Azzam Shaikh
Oct. 7, 2024

RBE 595: Vision-based Robotic Manipulation

Homework 5

Dr. Berk Calli

Step 1

Deliverable: Present the image taken from the virtual camera, the detected circle centers, a nd a snapshot of the related part of the code in your report.

Result:

Figure 1 shows the image taken from the virtual camera, the detected circle centers, and their positions.

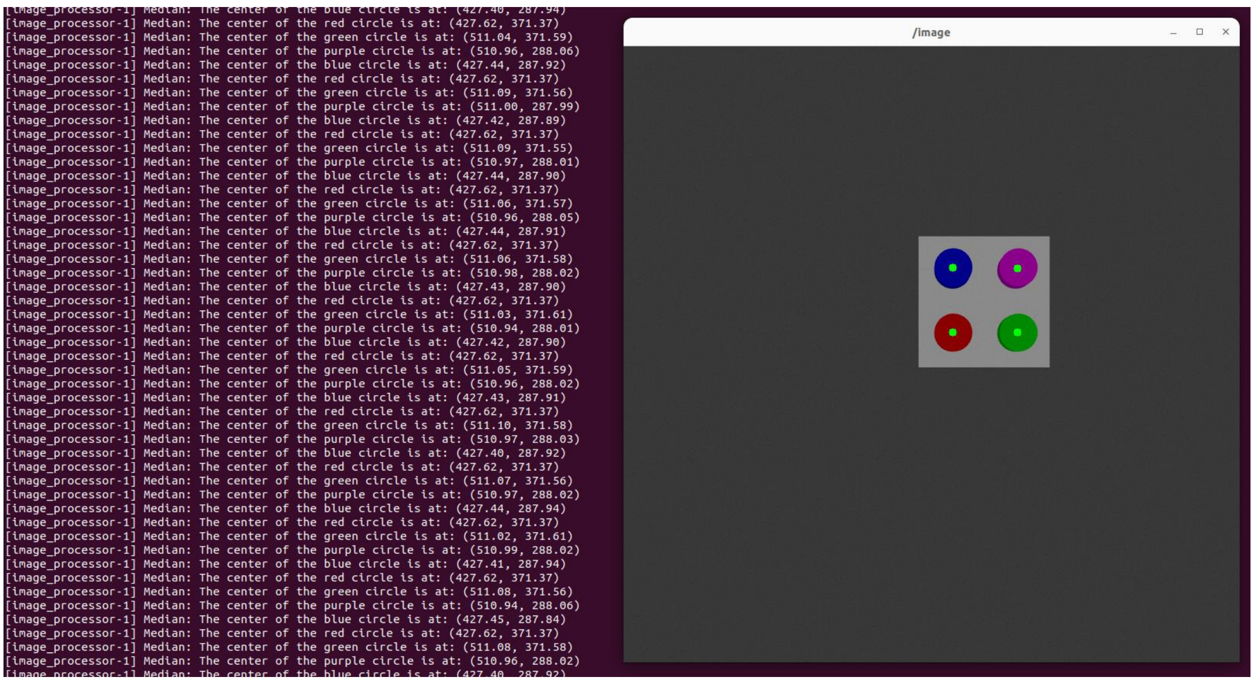

Figure 1: Virtual camera and circle centers and positions

Figure 2 shows the relevant code that implements this functionality. Additional code can be found in /src/opencv_test_py/opencv_test_py/camera.py

Azzam Shaikh
Oct. 7, 2024

```python
@staticmethod
def find_center(image,lower_range,upper_range,name):
    """
    Function to obtain the center of an image given upper and lower HSV ranges
    """
    # Create a mask for the specified color ranges
    mask = cv2.inRange(image, lower_range, upper_range)

    # Locate the x and y coordinates of all the pixels found in the mask
    pixels = np.column_stack(np.where(mask > 0))

    # Calculate the x and y pixel averages to find the center
    center_x = np.mean(pixels[:, 1])
    center_y = np.mean(pixels[:, 0])
    print(f"Median: The center of the {name} circle is at: ({center_x:.2f}, {center_y:.2f})")

    # Return the center
    return (center_x, center_y)

def find_center_of_colors(self, image):
    """
    Function to find the center of all four cirlces in the image
    """

    # Convert the image to HSV colorspace
    hsv_image=cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Call the find_center method given an HSV converted image, and the predefined bounds
    red_center = self.find_center(hsv_image,self.lower_red,self.upper_red,'red')

    green_center = self.find_center(hsv_image,self.lower_green,self.upper_green,'green')

    purple_center = self.find_center(hsv_image,self.lower_purple,self.upper_purple, 'purple')

    blue_center = self.find_center(hsv_image, self.lower_blue, self.upper_blue, 'blue')

    # Create a list of the centers
    centers = [red_center,green_center,purple_center,blue_center]

    # Loop through centers and add an small circle to visualize its center
    for center in centers:
        cv2.circle(image, (int(center[0]), int(center[1])), 5, (0, 255, 0), -1)

    # Return the updated image
    return image
```

Figure 2: Code to implement Step 1

Azzam Shaikh
Oct. 7, 2024

Step 2

Deliverable: Present the image taken from the virtual camera, the detected circle centers, a nd a snapshot of the related part of the code in your report.

Results:

Figure 3 shows the image taken from a different position and the respective circle centers.

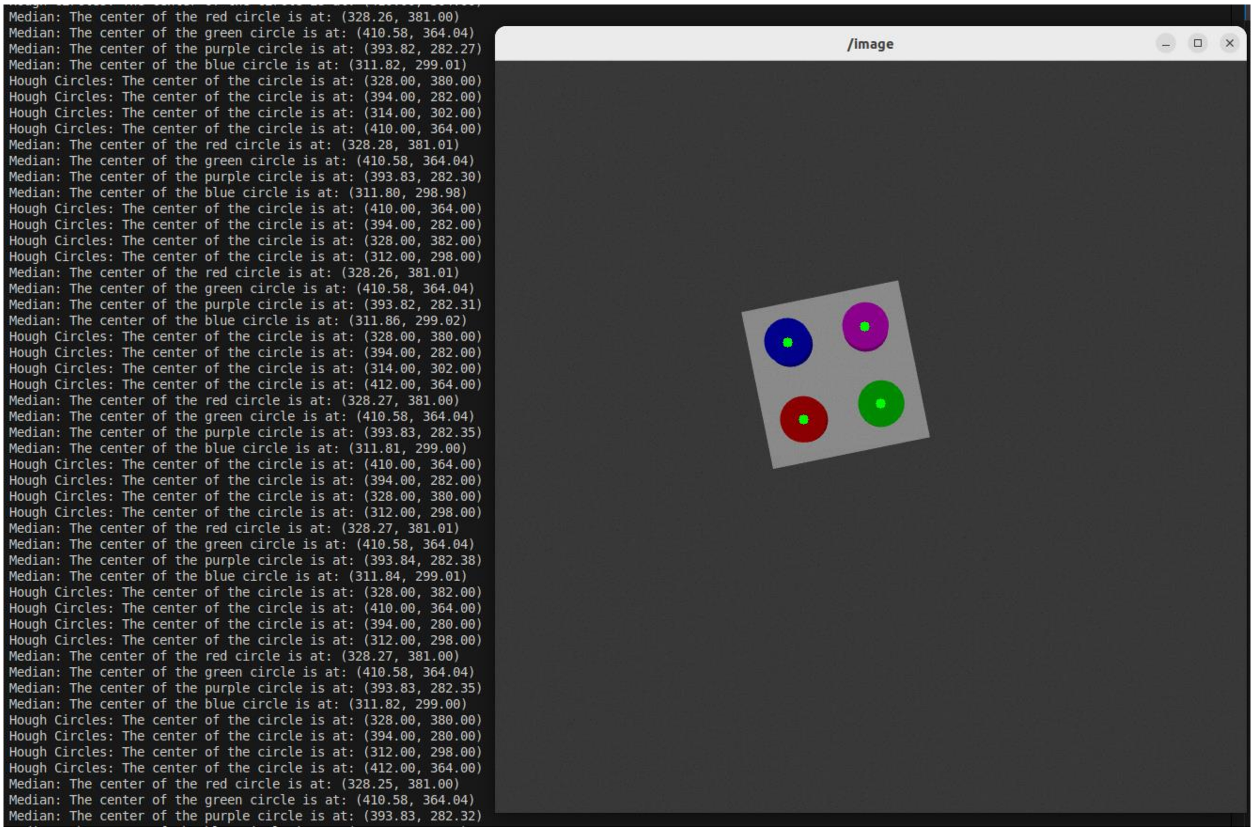

Figure 3: Virtual camera and circle centers and positions

Figure 4 shows the command used to move the camera.

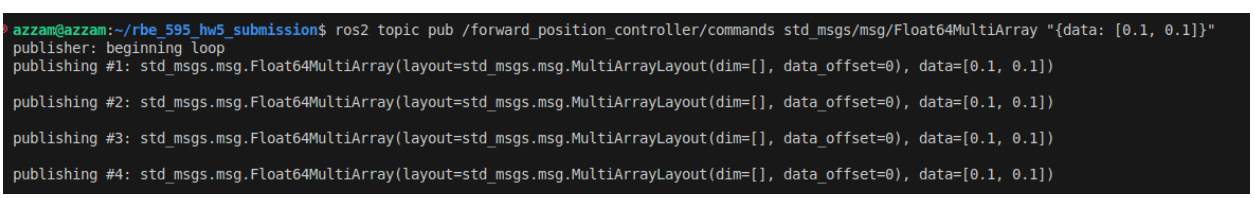

Figure 4: Command to move the robot

The code shown in Figure 2 is applied to this step as well.

Azzam Shaikh
Oct. 7, 2024

Step 3

Deliverable: Record the locations (x, y coordinates) of all the features over time during visual servoing. Plot them in the XY plane so that you visualize the trajectories of the features (you can record them in a file and plot in Matlab if you like). Include that plot, a snapshot of your code in your report as well as the ROS package that includes your implementation.

Results:

A side-by-side view of position 1 to position 2 can be seen in Figure 5.

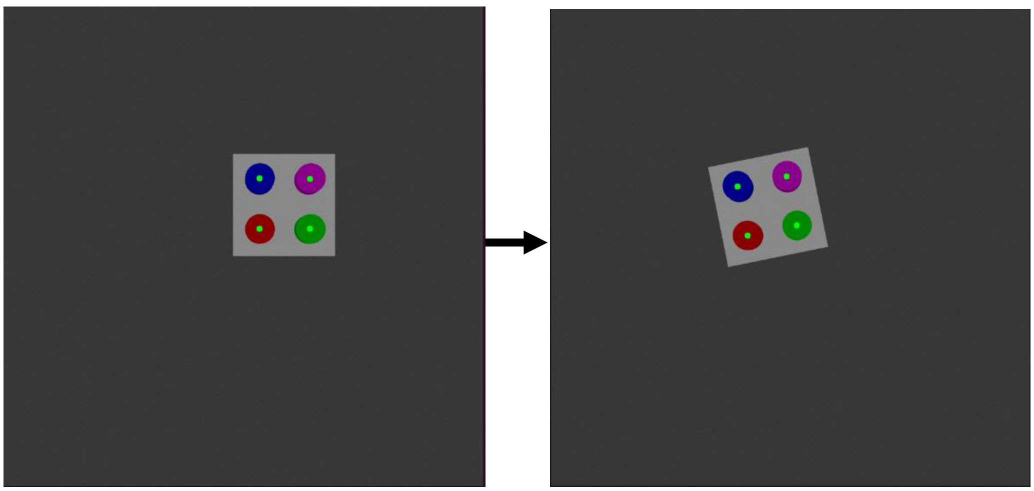

Figure 5: Position 1 to position 2

Figure 6 shows the path taken by each object center to go from position 1 to position 2.

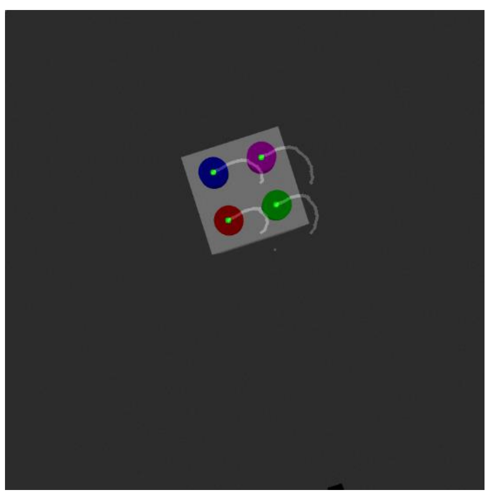

Figure 6: Trajectory over time of the 4 circle centers

A video of the trajectory is available with the submission for reference.

Azzam Shaikh
Oct. 7, 2024

Several snapshots of the code can be seen below. Full implementation can be seen in the /src/ folder and the respective packages.

The feature_extractor.py function in the opencv_test_py package is responsible for obtaining the center of each circle and publishing them as topics. Refer to Figure 7 for reference code.

```python
@staticmethod
def find_center(image,lower_range,upper_range,name):
    """
    Function to obtain the center of an image given upper and lower HSV ranges
    """
    # Create a mask for the specified color ranges
    mask = cv2.inRange(image, lower_range, upper_range)

    # Locate the x and y coordinates of all the pixels found in the mask
    pixels = np.column_stack(np.where(mask > 0))

    # In case no pixels are detected, return an empty message
    if len(pixels) == 0:
        msg = Int32MultiArray()
        msg.data = [int(0), int(0)]
        return msg

    # Calculate the x and y pixel averages to find the center
    center_x = np.mean(pixels[:, 1])
    center_y = np.mean(pixels[:, 0])

    # Create the output message that needs to be sent
    msg = Int32MultiArray()
    msg.data = [int(center_x), int(center_y)]

    return msg

def find_center_of_colors(self, image):
    """
    Function to find the center of all four cirlces in the image
    """

    # Convert the image to HSV colorspace
    hsv_image=cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Call the find_center method given an HSV converted image, and the predefined bounds
    red_center = self.find_center(hsv_image,self.lower_red,self.upper_red,'red')

    green_center = self.find_center(hsv_image,self.lower_green,self.upper_green,'green')

    purple_center = self.find_center(hsv_image,self.lower_purple,self.upper_purple, 'purple')

    blue_center = self.find_center(hsv_image, self.lower_blue, self.upper_blue, 'blue')

    # Create a list of the centers
    centers = [red_center, green_center, purple_center,blue_center]

    # Loop through centers and add an small circle to visualize its center
    for center in centers:
        cv2.circle(image, (center.data[0], center.data[1]), 5, (0, 255, 0), -1)

    # Return the updated image
    return centers, image
```

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

    # Call the find_center_of_colors to obtain the center of the images
    # Pass a copy of the current frame so the current frame doesnt get modified
    centers, image = self.find_center_of_colors(current_frame.copy())

    output_image = image

    # Publish the image and center topics
    self.publisher_.publish(self.br.cv2_to_imgmsg(output_image, encoding="bgr8"))
    self.red_publisher_.publish(centers[0])
    self.green_publisher_.publish(centers[1])
    self.purple_publisher_.publish(centers[2])
    self.blue_publisher_.publish(centers[3])
```

Figure 7: feature_extactor.py reference code

Azzam Shaikh
Oct. 7, 2024

The visual_servoing.py file in the controller package is responsible for computing image Jacobians and determining the camera velocity. Refer to Figure 8 for reference code.

```python
def controller_callback(self):
    """
    Controller callback function that computes the camera twist for control
    """

    # Define the list of current positions in meters
    current_poses = [self.get_image_plane_position(self.pixel_pose_red),
                     self.get_image_plane_position(self.pixel_pose_blue),
                     self.get_image_plane_position(self.pixel_pose_green),
                     self.get_image_plane_position(self.pixel_pose_purple)]

    # Define the list of reference positions in meters
    reference_poses = [self.get_image_plane_position(self.ref_red_circle),
                       self.get_image_plane_position(self.ref_blue_circle),
                       self.get_image_plane_position(self.ref_green_circle),
                       self.get_image_plane_position(self.ref_purple_circle)]

    # Compute the error vector (8x1)
    error_vector = self.compute_error_vector(reference_poses, current_poses)

    # Compute the image jacobian (8x6)
    Le = self.get_full_image_Jacobian(current_poses)

    # Compute vr = lambda * pinv(Le) * e [scalar * (6x8) @ (8x1)]
    vr = -0.3*(np.linalg.pinv(Le)@error_vector)

    # If the total error between the different circles is < 0.05, the servoing
    # is complete
    if np.abs(np.sum(error_vector)) < 0.05:
        self.get_logger().info(f"Goal Reached!")
        # Send 0 twist
        msg = Twist()
        msg.linear.x = 0.0
        msg.linear.y = 0.0
        msg.linear.z = 0.0
        msg.angular.x = 0.0
        msg.angular.y = 0.0
        msg.angular.z = 0.0

    else:
        self.get_logger().info(f"Current total error: {np.sum(error_vector)}.")
        # Send vr twist
        msg = Twist()
        msg.linear.x = float(vr[0])
        msg.linear.y = float(vr[1])
        msg.linear.z = float(vr[2])
        msg.angular.x = float(vr[3])
        msg.angular.y = float(vr[4])
        msg.angular.z = float(vr[5])

    # Call the plot function to plot the trajectory
    self.plot_positions([self.pixel_pose_red,
                         self.pixel_pose_blue,
                         self.pixel_pose_green,
                         self.pixel_pose_purple])

    # Publish the twist
    self.controller_publisher_.publish(msg)
```

Figure 8: visual_seroving.py reference code

Azzam Shaikh
Oct. 7, 2024

The rrbot_controller.py file in the robot_controller package is responsible for taking the camera twists and converting them into joint velocities. Refer to Figure 9 for reference code.

```python
def twist_callback(self, msg:Twist):
    """
    FUnction that extract the twist data and create it into a numpy array
    """
    vr = np.empty(6)
    vr[0] = msg.linear.x
    vr[1] = msg.linear.y
    vr[2] = msg.linear.z
    vr[3] = msg.angular.x
    vr[4] = msg.angular.y
    vr[5] = msg.angular.z
    self.vr = vr.reshape((6,1))

def joint_states_callback(self, msg:JointState):
    """
    Function that gets the current joint positions and jacobian
    """

    # Get current joint positions
    current = msg.position
    j1 = current[0]
    j2 = current[1]

    # Predefine variables for the matrix
    l1, l2 = 1,1
    J00 = l1*np.sin(j1)-l2*np.sin(j1+j2)
    J01 = -l2*np.sin(j1+j2)
    J10 = l1*np.cos(j1)+l2*np.cos(j1+j2)
    J11 = l2*np.cos(j1+j2)

    # Create the Jacobian
    self.jacobian = np.array([[J00, J01],
                              [J10, J11],
                              [0, 0],
                              [0, 0],
                              [0, 0],
                              [1, 1]])

    # Publish joint commands from below
    self.send_joint_commands()

def send_joint_commands(self):
    """
    Function to publish the joint commands
    """
    # Compute joint commands
    joint_cmds = (np.linalg.pinv(self.jacobian)@self.vr).flatten()
    # self.get_logger().info(f"Sending joint commands: {joint_cmds}")

    # Convert the joint commands to appropriate data format
    self.msg.data = [float(joint_cmds[0]), float(joint_cmds[1])]

    # Publish the message
    self.publisher_.publish(self.msg)
```

Figure 9: rrbot_controller.py reference code

Run instructions are available with the submission for reference.