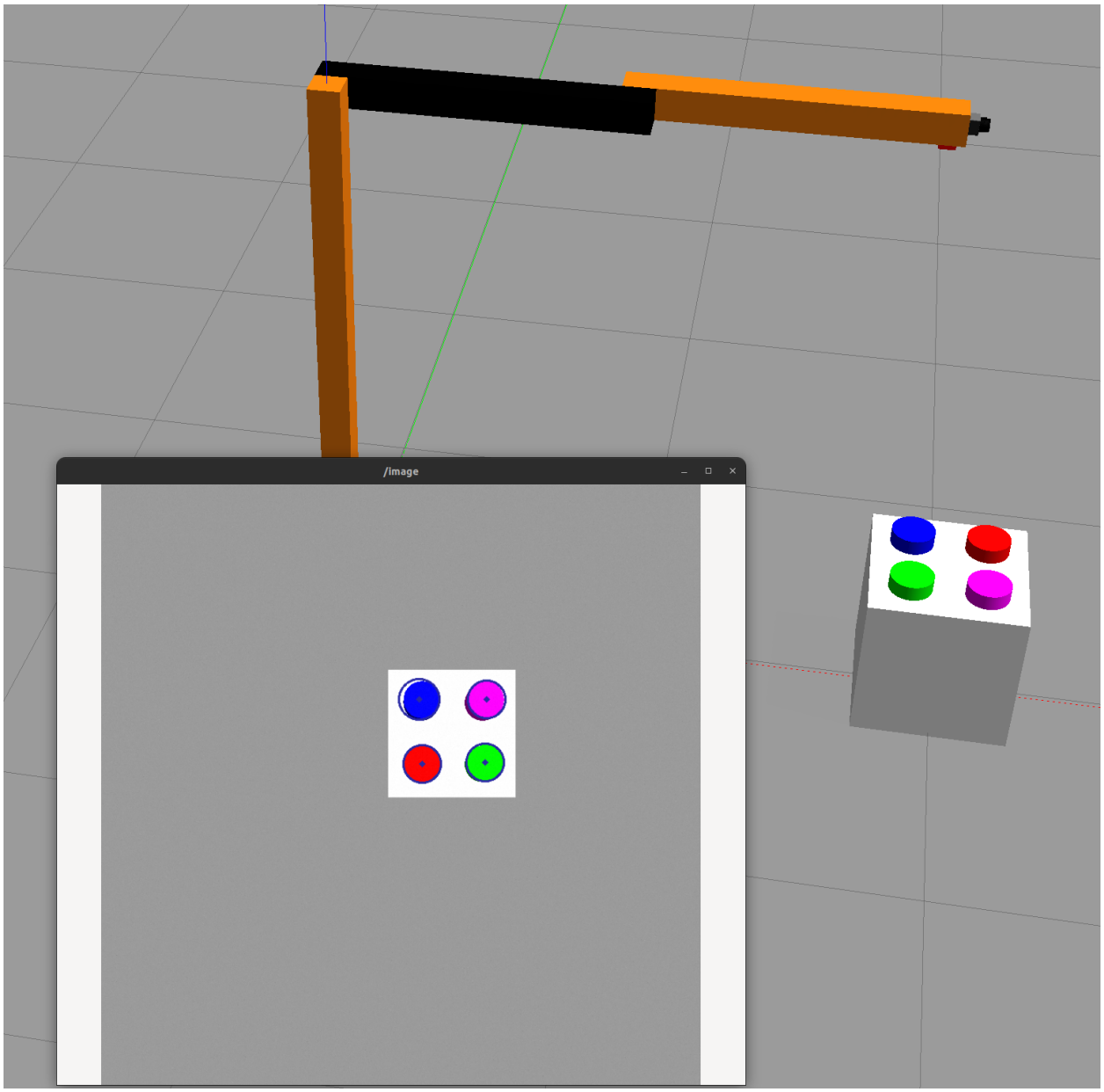
Step 1

Center’s from reference location
Blue: 287.9 427.4
Green: 371.6 511.0
Purple: 288.0 511.0
Red: 371.0 427.7

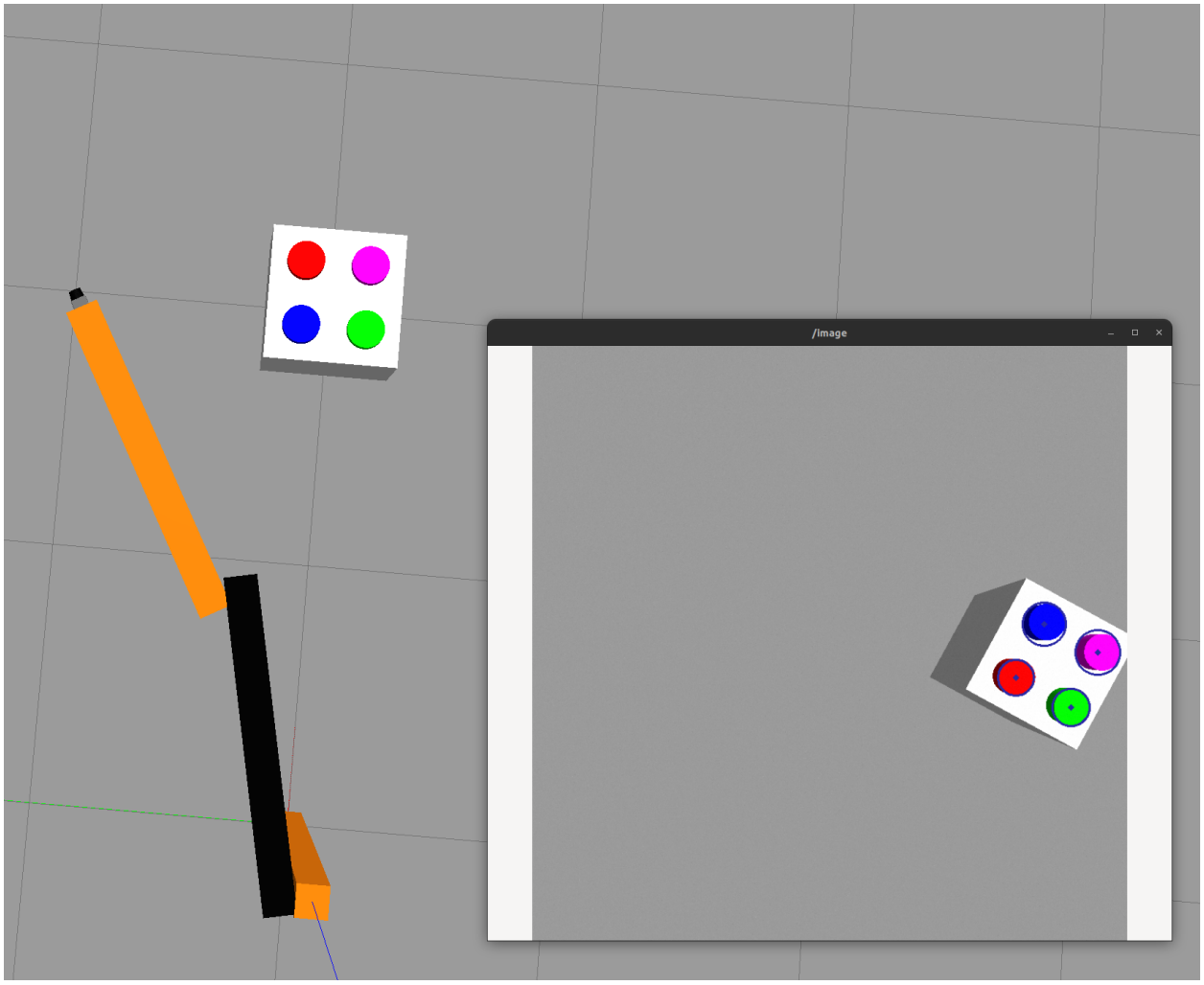
Below is the figure for the detected center’s which is used as reference



Step 3

Center’s from new location
Blue: 371.3 687.7
Green: 484.6 720.9
Purple: 411.4 760.7
Red: 444.6 647.7

Below is the figure of the perturbed manipulator position



Below is the code for step1 and step3 the same

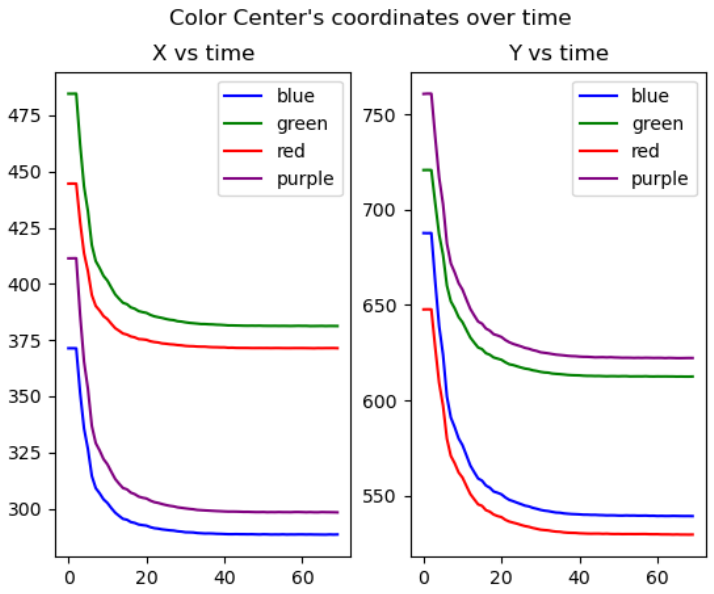
```python
def get_center_of_mask(mask):
    x_i = 0;
    y_i = 0;
    count = 0;
    coords = np.argwhere(mask)
    count = coords.shape[0]
    coords = np.sum(coords,axis=0)
    if count > 0:
        coords = coords/count
        return coords

    return (0,0)

class VisualServoController(Node):
    def __init__(self):
        super().__init__('visual_servo_controller')
```

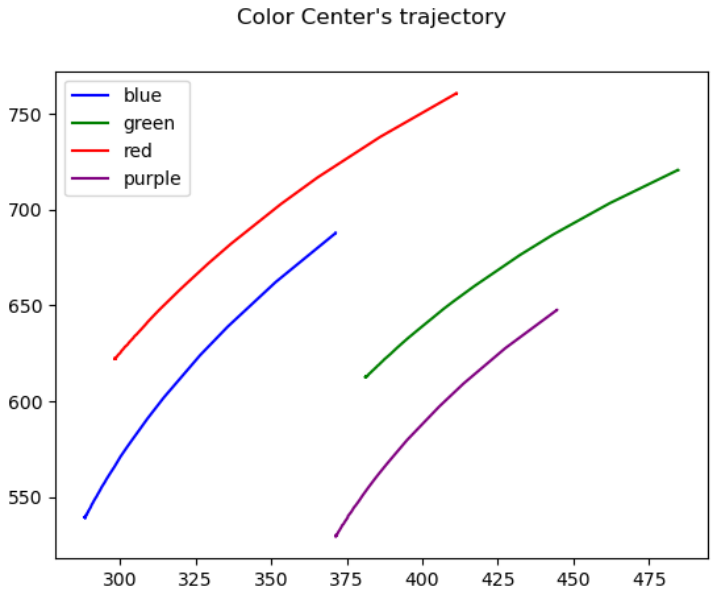
src/visual_servo_controller/visual_servo_controller/visual_servo.py

```python
    def calculate_image_feature_centers(self,current_frame):
        hsv = cv2.cvtColor(current_frame,cv2.COLOR_BGR2HSV)
        blue_mask = cv2.inRange(hsv,self.blue_low_,self.blue_high_)
        red_mask = cv2.inRange(hsv,self.red_low_,self.red_high_)
        green_mask = cv2.inRange(hsv,self.green_low_,self.green_high_)
        pink_mask = cv2.inRange(hsv,self.pink_low_,self.pink_high_)

        final_mask = blue_mask + red_mask + green_mask + pink_mask

        x_blue,y_blue =  get_center_of_mask(blue_mask)
        x_green,y_green =  get_center_of_mask(green_mask)
        x_pink,y_pink =  get_center_of_mask(pink_mask)
        x_red,y_red =  get_center_of_mask(red_mask)
        feats_curr = np.array([[x_blue,y_blue],
                               [x_green,y_green],
                               [x_pink,y_pink],
                               [x_red,y_red]],dtype=np.float32)

        return feats_curr,final_mask

        # print(f"blue:{x_blue,y_blue}")
        # print(f"green:{x_green,y_green}")
        # print(f"pink:{x_pink,y_pink}")
        # print(f"red:{x_red,y_red}")
        # blue:(287.82439744220363, 427.43384161337923)
```

Step 4

Below are the plots of X-Y trajectories and how X-Y are changing w.r.t time





Below are the snippets of code along with line numbers taken from `visual_servo.py`

```
root@bhagyasyam: ~/vbm/HW4/ros_ws/src/visual_servo_controller/visual_servo_contr
```

```python
    def calculate_required_joint_velocities(self, feats_ref, feats_curr, lmbda = 0.003):
        """
        input:
            feats_ref - reference feature points 4 x 2
            feats_curr - current feature points 4 x 2
        output:
            joint_velocities - 2 x 1
        """
        # Calculate image inverse jacobian
        L_e_i = np.eye(2) # 2 x 2
        L_e = np.vstack((L_e_i,L_e_i,L_e_i,L_e_i)) # 8 x 2
        L_e_inv = np.linalg.pinv(L_e) # 2 x 8

        # calculate error in feature points
        error = feats_curr - feats_ref # 4 x 2
        error = lmbda*error # TODO confirm these equations
        error = error.reshape((8,1))
        print(f"error:{error.flatten()}")

        # calculate required camera velocities in camera frame
        v_cam = L_e_inv @ error # 2 x 1

        # calculate required camera velocities in joint1 frame
        is_transform,R,T = self.get_transforms_R_and_T("link1","camera_link")
        if not is_transform:
            return None

        v_link1 = R[0:2,0:2] @ v_cam # 2 x 1
        Jaco = self.get_rrbot_jacobian() # 6 x 2
        if Jaco is None:
            return None
        Jaco_inv = np.linalg.pinv(Jaco) # 2 x 6
        joint_vel = Jaco_inv[0:2,0:2] @ v_link1 # 2 x 1
        return joint_vel


```

visual_servo.py

```python
    def listener_callback(self, data):
        # Convert ROS Image message to OpenCV image
        current_frame = self.br.imgmsg_to_cv2(data)

        # get current feature locations
        feats_curr,final_mask = self.calculate_image_feature_centers(current_frame)
        print(f"features:{feats_curr.flatten()}")

        x_ref_blue,y_ref_blue=(288,427)
        x_ref_green,y_ref_green=(371,511)
        x_ref_pink,y_ref_pink=(288,510)
        x_ref_red,y_ref_red=(371,428)

        feats_ref = np.array([[x_ref_blue,y_ref_blue],
                              [x_ref_green,y_ref_green],
                              [x_ref_pink,y_ref_pink],
                              [x_ref_red,y_ref_red]],dtype=np.float32)

        # get the required joint velocities
        joint_velocities = self.calculate_required_joint_velocities(feats_ref,feats_curr)
        if joint_velocities is None:
            return

        with open('/root/vbm/data.csv','a') as dataFile:
            writer = csv.writer(dataFile,delimiter=',')
            writer.writerow(feats_curr.flatten())

        # format and publish the velocities
        velocities = Float64MultiArray()
        velocities.data = [joint_velocities[0,0],joint_velocities[1,0]]
        self.velocity_publisher_.publish(velocities)

        # extract and publish the masked output
        masked_img= cv2.bitwise_and(current_frame, current_frame, mask=final_mask)

        self.publisher_.publish(self.br.cv2_to_imgmsg(masked_img , encoding="bgr8"))

def main(args=None):

    # Initialize the rclpy library
```

```python
    def get_transforms_R_and_T(self, to_frame_rel, from_frame_rel):
        """
        input:
            to_frame_rel - to which transformation has to be calculated
            from_frame_rel - from which transformation has to be calculated
        output:
            T - translation vector (x,y,z) 3 x 1
            R - homogeneous rotation matrix 4 x 4
        """
        try:
            now = rclpy.time.Time()
            trans = self.tf_buffer.lookup_transform(
                        to_frame_rel,
                        from_frame_rel,
                        now)

            x = trans.transform.rotation.x
            y = trans.transform.rotation.y
            z = trans.transform.rotation.z
            w = trans.transform.rotation.w
            R = quaternion_matrix([x,y,z,w])

            T = np.array([trans.transform.translation.x,
                          trans.transform.translation.y,
                          trans.transform.translation.z])

            return True, R, T

        except TransformException as ex:
          self.get_logger().info(
          f'Could not transform {to_frame_rel} to {from_frame_rel}: {ex}')
          return False, None, None

    def get_rrbot_jacobian(self):
        """
        output:
```

visual_servo.py

```python
    def get_rrbot_jacobian(self):
        """
        output:
            RR Bot Jacobian - 6 x 2
        """
        is_transform1, R_link1_to_cam, T_link1_to_cam = self.get_transforms_R_and_T("link1","camera_link")
        is_transform2, R_link1_to_link2, T_link1_to_link2 = self.get_transforms_R_and_T("link1","link2")
        if not is_transform1 or not is_transform2:
            return None
        z_local = np.array([0,0,1]) # 3,
        z_local = skew(z_local) # 3 x 3
        J11 = np.eye(3) @ z_local @ T_link1_to_cam # 3 x 1
        J12 = R_link1_to_link2[0:3,0:3] @ z_local @ (T_link1_to_cam - T_link1_to_link2) # 3 x 1
        J21 = np.eye(3) @ z_local # 3 x 1
        J22 = R_link1_to_link2[0:3,0:3] @ z_local # 3 x 1
        J1  = np.hstack((J11,J12)) # 3 x 2
        J2  = np.hstack((J21,J22)) # 3 x 2
        J   = np.vstack((J1,J2)) # 6 x 2
        return J
```

```
81      def calculate_image_feature_centers(self,current_frame):
82          hsv = cv2.cvtColor(current_frame,cv2.COLOR_BGR2HSV)
83          blue_mask = cv2.inRange(hsv,self.blue_low_,self.blue_high_)
84          red_mask = cv2.inRange(hsv,self.red_low_,self.red_high_)
85          green_mask = cv2.inRange(hsv,self.green_low_,self.green_high_)
86          pink_mask = cv2.inRange(hsv,self.pink_low_,self.pink_high_)
87
88          final_mask = blue_mask + red_mask + green_mask + pink_mask
89
90          x_blue,y_blue =  get_center_of_mask(blue_mask)
91          x_green,y_green =  get_center_of_mask(green_mask)
92          x_pink,y_pink =  get_center_of_mask(pink_mask)
93          x_red,y_red =  get_center_of_mask(red_mask)
94          feats_curr = np.array([[x_blue,y_blue],
95                                 [x_green,y_green],
96                                 [x_pink,y_pink],
97                                 [x_red,y_red]],dtype=np.float32)
98          return feats_curr,final_mask
99
100         # print(f"blue:{x_blue,y_blue}")
101         # print(f"green:{x_green,y_green}")
102         # print(f"pink:{x_pink,y_pink}")
103         # print(f"red:{x_red,y_red}")
104         # blue:(287.82439744220363, 427.43384161337923)
105         # green:(371.5760549558391, 511.1143277723258)
106         # pink:(288.0802088277171, 510.936877076412)
107         # red:(371.3253638253638, 427.65696465696465)
108
```