, 'blue')
    green_center, green_obj = find_center(img_copy, 'green')
    pink_center, pink_obj = find_center(img_copy, 'pink')

    # Displays the output of object segmentation
    # cv2.imshow('red_object', red_obj)
    # cv2.imshow('blue_object', blue_obj)
    # cv2.imshow('green_object', green_obj)
    # cv2.imshow('pink_object', pink_obj)
    # cv2.imshow('original', img)
    cv2.imshow('detected centers', pink_obj)
    key = cv2.waitKey(1)

    # feature references 1 - [green: (63, 187), pink: (121, 218), red: (107, 174), blue: (77, 231)], joint ref - [joint1 0.2, joint2 0.1]
    # feature references 3 - [green: (191, 173), pink: (221, 231), red: (235, 187), blue: (178, 217)] - [joint1 -0.2, joint2 -0.1]
    # feature references 2 - [green: (126, 169), pink: (172, 215), red: (173, 169), blue: (127, 215)] - [joint1 0.0, joint2 0.0]

    # Visual servoing
    pub_q1_vel = rospy.Publisher('/vbmbot/joint1_velocity_controller/command', Float64, queue_size=10)
    pub_q2_vel = rospy.Publisher('/vbmbot/joint2_velocity_controller/command', Float64, queue_size=10)

    # reference_features = [[63, 187], [121, 218], [107, 174], [77, 231]]      #[green, pink, red, blue]
    # reference_features = [[191, 173], [221, 231], [235, 187], [178, 217]]      #[green, pink, red, blue]
    reference_features = [[126, 169], [172, 215], [173, 169], [127, 215]]      #[green, pink, red, blue]

    l1 = 0.5;
    l2 = 0.5;
```



```python
a1 = np.array([[np.cos(joint_angles[0]), -np.sin(joint_angles[0]), 0, l1*np.cos(joint_angles[0])],
               [np.sin(joint_angles[0]),  np.cos(joint_angles[0]), 0, l1*np.sin(joint_angles[0])],
               [0                       ,                        0,  1,                          0],
               [0                       ,                        0,  0,                          1]])
a2 = np.array([[np.cos(joint_angles[1]), -np.sin(joint_angles[1]), 0, l2*np.cos(joint_angles[1])],
               [np.sin(joint_angles[1]),  np.cos(joint_angles[1]), 0, l2*np.sin(joint_angles[1])],
               [0                       ,                        0,  1,                          0],
               [0                       ,                        0,  0,                          1]])

t0_1 = a1
t0_2 = np.matmul(a1, a2)
f = 1
z = 1

green_feature_jacobian = np.array([[-1/z, 0, green_center[0]/z, green_center[0]*green_center[1], -(1+green_center[0]*green_center[0]), gree
                                  [0, -1/z, green_center[1]/z, 1+green_center[1]*green_center[1], -green_center[0]*green_center[1], -green

pink_feature_jacobian = np.array([[-1/z, 0, pink_center[0]/z, pink_center[0]*pink_center[1], -(1+pink_center[0]*pink_center[0]), pink_cente
                                  [0, -1/z, pink_center[1]/z, 1+pink_center[1]*pink_center[1], -pink_center[0]*pink_center[1], -pink_cente

red_feature_jacobian = np.array([[-1/z, 0, red_center[0]/z, red_center[0]*red_center[1], -(1+red_center[0]*red_center[0]), red_center[1]],
                                  [0, -1/z, red_center[1]/z, 1+red_center[1]*red_center[1], -red_center[0]*red_center[1], -red_center[0]]]

blue_feature_jacobian = np.array([[-1/z, 0, blue_center[0]/z, blue_center[0]*blue_center[1], -(1+blue_center[0]*blue_center[0]), blue_cente
                                  [0, -1/z, blue_center[1]/z, 1+blue_center[1]*blue_center[1], -blue_center[0]*blue_center[1], -blue_cente

image_jacobian = np.array(np.vstack((green_feature_jacobian, pink_feature_jacobian, red_feature_jacobian, blue_feature_jacobian)))

joint1_lv = np.cross(np.array([[0],[0],[1]]), np.array([[t0_2[0][3]],[t0_2[1][3]],[t0_2[2][3]]]), axis=0)
joint1_av = np.array([[0],[0],[1]])
joint2_lv = np.cross(np.array([[t0_1[0][3]],[t0_1[1][3]],[t0_1[2][3]]]), \
            np.array([[t0_2[0][3]],[t0_2[1][3]],[t0_2[2][3]]]) - np.array([[t0_1[0][3]],[t0_1[1][3]],[t0_1[2][3]]]), axis=0)
joint2_av = np.array([[t0_1[0][3]],[t0_1[1][3]],[t0_1[2][3]]])

robot_jacobian = np.vstack((np.hstack((joint1_lv, joint2_lv)), np.hstack((joint1_av, joint2_av))))

lam = 0.01
lam_mat = np.eye(6)*lam
```



```python
error = np.array([[green_center[0] - reference_features[0][0]],
                  [green_center[1] - reference_features[0][1]],
                   [pink_center[0] - reference_features[1][0]],
                   [pink_center[1] - reference_features[1][1]],
                   [red_center[0] - reference_features[2][0]],
                   [red_center[1] - reference_features[2][1]],
                   [blue_center[0] - reference_features[3][0]],
                   [blue_center[1] - reference_features[3][1]]])

reference_cartesian_velocities = np.matmul(lam_mat, np.matmul(np.linalg.pinv(image_jacobian), error))
input_joint_velocities = np.matmul(np.linalg.pinv(robot_jacobian), reference_cartesian_velocities)

# if(rospy.Duration(30) < time_diff):
#     rospy.loginfo('controller started')
#     pub_q1_vel.publish(input_joint_velocities[0])
#     pub_q2_vel.publish(input_joint_velocities[1])
#     pass

pub_q1_vel.publish(input_joint_velocities[0])
pub_q2_vel.publish(input_joint_velocities[1])

green_x_pub.publish(green_center[0])
green_y_pub.publish(green_center[1])
pink_x_pub.publish(pink_center[0])
pink_y_pub.publish(pink_center[1])
red_x_pub.publish(red_center[0])
red_y_pub.publish(red_center[1])
blue_x_pub.publish(blue_center[0])
blue_y_pub.publish(blue_center[1])
```