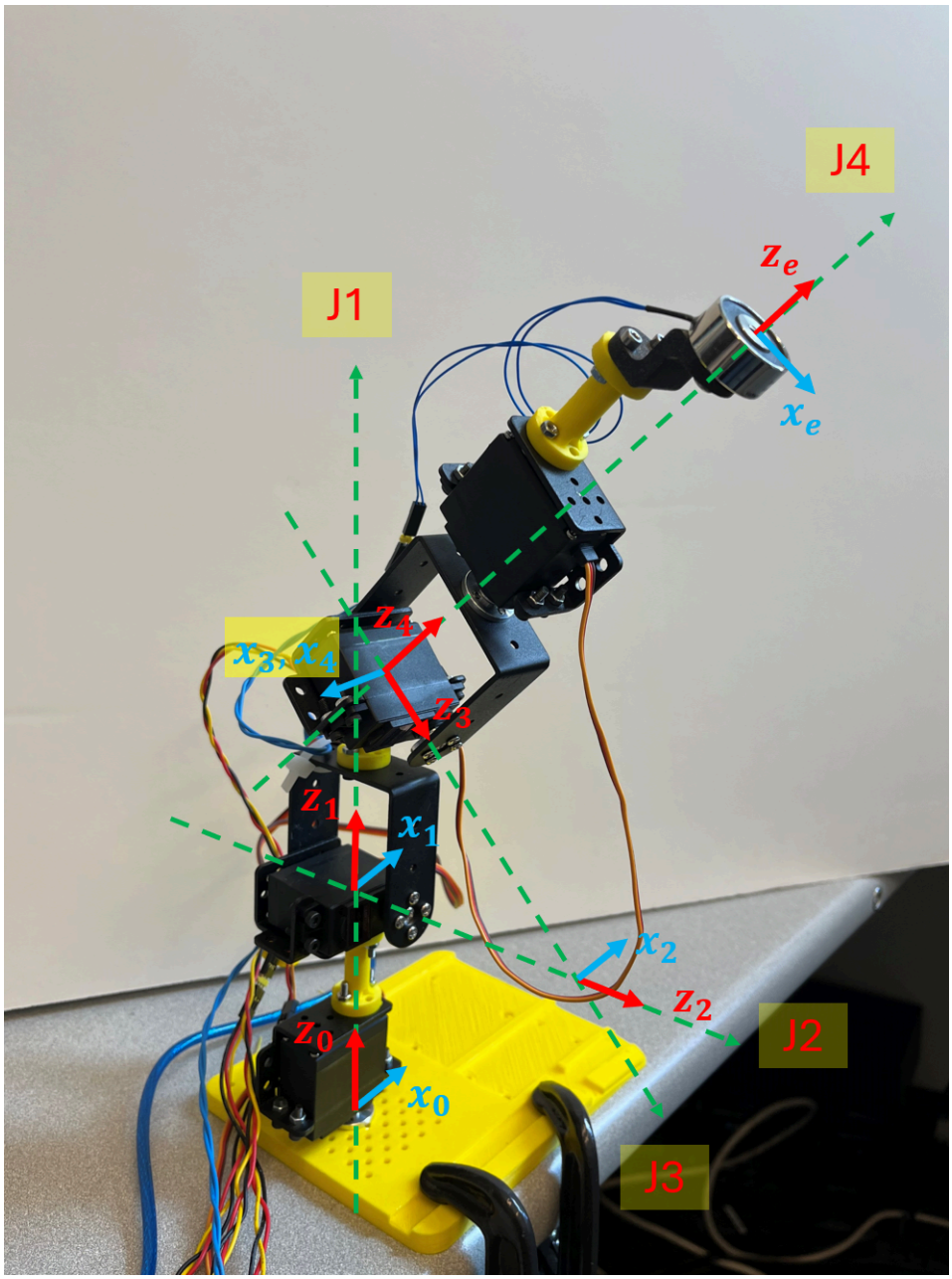
Models of Robot Manipulation
Project Manual

Haonan Peng, Dun-tin Chiang, Niveditha Kalavakonda, Haokun Feng, Armina Moghadasi, Blake Hannaford
penghn@uw.edu



1. Introduction and Course Material

1.1 Introduction

This is the associated project for EE 543 Models of Robot Manipulation. In this project, you will be guided to build a robot arm with 3-4 joints from scratch. Based on EE 543 course materials (all open-sourced, links in 1.2), you will learn and implement:

- Design and assemble robot arm from provided servo motors, arm links, power supply, and so on
- Assign frames for each joint and derive proximal DH parameters
- Apply forward kinematics and analyze the workspace of the robot
- Solve inverse kinematics and achieve intuitive end-effector position control in Cartesian space
- Derive the Jacobian matrix and achieve end-effector velocity control, and analyze the singularity (optional).

Note: An example and solution will be given after each milestone assignment due for your reference. However, for grading purposes, you cannot copy any example solution and code as your assignment, unless stated otherwise.

1.2 Course Materials and Prerequisites

To smoothly follow this tutorial, it is recommended to learn through the **textbook Chapters 1-5: Models of Robot Manipulation** (open-sourced, copyright Prof. Blake Hannaford). Besides the textbook, it is also recommended to have knowledge/skills of:

- Basic Linear algebra and Trigonometric functions
- Basic Python programming

Although designing and building robot arms also requires mechanical design, motor control, and so on, these parts are provided in this tutorial so that we can focus on the course topic: Models of Robot Manipulation.

All provided code can be found at:
https://github.com/HaonanPeng/EE543-Robot-Manipulation-Project
You are free to make any changes to the code as you need.

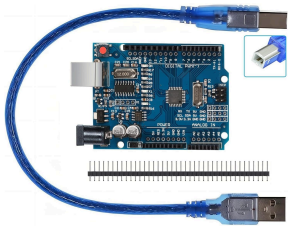
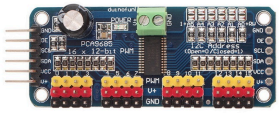
2. Safety Manual

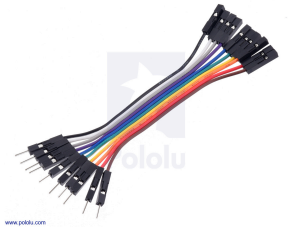
Safety is always the No.1 priority!

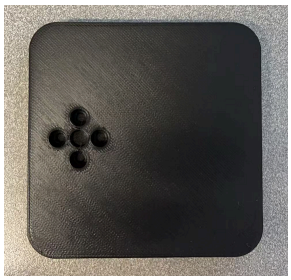
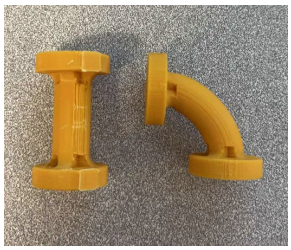
- If the robot arm is powered on, **always have at least one person supervising the robot** and ready to **turn off the power**.
- **Turn off the power immediately in the event of an emergency**, **this is always the first step.** Then unplug the power supply.
- **Always make sure you are safe first**, then try to save the robot.
- If you are not sure of anything, **ask the instructor or TA**.
- Ensure a **clean workspace**. Make sure that there are only the necessary instruments and components in your workspace. Especially, beware of any combustible.
- **Ware glasses.** Debris from a failing component can hurt your eyes.
- **Beware of water!** Water should never meet with instruments and components. Always make sure your hand is dry before doing any operation.
- Always **pay full attention** when working on labs. Danger hides behind carelessness.
- **Do not leave the powered arm alone**. If you need a pause, turn off the power supply.
- **Components can be very hot!** Although the robot uses only low voltages. If everything is correct, nothing should be burning hot. However, shortcuts can make components burning hot. If the robot does not work properly, turn off the power and wait until cools down.
- **No food or drinks when doing labs.**

3. Design and Assemble the Arm

3.1 Component Checklist

Item	Figure	Amount	Note
Arduino Uno MCU		1 Pcs	
PCA9685 Servo Control Board		1 Pcs	
MG996R Servo Motor		5 Pcs	
Servo Disc		> 4 Sets	Each set includes: Servo Disc x 1 M3x5mm screw x 5
Servo Bracket Counter Side Bearing		> 4 Sets	Each set Includes: Bearing x 1 M3 nut x 1 M3x9mm screw x 1

Servo Motor Mounting Screws		4 Sets	Each set Includes: M4x7mm screw x 4 M4 nut x 4
Servo Brackets - Mounting		4 Pcs	
Servo Bracket - U Shaped		4 Pcs	
Power Supply		1 Pcs	
Jumper wires		> 4 Pcs	

3D-Printed Base		1 Pcs	
3D-Printed Links		4 Pcs	
Table Clamp		2 Pcs	
Screw Driver		1 Pcs	
Needle Plier		1 Pcs	
Grasper		1 Pcs	
Zip ties			

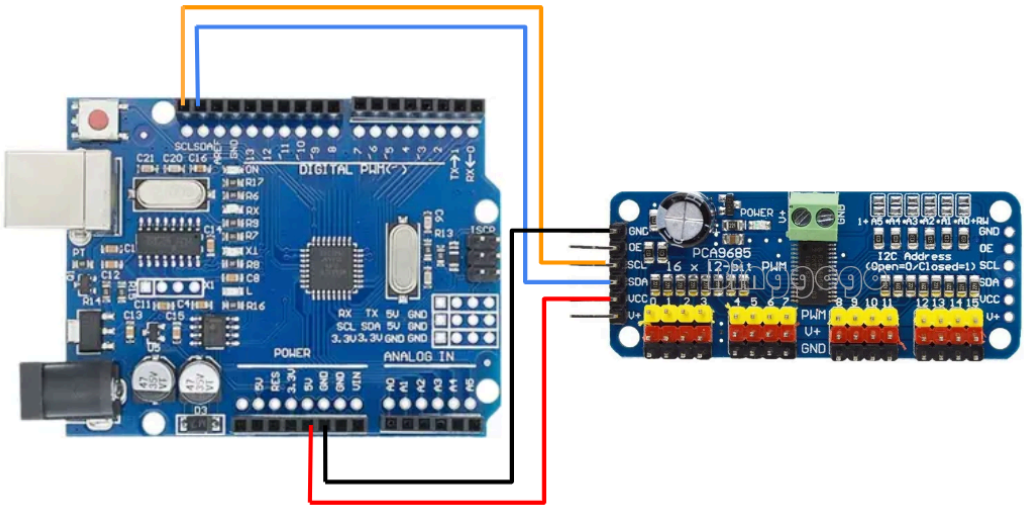
3.2 Electronics Assembly

All provided code can be found at:
https://github.com/HaonanPeng/EE543-Robot-Manipulation-Project
You are free to make any changes to the code as you need.

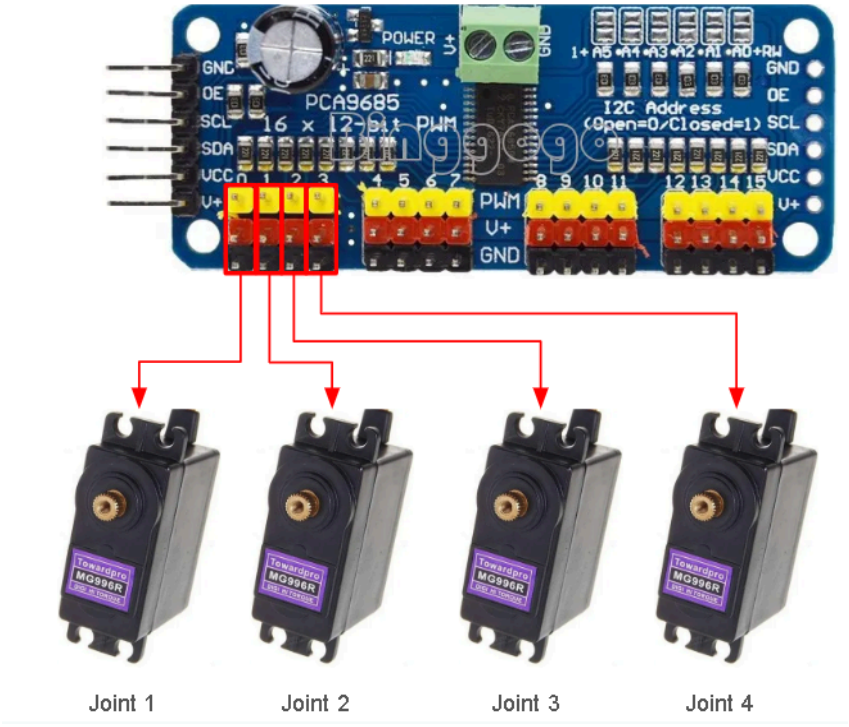
Before you start designing your own robot arm, it's important to reset all the servo motors to 0 position. It gives you a fresh start and helps you think about the range of motion of each joint.

To do that, the first thing you need is a simple system that can control the servo motor.

- **Connect the Arduino Uno to PCA9685**



- **Connect the Servo motors to PCA9685**



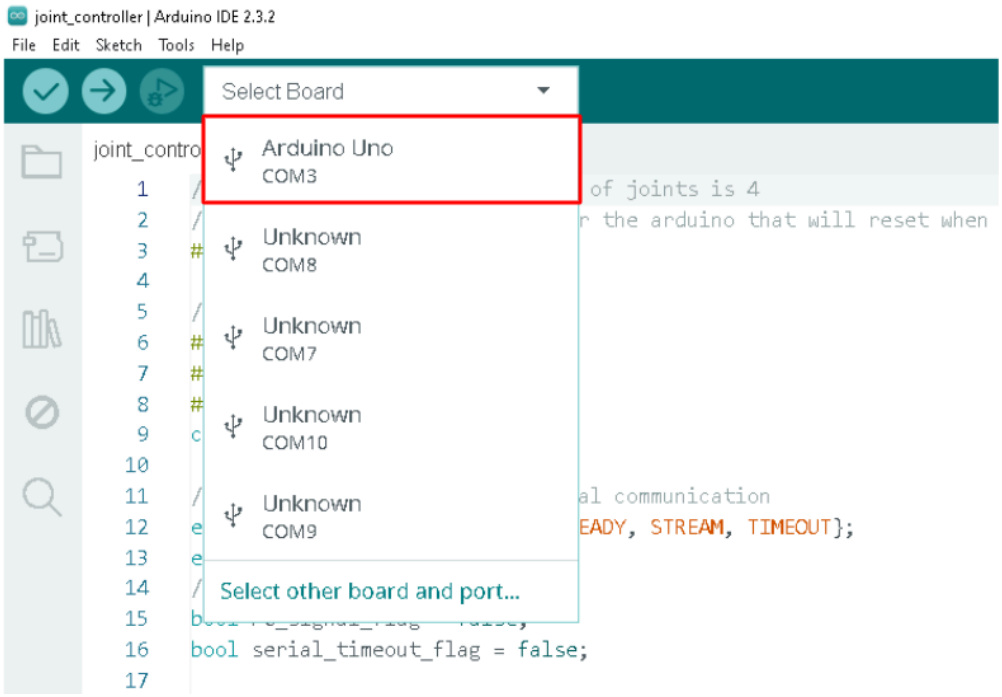
- **Connect the power module to PCA9685**

- **Flash the firmware to the Arduino board**
  - Download the code from GitHub
  - Under the main folder, go into “Arduino_Code” folder, and open the “joint_controller.ino” file.
  - Make sure the Joint number is 4

joint_controller.ino

```
1  // For EE543 project, the amount of joints is 4
2  /*This version of firmware is for the arduino that will reset when the serial port is connected*/
3  #define JOINT_NUM 4
4
```

  - Compile and flash the code to Arduino. Take note of which COM port your Arduino is using, for example, the Arduino here is communicating with the PC through Serial port COM3



- **Home all the servo motors to the zero position**
  - Under the main folder, open the “Python controller” folder in your selected Python IDE.
  - Open the “robot_controller.py” in your Python IDE.
  - Within the init function under the “robot_controller” class, search for the “self.joint_num” variable and make sure the value is 4.

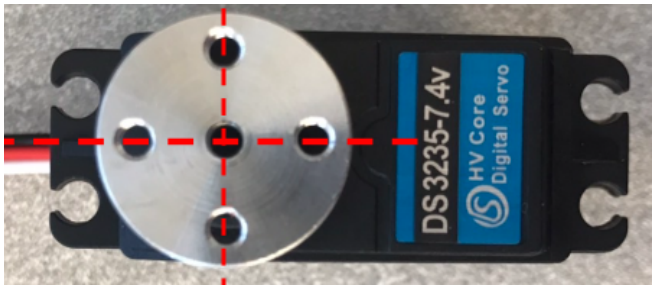
```
class robot_controller():
    def __init__(self) -> None:
        #define robot parameter
        self.joint_num = 4
        self.joints_goto_tolerance = 10e-3
```

  - Within the same init function, search for the “self.com_port” variable and make sure it aligns with the COM port number.

```
#define the serial communication parameter
self.com_port = 'COM3' # change it if needed
self.com_baudrate = 115200 #bps
self.com_frequency = 30 #Hz
```

  - Under the same folder, open the “servo_homing.py”, and simply run the script. You should hear a brief noise from servo motors. Now, all of your servo motors are reset to the 0 position.

- **Install the servo discs and mark out the zero position**
  After completing the steps above, don't disconnect the motors from the PCA9685 just yet. Install the servo disc on each motor. You can think about how you want your robot arm to be in the zero position. For ease of demonstration and frame assignment, the example here will align the screw hole of the servo disc with the central line of the servo motor.



  After that, install the central screw and mark out the zero position on the disc. The purpose of the marker is that if you accidentally rotate the motor shaft during the arm assembly (which is very likely), you won't lose track of the zero position of your motor.

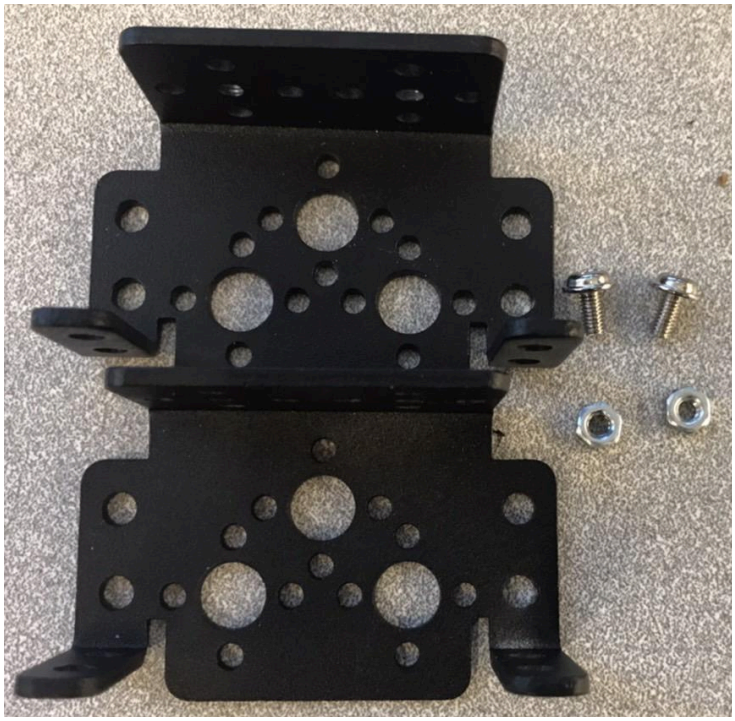
3.3 Arm Assembly

To this step, you should have all of your servo motors homing to zero position, mounted the servo disc, and marked the zero position. At this point, you can disconnect the servo motors from the PCA9685 board.

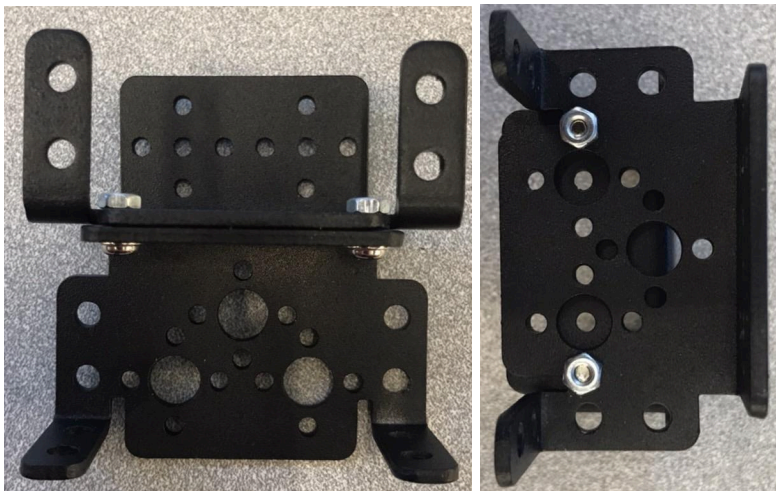
Follow the instructions below, you will build a 4DOF robot arm step by step. The example CAD assembly and DH parameters analysis will be based on this particular design. However, feel free to be creative and try out different combinations of arm assembly.

- **Assemble the mounting brackets for Joint 1 and Joint 2**

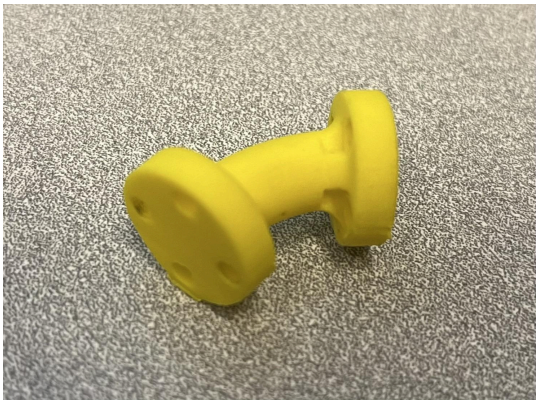
For this part, you need two mounting brackets, two M3X6 pan head screws, and 2 M3 nuts:



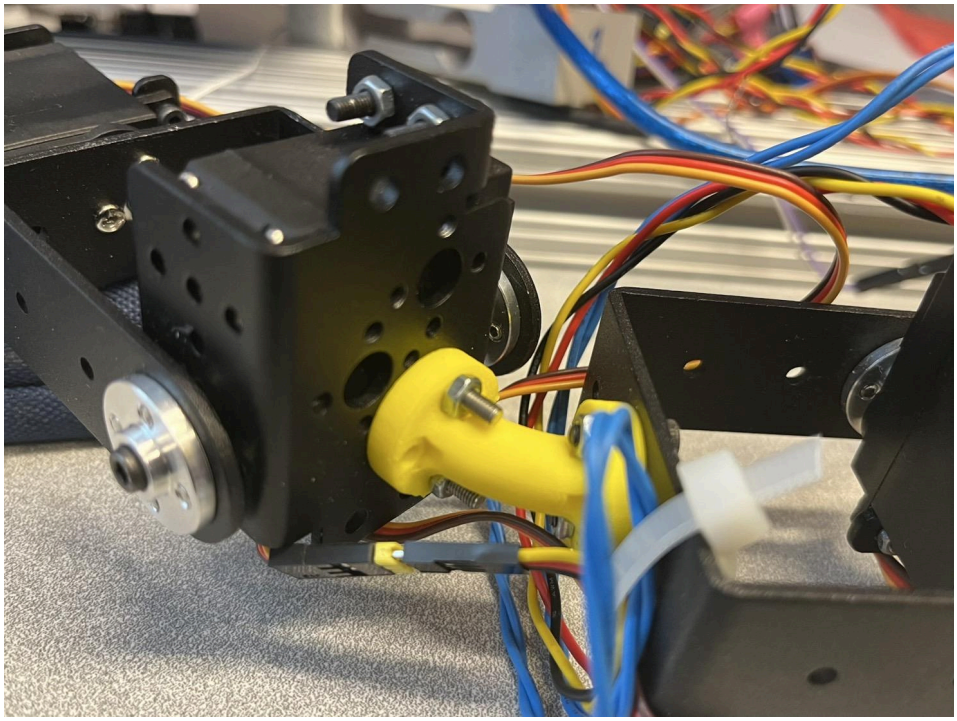
The completed assembly is shown below:



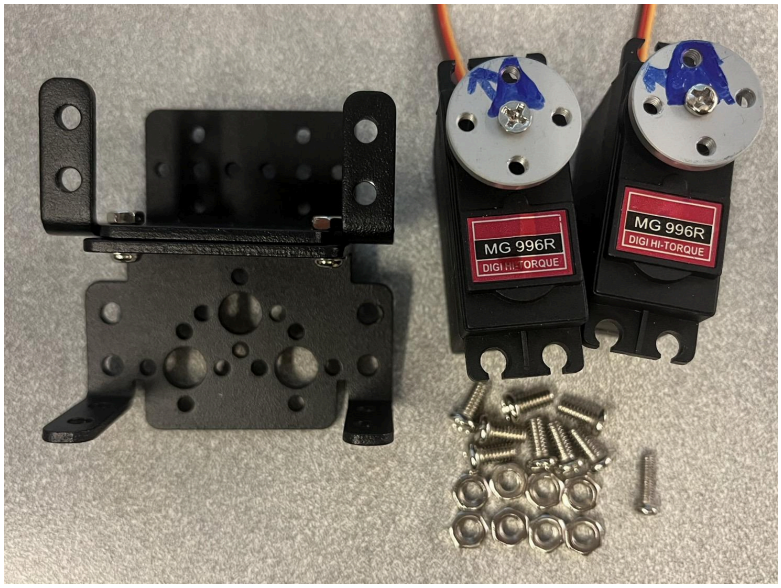
- **Mounting bending and offset links**
  The mounting plate of the 3D-printed links features pre-drilled holes compatible with the servo bracket. Secure the plate using eight M3x9mm screws and M3 nuts. For the bending link, inserting a screw on the concave side may be challenging. If necessary, you may omit this screw, provided the overall assembly maintains sufficient structural integrity.

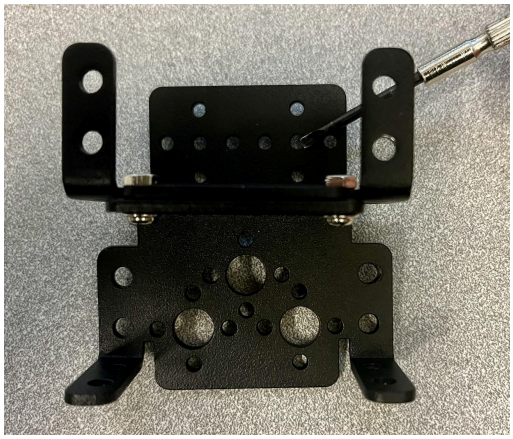
The following is an example of mounting the bending link. You can use any set of mounting holes on the bracket, but ensure the link is mounted before attaching the servo motor. In this example, a 45° bending link is used. However, note that bending angles other than 0°, 90°, or 180° may complicate closed-form solutions for inverse kinematics in real applications.

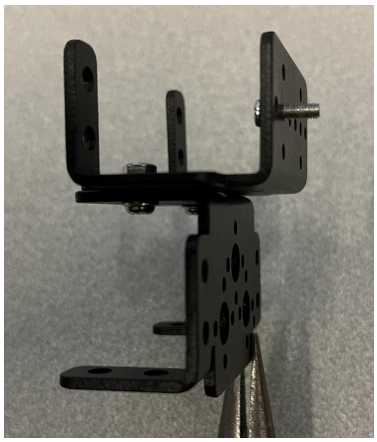


- **Mount the Servo motors to the Joint1 and Joint2 bracket**
  The required components include the assembled Joint 1 and Joint 2 brackets, two servo motors, eight M4x7 screws, eight M4 nuts, and one M3x9mm screw.

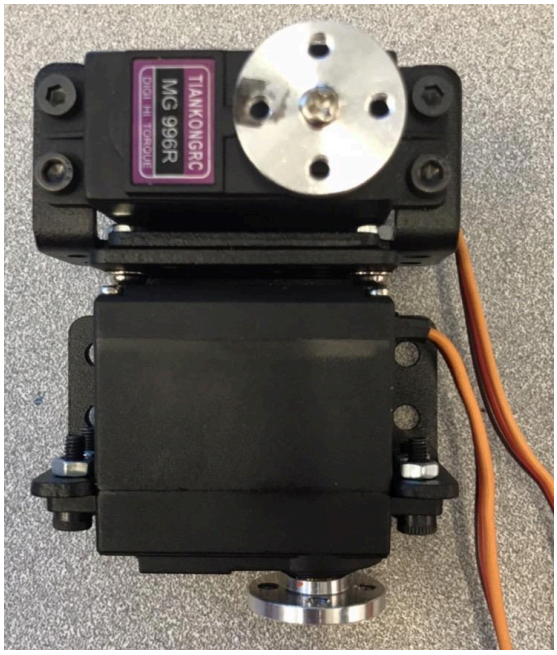


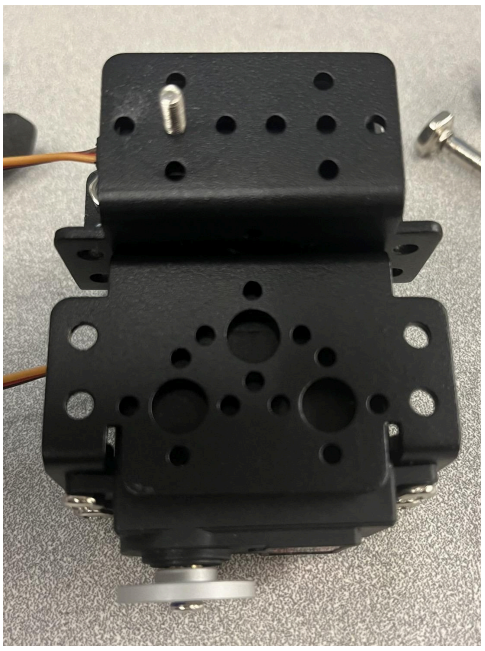
Before mounting the servo motors, ensure the screw for the counter-side bearing is installed on the U-shaped bracket. Start by identifying the bracket that will connect to the U-shaped bracket. Then, insert an M3x9mm screw into the bracket, ensuring it aligns with the motion axis of the servo motor.



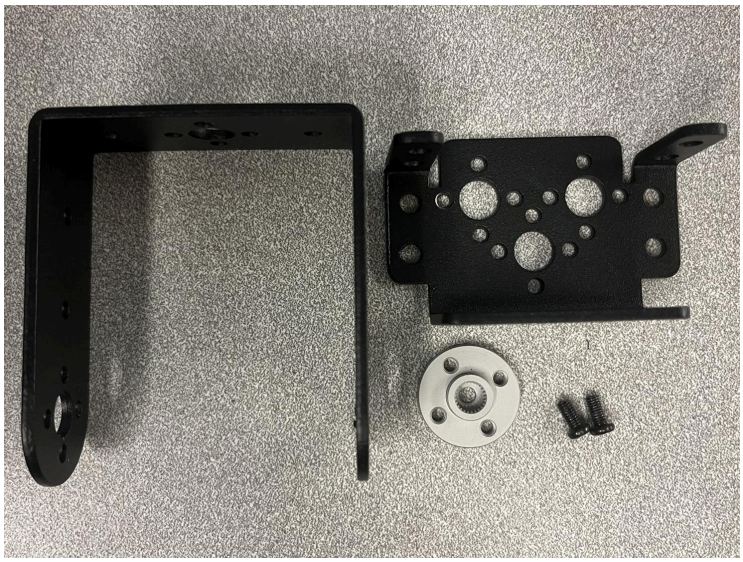


Mount the servo motors onto the bracket using M4 screws and nuts. Use needle-nose pliers to hold the nuts in place, ensuring they are securely tightened. The completed assembly is shown below:

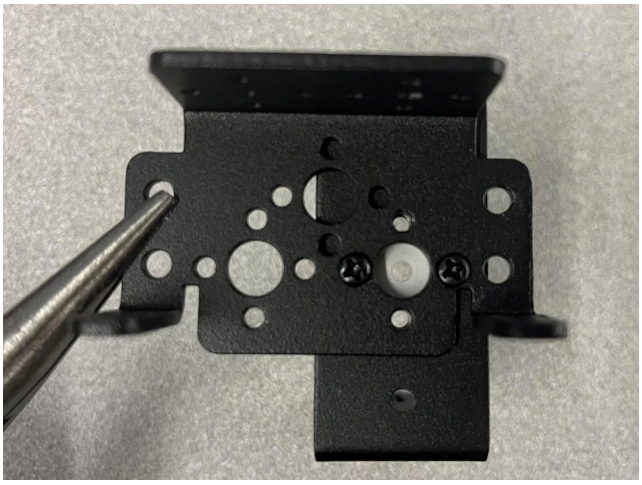




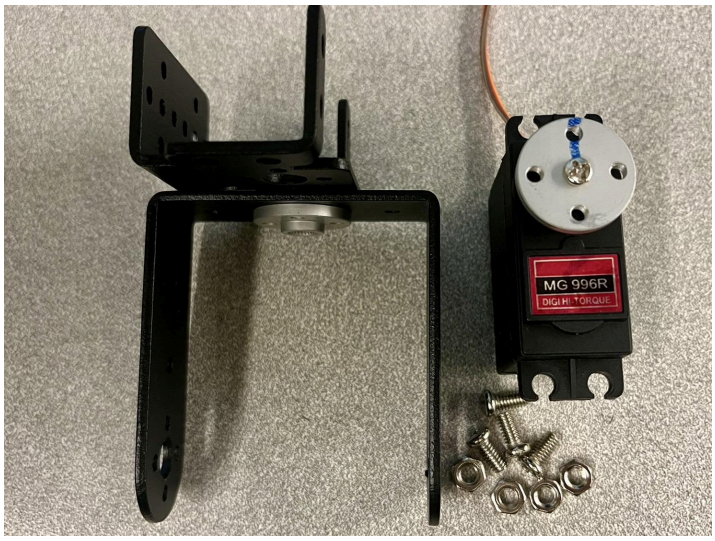
- **Assemble the mounting bracket of Joint3 with the U-shaped bracket**
  For this section, you will need the following components: a mounting bracket, a U-shaped bracket, a servo disc, and two M3x5mm screws.



The completed assembly is shown below:

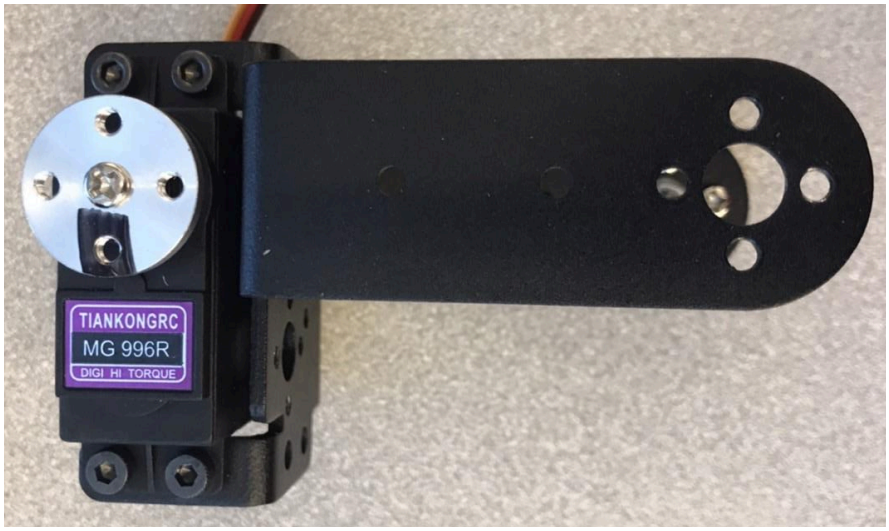


- **Mount the Servo motors to the joint3 brackets**
  The required components for this assembly are the assembled Joint 3 bracket, a servo motor, four M4x7mm screws, four M3 nuts, and one M3x9mm screw.



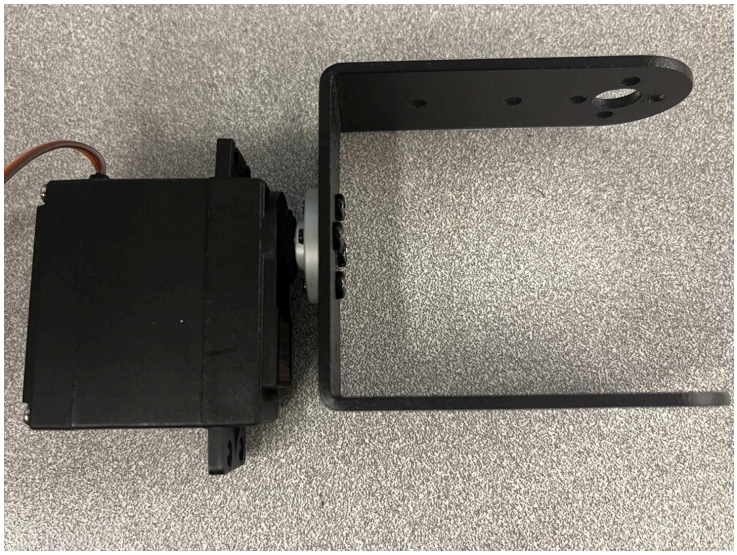
This servo motor will be connected to a U-shaped bracket. Before mounting the servo motor, ensure that the M3x9mm screw is securely installed along the motion axis.

The completed assembly is shown below:



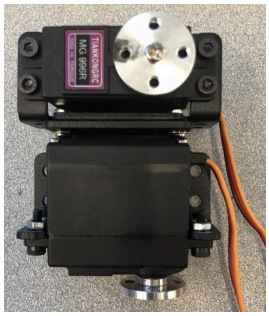
- **Assemble the Joint4 with the U-shaped bracket**
  For this step, you will need the following components: a servo motor, a U-shaped bracket, and four M3x4 pan head screws included with the servo disc.

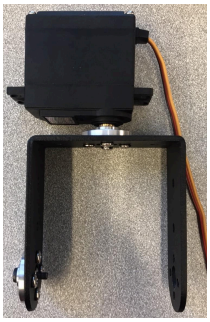
The completed assembly is shown below:



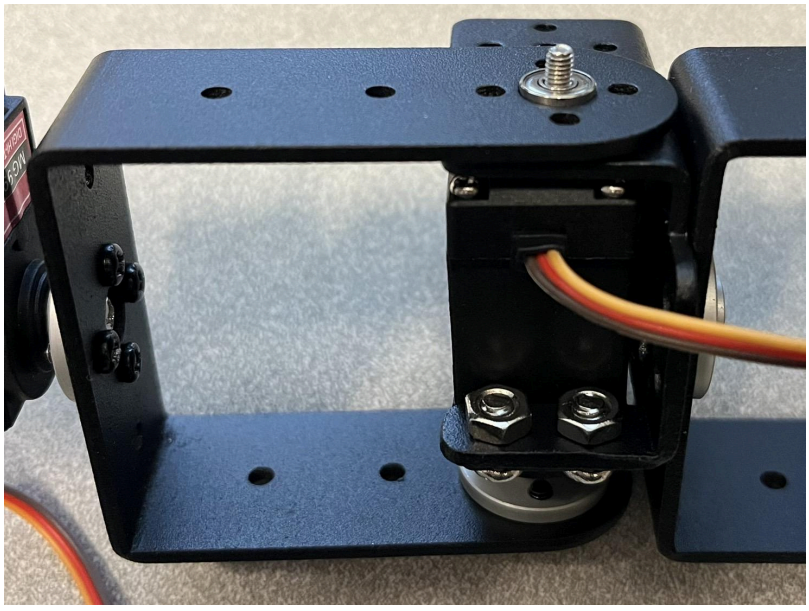
- **Mount the U-shaped brackets to the servo motors**
  To this point, you should have the following module:

Joint 1 & Joint 2	Joint 3	Joint 4

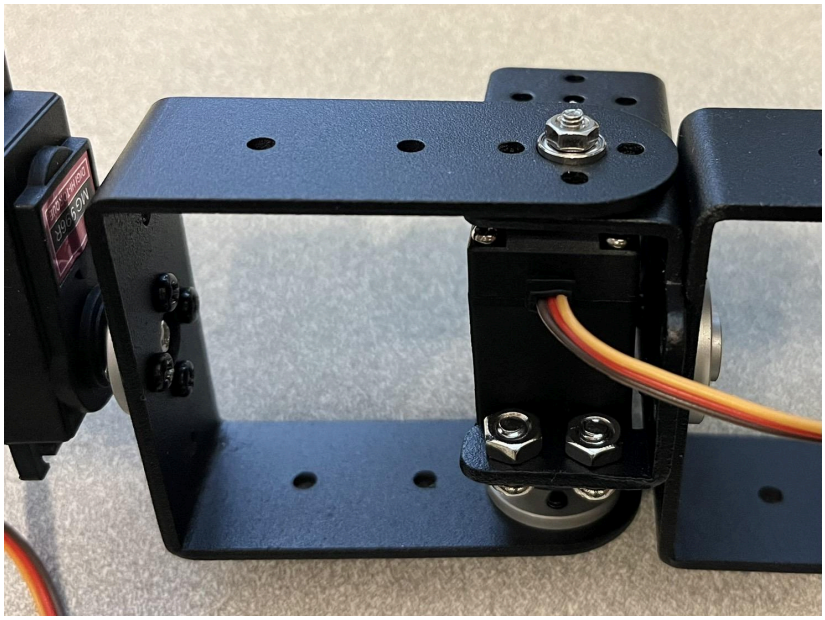




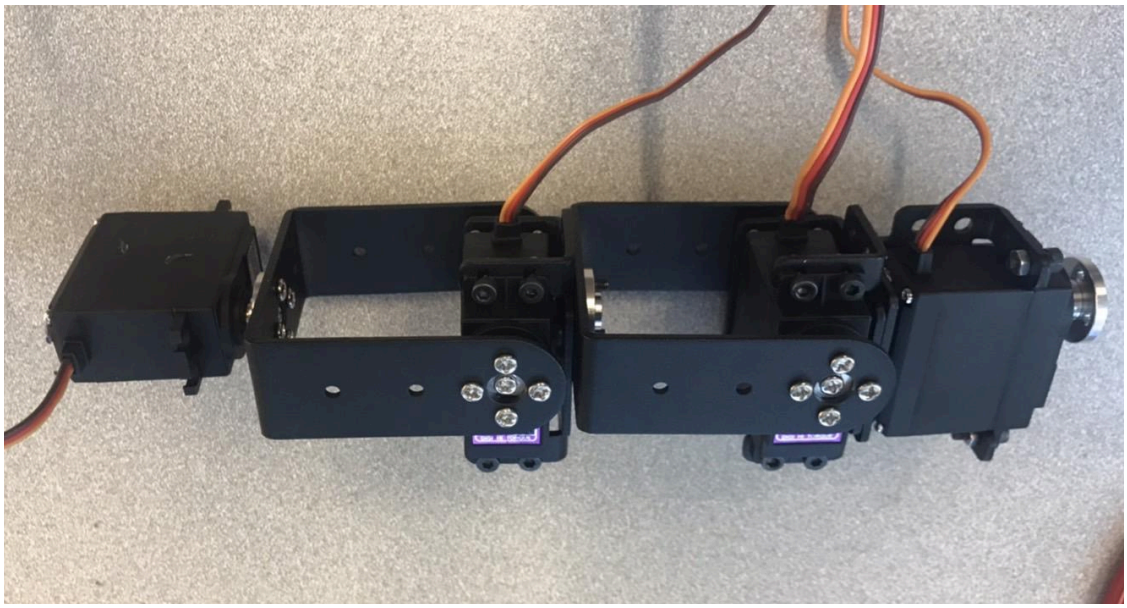
Assemble the complete arm by connecting the U-shaped brackets to the servo discs. Before proceeding, make sure the counter-side bearing is mounted on the M3x9mm bolt. Attaching the U-shaped bracket may be challenging due to the protruding shaft, but don’t worry—simply bend the U-shaped bracket slightly and carefully maneuver it into place. It’s flexible enough to handle without damage. The counter side of the U-shaped bracket should look like this:



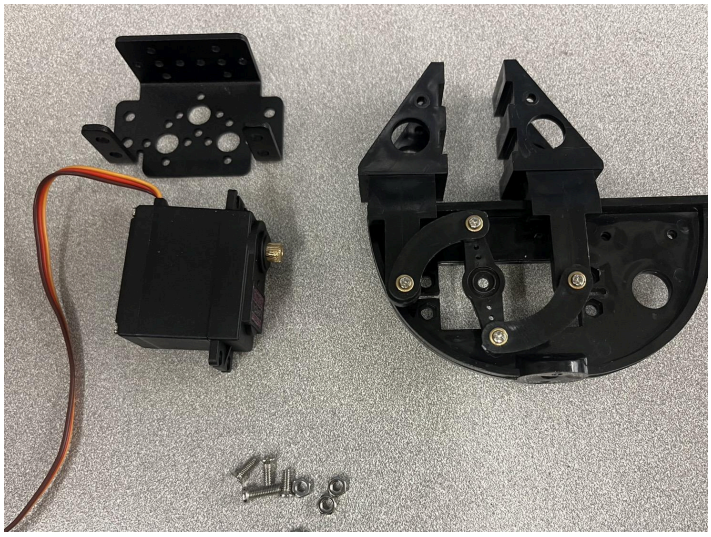
After mounting the U-shaped bracket, adjust it to align with the desired zero position, then secure the bracket to the servo disc. The counter side of the U-shaped bracket should hold the bearing securely, but you can add an M3 nut to prevent any sliding.



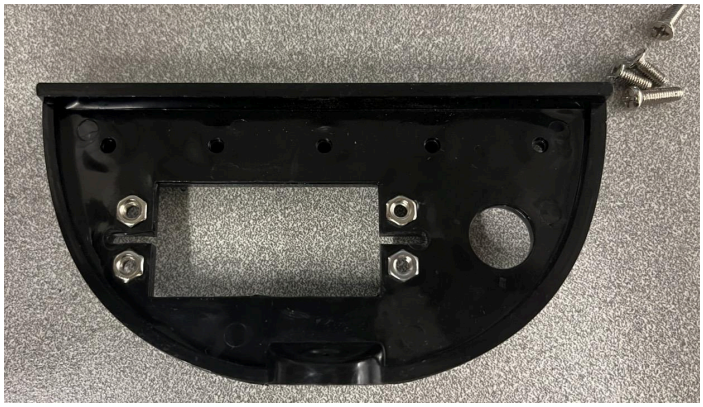
The completed arm assembly is shown below:



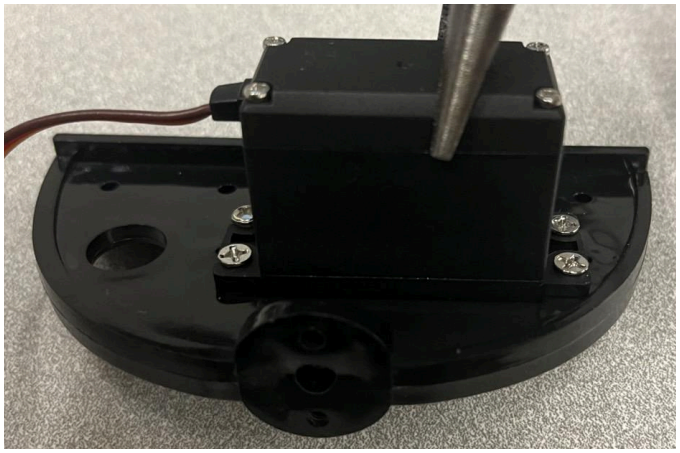
- **Assemble the grasper**
  The components required for this step are a servo motor, a grasper kit, and four sets of M3x9mm screws and nuts.



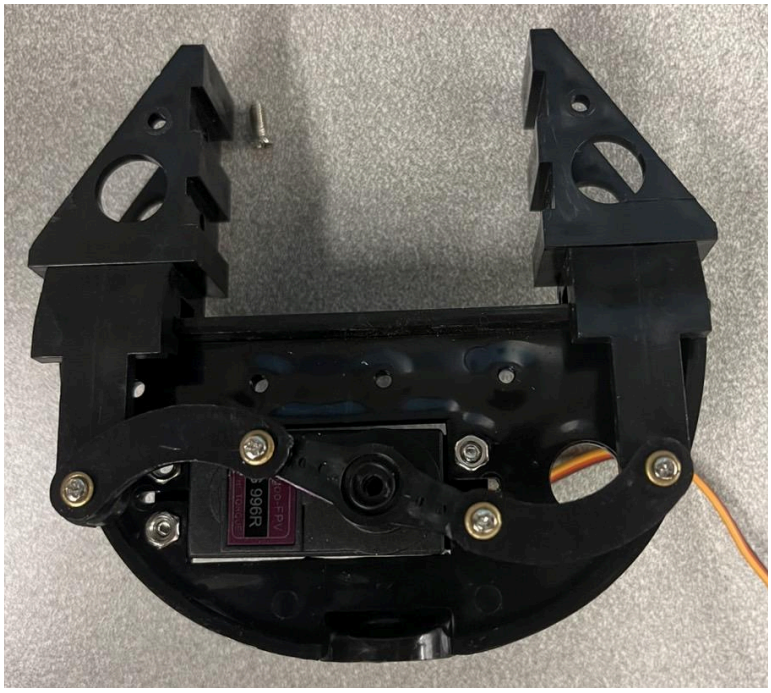
Slide the grasper jaws off the base, then insert four M3 nuts into the hexagonal recesses on the base.



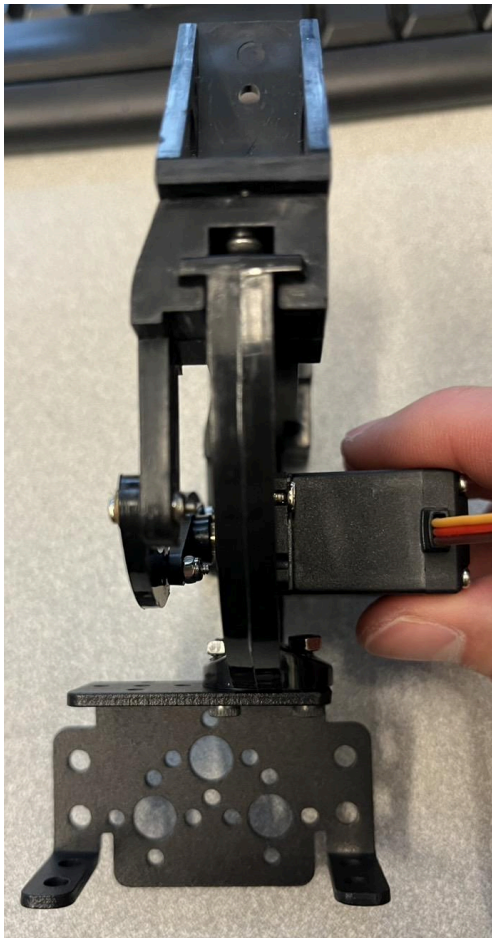
Loosely attach the servo motor to the opposite side of the base, ensuring the motor shaft is aligned close to the center of the base.

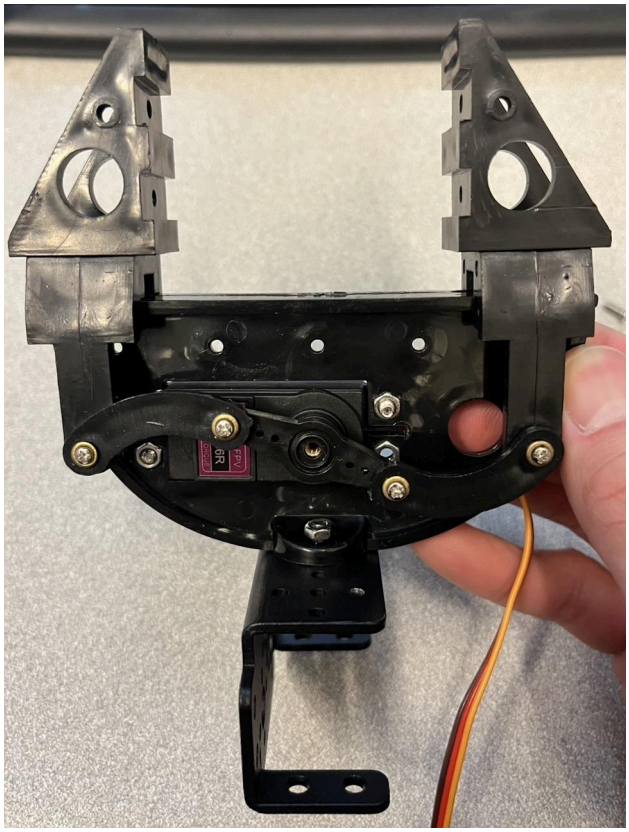


Reattach the grasper jaws to the base, keeping them in the open position. Align the connector with the motor shaft and press it down. (You can tape the jaws in place during this step to prevent slipping.) Once aligned, tighten the screws on the servo motor to secure the assembly.

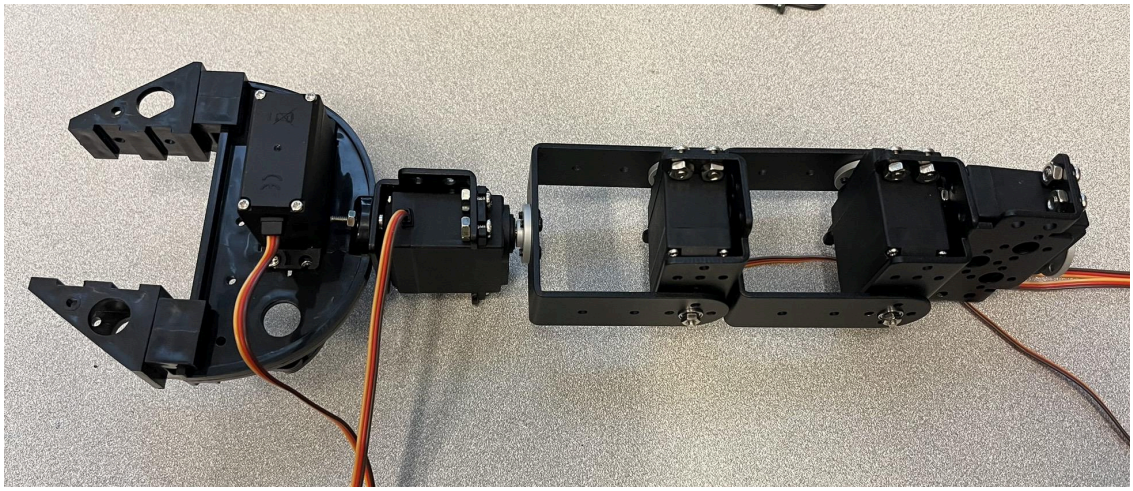


- **Mount the grasper to the arm**
  At this stage, the last joint of the arm is the servo motor, which still needs to be mounted with the servo bracket. Find a suitable servo bracket and attach the grasper to one of the mounting hole sets. The example shown aligns the grasper with the last joint axis, but you are free to choose any position for mounting the grasper. You will need two sets of M3x9mm screws and nuts to complete this step.



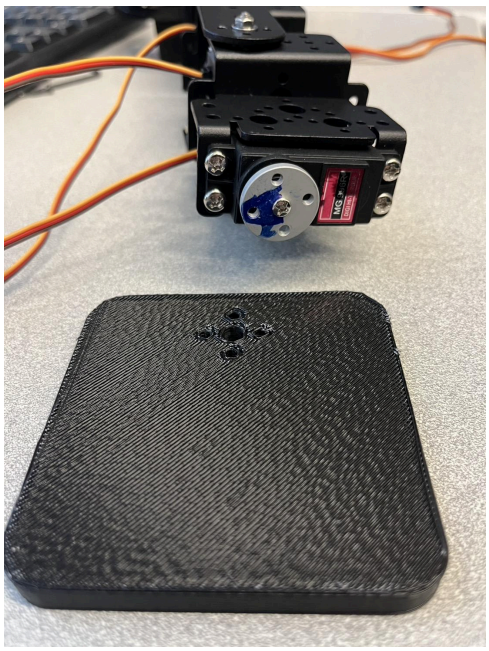


Next, mount the servo bracket to the last servo motor on the arm. You will need four sets of M4x7mm screws and M4 nuts to secure it. The completed assembly should look like this:



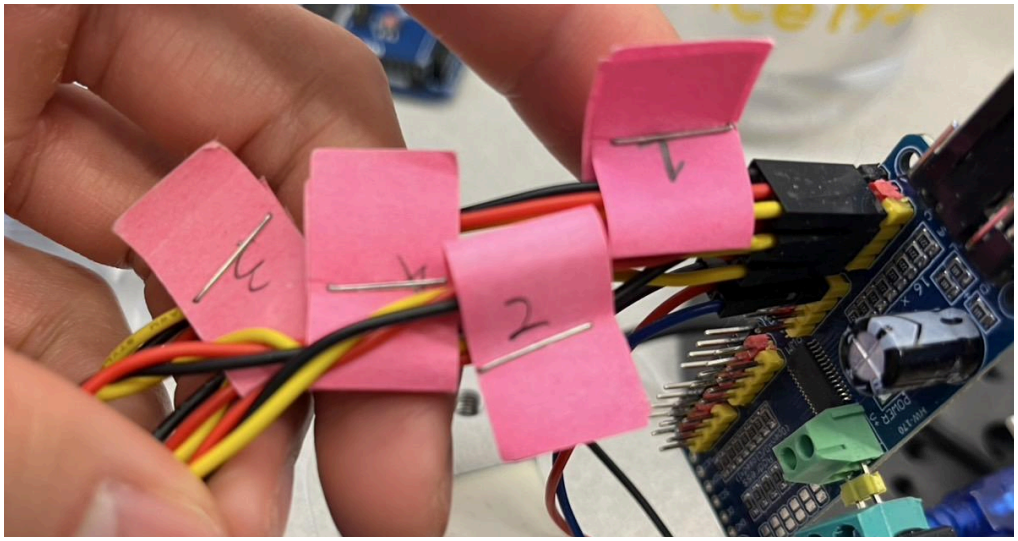
- **Mount the arm to the base**

The mounting hole on the base aligns with the one on the servo disc. Before mounting, ensure the zero position of Joint 1 is properly set. Use four M3x6mm screws to secure the arm in place.



If the screw appears too short to connect the servo disc and the base, use a screwdriver to apply pressure until the screw reaches the bottom of the counterbore hole on the base. Once this is done, the arm structure assembly is complete.

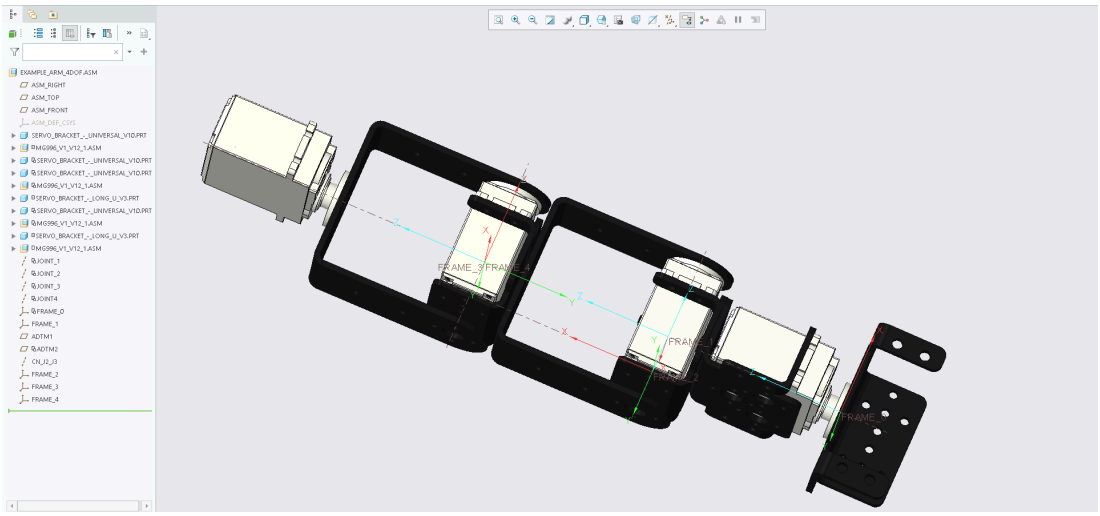
- **Extend and secure the motor cables**
  If needed, use the provided jumper wires to extend the servomotor cables. Be sure to mark the joint number at the end of each cable group to avoid plugging them into the wrong ports.



- **Connect the servo motor cables to PCA9685**

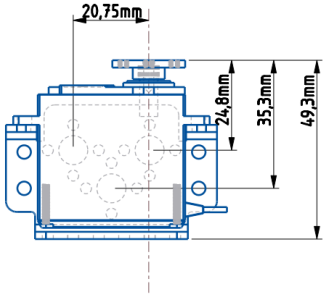
3.5 Preparation for Serial Manipulator Kinematics

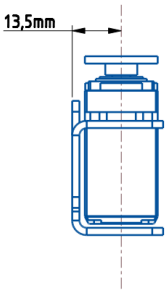
At this stage, you should have a fully assembled arm and be ready to begin modeling the robot's kinematics. CAD models of all components are available, allowing you to assemble the arm in your CAD software and perform the necessary measurements there. **Usage of CAD software is optional.**

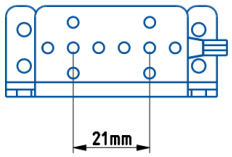


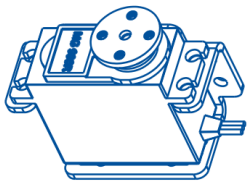
However, proficiency in CAD is not required for this project. Instead, we will provide the critical dimensions of each component so you can determine the DH parameters.

- Servo Motor with Bracket

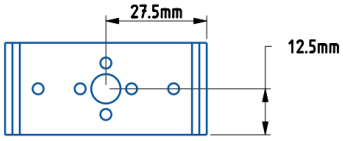


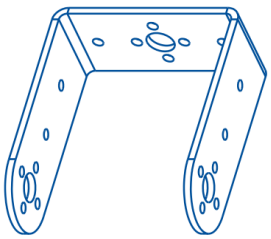


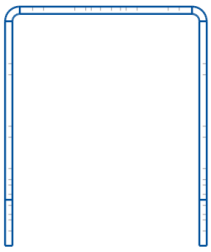


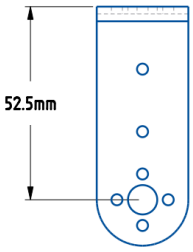


- U-shaped Bracket

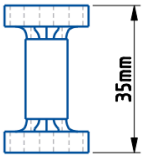




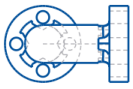


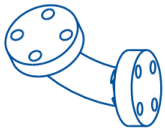


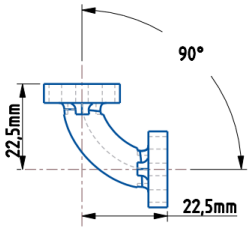
- Offset Link



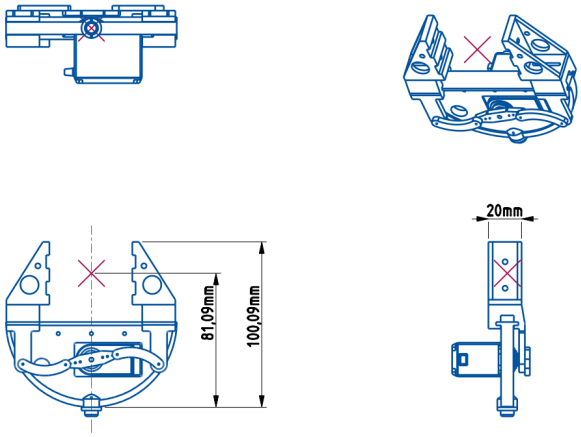
- Bending Link







- Robot Gripper

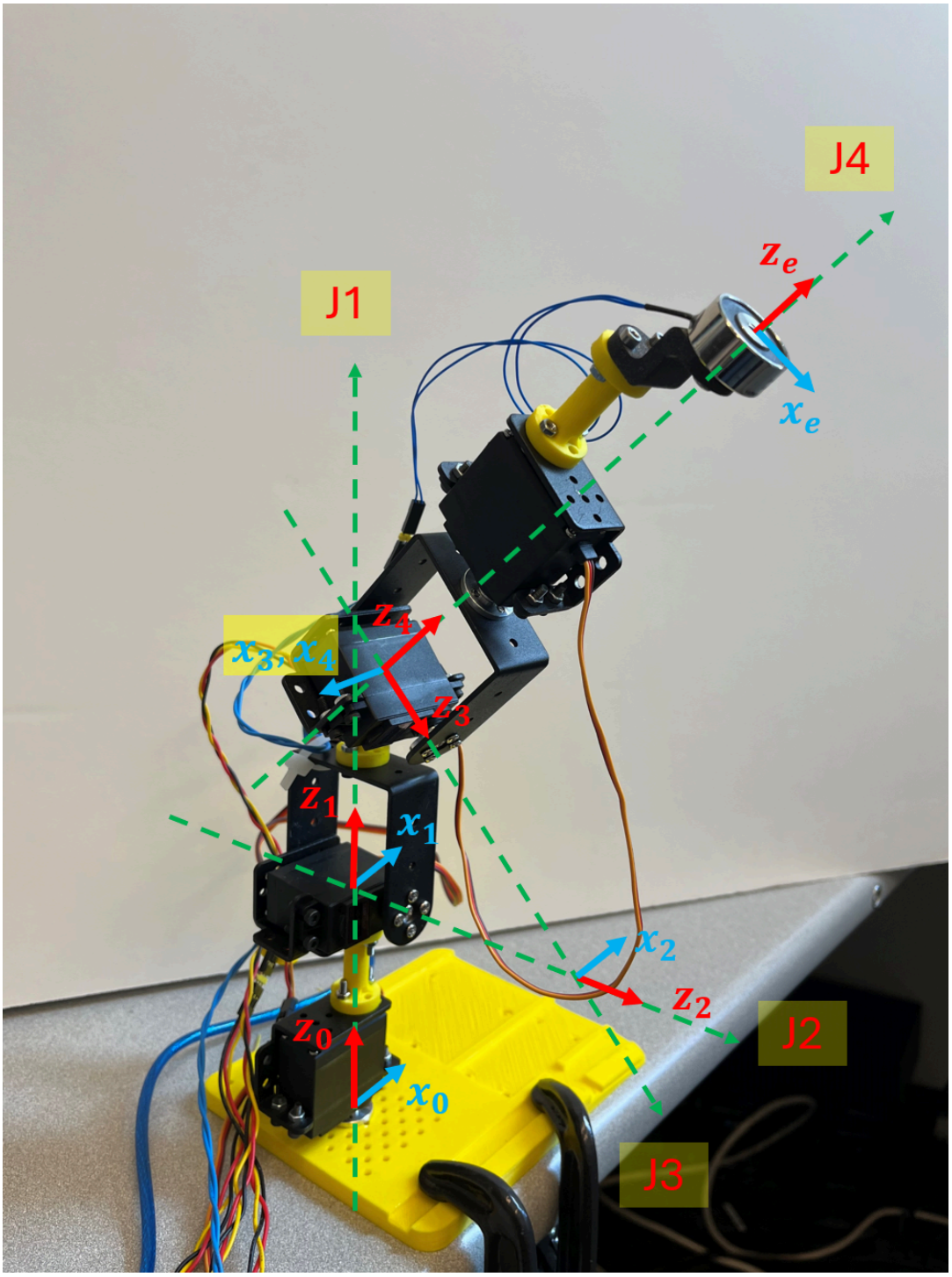


4. Frame Assignment and DH Parameters

4.1 Frame Assignment from Robot Arm Assembly

After you have designed your robot arm, to obtain DH parameters (*textbook 3.4, slides week 3*), we need to assign frames to the robot arm links (*textbook 3.2, slides week 3*). It is worth noting that technically frames are assigned to the links of the robot arms, but the frame assignment is determined by the joint axes. The choice of different links can change the direction and location of joint axes and thus indirectly determine the frame assignment, and further to DH parameters.

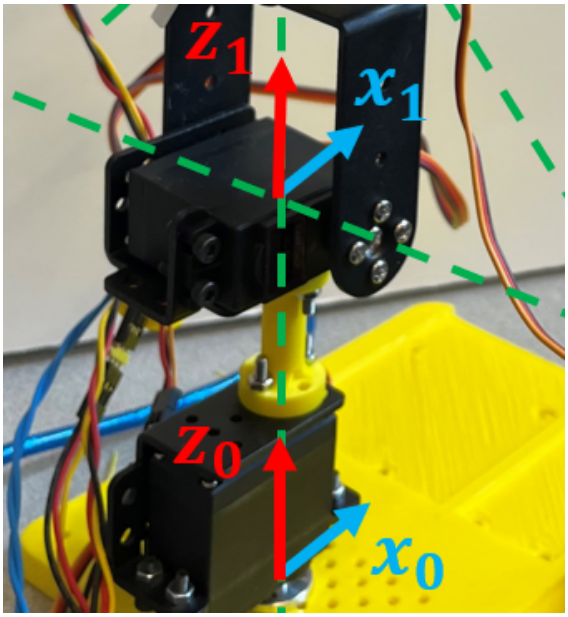
The example of frame assignment is shown in the following figure.



4.1.1 The choice of the base frame (frame 0)

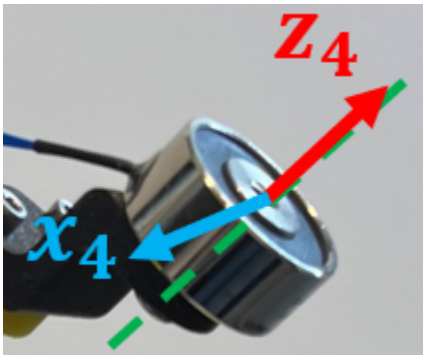
The base frame does not move with any joint rotations and can serve as the reference point for all subsequent calculations. The choice of origin point and z-axis of frame 0 can usually be arbitrary, and x axis will be determined by common normal to joint 1, as

introduced later on. However, for easy computing and use of DH parameter tables, it is recommended to choose in such a way that its z-axis aligns with the axis of joint 1. And x0 can be arbitrarily chosen to align with workspace edges to better represent tasks. In this project, x0 can be assigned as the arrow shown on the robot arm base, as shown in the following figure.



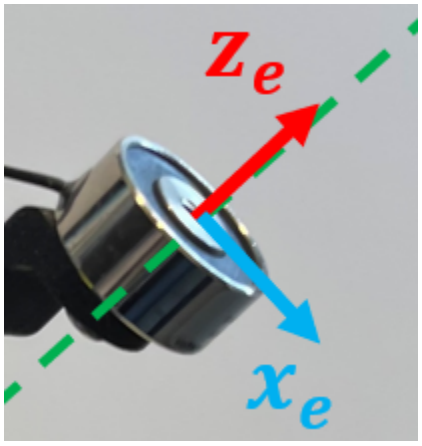
4.1.2 The choice of the last frame (frame e)

The last frame of the 4-joint robot arm can generally be frame 4, which is aligned to the joint 4 axis. For frame 4, z4 is naturally assigned on the joint 4 axis. Since joint 4 is the last joint in our design, there is no common normal to the next joint axis and thus x4 is not determined. In our application, we can choose the origin of frame 4 at the tip of the robot arm and the direction of x4 is in parallel with x3 so that the joint angle θ4 has no offset. As the following figure shows. This choice of frame 4 is beneficial because the location and orientation of frame 4 also represent the location and orientation of the end-effector of the robot.



However, in some other cases, since frame 4 must be on the joint 4 axis, it may not be able to represent the pose of the end-effector. In this case, we can add one more frame,

frame e to represent the end-effector, as the following figure shows. Please note that this frame actually has no corresponding joint so the DH parameters will be all fixed for this frame. This method is usually preferred when the frame on the last joint cannot well represent the end effector.



4.1.3 The assignment of the rest of the frames

Once the base frame and the last frame are chosen, we can assign frames to the rest of the links / joints (*textbook 3.2.3, slides week 3 forward kinematics*). There are several key points to correctly assign frames:

- The assignment of frame N is only determined by joint axis N and N+1. The assignment of frame N has nothing to do with frame N-1.
- The z-axis of frame N should be on the axis of joint N, and preferably in the same direction as the positive rotation of joint N.
- The x-axis of frame N should be on the common normal between the joint N axis and joint N+1 axis, and point to the joint N+1 axis. In some special cases, i.e. joint N and N+1 are coincide, parallel, or intersect, the assignment of the x-axis can also be special (details in *textbook 3.2.3, slides week 3 page 11*).

4.2 Derive DH Parameters from Assgined Frames

Once the frames are assigned, we can then derive the DH parameters of the robot arm (*textbook 3.4, slides week 3*). The table of the DH parameters of our robot arm should look like:

Frame (n)	α_{n-1}	$\mathbf{a}_{n-1}$	$\mathbf{d}_n$	θ_n
1	α_0	a_0	d_1	θ_1
2	α_1	a_1	d_2	θ_2

3	α_2	a_2	d_3	θ_3
4	α_3	a_3	d_4	θ_4
e	α_4	a_4	d_5	θ_5

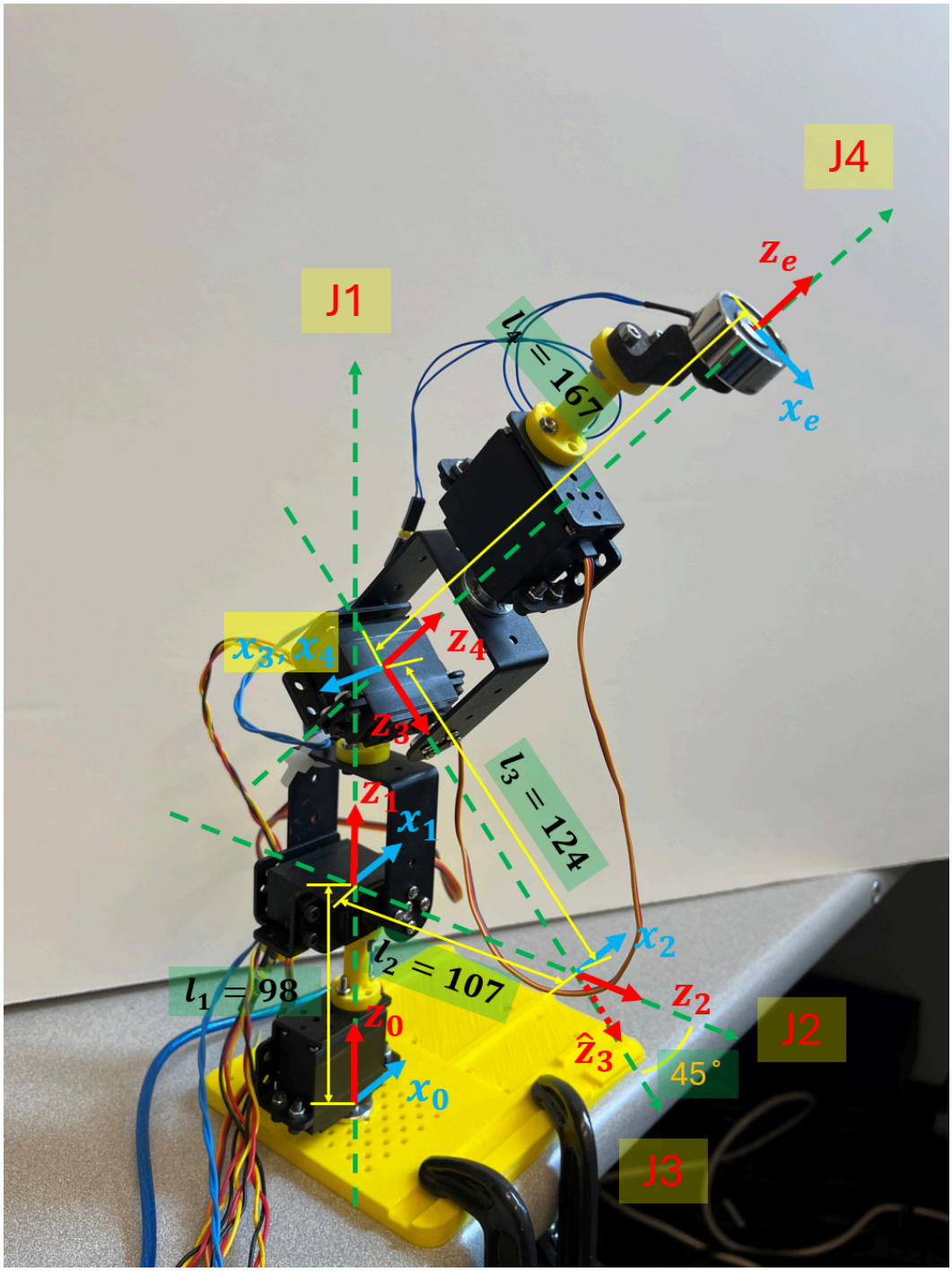
Since all joints of our robot arm are rotational, θ_1 - θ_4 are variable and the rest parameters are constant (θ_5 is constant too, since the end-effector frame is chosen to better represent the pose of the end-effector which has no corresponding joint). If we assume that joint 3 is translational, then θ_3 will be constant and d_3 will be variable.

In one row of the DH parameters, α and a have subscripts of n-1 while d and θ have subscripts of n, because the subscripts are determined by the definition of these parameters. This can be confusing. However, when we want to derive the DH parameters, it may be easier to understand that the DH parameters for frame 1 is the DH parameters that can move and rotate frame 0 to frame 1.
For each row of the DH table:
The row of frame 1 means that this row of DH parameters represents the transformation from frame 0 -> frame 1, and so on and so forth.

- α_{n-1} Is the rotation angle from z0 to z1 according to x0
- a_{n-1} is the distance from z0 to z1 along the common normal, according to x0
- d_n is the distance from x0 to x1, according to z1
- θ_n is the rotation angle from x0 to x1, according to z1

The DH parameters of the example robot arm are shown in the following table

Frame (n)	α_{n-1} **(deg)**	a_{n-1} **(mm)**	d_n **(mm)**	θ_n **(deg)**
1	0	0	98	θ_1
2	90	0	107	θ_2
3	45	0	-124	θ_3+180
4	90	0	0	θ_4
e	0	0	168	90



4.3 Find DH Parameters from 2 Points of Each Axis (Optional)

As introduced before and in course materials (*textbook 3.2 - 3.4, slides week 3*), 1 row of DH parameters represents the relationship between 2 joint axes. And a joint axis can be represented by 2 points respecting the positive direction. Thus, in theory, after choosing the base frame and the end-effector frame, the DH parameters can be

determined by giving 2 points of each joint axis. With 2 points representing each joint axis, we can find the common normals and assign frames as introduced before. Then we can derive the DH parameters from the assigned frames, joint axes, and common normals.

It is convenient to find the location of points on axes in CAD assembly. Before you find the point locations, it is recommended to make sure the assembly is in the zero position (all joint/motor rotations are 0), this is to enable that the result can also provide accurate joint offsets. Of course you can also manually find joint offsets by inspecting the frame assignment or using joint control simulation in the next chapter. With the joint points, you can then modify the code with your arm design.

```
58  # 2 points for each joint axis, respecting the positive direction
59  joint_points = np.array([[[10.5, 6, 0],[10.5, 6, -49.3]],
60                           [[6.71573, -21.9439, -97.8000],[14.0967, 32.5586, -97.8000]],
61                           [[8.94996, 5.66607, -205.257],[14.6142, 44.1449, -166.369]],
62                           [[15.8664, 60.3866, -221.516],[25.3533, 142.801, -304.444]],
63                          ])
64
65  # The 'x_ax' and 'z_ax' should be unit vectors representing the direction of axes
66  frame_0 = {'org': np.array([10.5, 6, 0]),
67             'x_ax': np.array([1, 0, 0]),
68             'z_ax': np.array([0, 0, -1])}
69  frame_e = {'org': np.array([25.3533, 142.801, -304.444]),
70             'x_ax': (np.array([26.5847, 151.167, -295.989]) - np.array([25.3533, 142.801, -304.444])) / np.linalg.norm(np.array([
71             'z_ax': (np.array([25.3533, 142.801, -304.444]) - np.array([15.8664, 60.3866, -221.516])) / np.linalg.norm(np.array([
72
```

Joint offset: The rule of frame assignment introduced before has its own ‘preferred’ zero pose, while it may not be consistent with the actual zero joint/motor positions of our actual robot. In this case, though joint positions (rotation θ or translation d) are variables, we may need to add a constant to the actual motor/joint rotation angle.

The following result is the automatically found DH parameters of the example robot. You can also verify your manually derived DH parameters.

```
----------------------------------------------------
Joint 0 to Joint 1
Axis relation: Coincide
Common normal:
{'p_start': array([10.50, 6.00, 0.00]), 'dir': array([1.00, 0.00, 0.00]), 'length': 0}
Frame 0:
{'org': array([10.50, 6.00, 0.00]), 'x_ax': array([1.00, 0.00, 0.00]), 'z_ax': array([0.00, 0.00, -1.00])}
----------------------------------------
----------------------------------------------------
Joint 1 to Joint 2
Axis relation: Intersect
Common normal:
{'p_start': array([10.50, 6.00, -97.80]), 'dir': array([0.99, -0.13, 0.00]), 'length': 0}
Frame 1:
{'org': array([10.50, 6.00, -97.80]), 'x_ax': array([0.99, -0.13, 0.00]), 'z_ax': array([0.00, 0.00, -1.00])}
----------------------------------------
----------------------------------------------------
Joint 2 to Joint 3
Axis relation: Intersect
Common normal:
{'p_start': array([24.85, 111.96, -97.80]), 'dir': array([0.99, -0.13, -0.01]), 'length': 0}
Frame 2:
{'org': array([24.85, 111.96, -97.80]), 'x_ax': array([0.99, -0.13, -0.01]), 'z_ax': array([0.13, 0.99, 0.00])}
----------------------------------------
----------------------------------------------------
Joint 3 to Joint 4
Axis relation: Intersect
Common normal:
{'p_start': array([11.78, 24.91, -185.81]), 'dir': array([-0.99, 0.13, 0.02]), 'length': 0}
Frame 3:
{'org': array([11.78, 24.91, -185.81]), 'x_ax': array([-0.99, 0.13, 0.02]), 'z_ax': array([0.10, 0.70, 0.71])}
----------------------------------------
----------------------------------------------------
Joint 4 to Joint 5
Axis relation: Coincide
Common normal:
{'p_start': array([11.78, 24.91, -185.81]), 'dir': array([-0.99, 0.13, 0.02]), 'length': 0}
Frame 4:
{'org': array([11.78, 24.91, -185.81]), 'x_ax': array([-0.99, 0.13, 0.02]), 'z_ax': array([0.08, 0.70, -0.71])}
----------------------------------------
||||||||||||||||||||||||||||||||||||||||
DH Parameters (alpha, a, d, theta):
[[0.000 0.000 97.800 7.712]
 [90.000 0.000 106.929 0.662]
 [45.000 0.000 -124.483 -179.658]
 [90.000 0.000 0.000 0.000]
 [0.000 0.000 167.800 89.999]]
```

5. Forward Kinematics, Workspace, and Robot State

5.1 Applying Forward Kinematics and Check DH Parameters

5.1.1 DH-based simulation and verification of DH parameter

Once we have the DH table, we can simulate the robot arm kinematics. To enable the real robot and simulation at the same time, change the following lines:

```
26    mod_real_robot = True
27    mod_dh_sim = True
```

You can put in the DH parameters of your robot arm. DH Parameters are given in order of [a, alpha, d, theta] for each joint, angles should be given in degree. If the joint is rotational, then the 4th entry will be variable and the number given here will be treated

as offset of that joint. This is the same if the joint is translational so that d is variable. Since we have one more frame assigned to the end-effector but with no joint, we define the joint type as ‘f’ (fixed). The rest of the joints are all ‘r’ (rotational). If you have translational joints, it should be defined as ‘t’ (translational). When we define the initial and home position of the joints, in our example, we only have 4 entries since we actually have 4 joints.

```
32    joint_type = ['r', 'r', 'r', 'r', 'f']
33    joint_pos_init = [0, 0, 0, 0]
34    joint_pos_home = [0, 0, 0, 0]  # this is the home position,
```

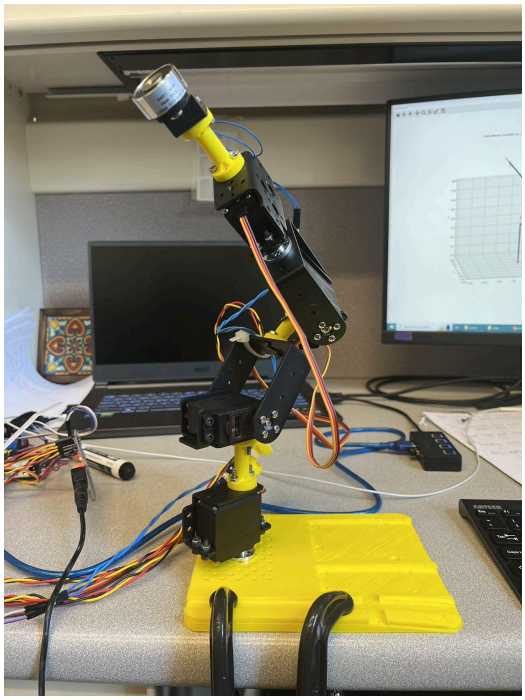
```
43    dh_parameters = [[0.0, 0.0, 98, 0.0],
44     [90.0, 0.0, 107, 0.0],
45     [45.0, 0.0, -124, 180.0],
46     [90.0, 0.0, 0.0, 0.0],
47     [0.0, 0.0, 168, 90]]
```

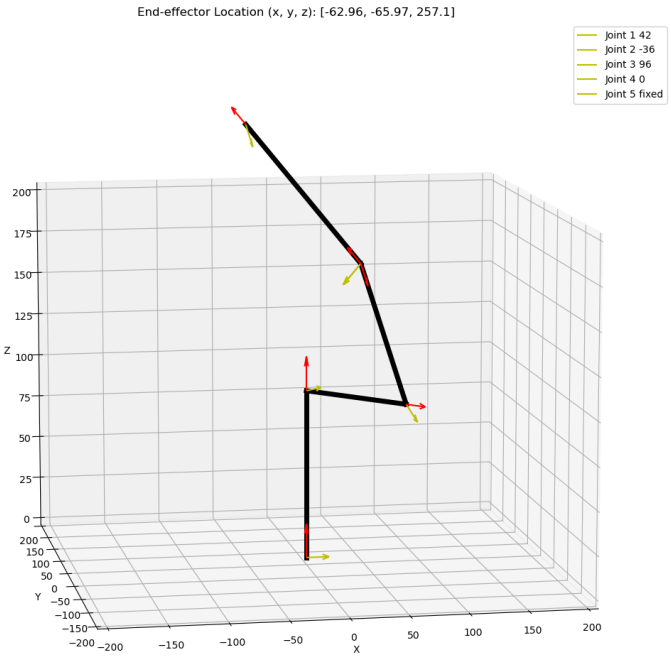
You can also add more joints similarly. When new joints are added, the keyboard control should also be extended for new joints, you can do it by adding a similar code as follows.

```
elif input_key == '1':
    print_no_newline(" Moving: Joint 1 +++         ")
    joint_positions[0] += joint_speed[0]
    command = True


elif input_key == 'q':
    print_no_newline(" Moving: Joint 1 ---         ")
    joint_positions[0] -= joint_speed[0]
    command = True
```

By defult, the velocity of real robot and simulation are set as the same, thus, we can verify the DH parameter by applying same control on both the simulation and real robot. We should be able to observe same end-effector movement if everything is good (the movement of links can be different since DH method assigns virtual frames that can be outside the real robot arm. For the example robot arm, we can find that frame 2 is assigned at the intersection point of axes of joints 2 and 3, and is outside the robot arm, which caused the sharp corner link in the simulation, as shown in the following figure). If the simulation and real robot result in different poses and movements, it may indicate that the DH parameters we derived in the previous section are wrong. However, we should also pay attention to the joint position offsets caused by motor 0 positions.





Assignment: verify your DH parameters and show a screenshot of the simulation and a photo of the real robot with a similar pose.

5.1.2 Symbolic forward kinematics

To obtain symbolic forward kinematics and prepare for closed-form inverse kinematics solution, we can get the transformation from the robot base to the end-effector

$$ {}^{0}_{e}T = {}^{0}_{1}T \; {}^{1}_{2}T \; {}^{2}_{3}T \; {}^{3}_{4}T \; {}^{4}_{e}T $$

Where ${}^{0}_{1}T$ is determined by row 1 of the DH table, so on and so forth.

Recall that:

$$ {}^{N}_{N+1}T = \begin{bmatrix} c\theta_{N+1} & -s\theta_{N+1} & 0 & a_N \\ s\theta_{N+1}c\alpha_N & c\theta_{N+1}c\alpha_N & -s\alpha_N & -s\alpha_N d_{N+1} \\ s\theta_{N+1}s\alpha_N & c\theta_{N+1}s\alpha_N & c\alpha_N & c\alpha_N d_{N+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Assignment: Now we can compute the forward kinematic in symbolic form. Please keep the symbolic forward kinematics and it will be used in finding the inverse kinematics solution in the later sections.

The forward kinematics of the example robot arm is:

$$\begin{bmatrix} 1.0C_4 \cdot (0.707C_1S_2 + 0.707S_1) - 1.0S_4 (C_1C_2C_3 - 0.707C_1S_2S_3 + 0.707S_1S_3) & -1.0C_4 (C_1C_2C_3 - 0.707C_1S_2S_3 + 0.707S_1S_3) - 1.0S_4 \cdot (0.707C_1S_2 + 0.707S_1) \\ 1.0C_4 (-0.707C_1 + 0.707S_1S_2) - 1.0S_4 (-0.707C_1S_3 + 1.0C_2C_3S_1 - 0.70703125S_1S_2S_3) & -1.0C_4 (-0.707C_1S_3 + 1.0C_2C_3S_1 - 0.70703125S_1S_2S_3) - 1.0S_4 (-0.707C_1 + 0.707S_1S_2) \\ -0.70703125C_2C_4 - 1.0S_4 \cdot (0.707C_2S_3 + 1.0C_3S_2) & 0.70703125C_2S_4 - 1.0C_4 \cdot (0.707C_2S_3 + 1.0C_3S_2) \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1.0C_1C_2S_3 + 0.707C_1C_3S_2 - 0.707C_3S_1 & 0.707C_1S_2l_3 + 1.0S_1l_2 + 0.707S_1l_3 + 1.0l_4 \cdot (1.0C_1C_2S_3 + 0.707C_1C_3S_2 - 0.707C_3S_1) \\ 0.707C_1C_3 + 1.0C_2S_1S_3 + 0.70703125C_3S_1S_2 & -1.0C_1l_2 - 0.707C_1l_3 + 0.70703125S_1S_2l_3 + 1.0l_4 \cdot (0.707C_1C_3 + 1.0C_2S_1S_3 + 0.70703125C_3S_1S_2) \\ -0.707C_2C_3 + 1.0S_2S_3 & -0.707C_2l_3 + 1.0l_1 + 1.0l_4 (-0.707C_2C_3 + 1.0S_2S_3) \\ 0 & 1 \end{bmatrix}$$

Although it looks ‘crazy’, we do not need to type these equations when coding, since we can code the transformation matrix for each row of the DH table and then dot multiply them together.

5.2 Robot Workspace

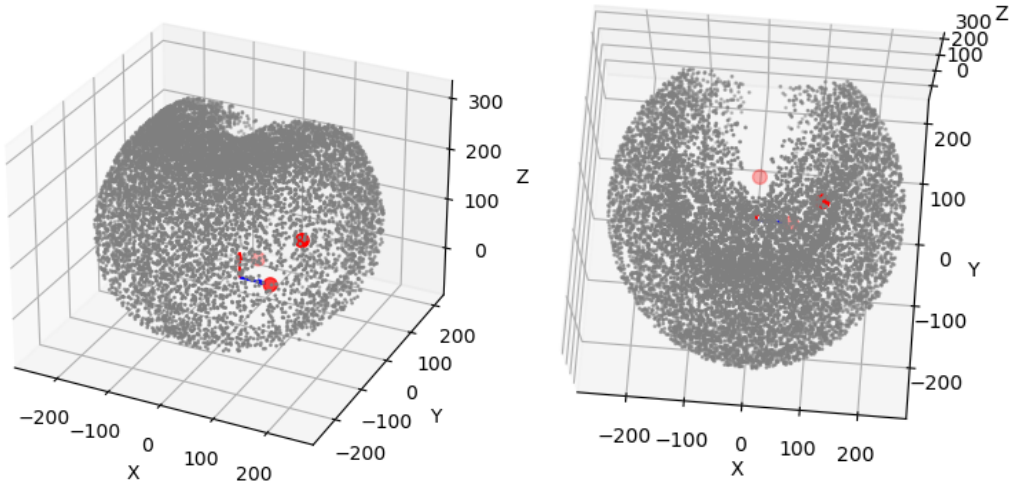
Before we finalize the design of the robot arm and work on cartesian-based control, it is beneficial to check if the robot workspace can satisfy the desired task. Conventionally, we can manually find the workspace based on the robot arm design (*textbook 4.2.1, slides week 4*). However, manually deriving workspace can be more difficult for more joints and non-zero joint axes twist (non-zero α).

Thus, here we use an approximate Monte Carlo (MC) method that can be easily applied to a robot arm with any number of joints, given the DH parameters. Given the DH parameters and joint limits of the arm, this method randomly samples a certain number of arm configurations and computes the end-effector pose. The workspace of the robot arm can be represented by sufficient samples.

We can write Monte Carlo simulation code, which takes inputs of the DH parameters and joint limits. You can also set several test points and the code can also tell whether these points are inside or outside the the robot workspace.

```
25  dh_parameters = [[0.0, 0.0, 98, 0.0],
26   [90.0, 0.0, 107, 0.0],
27   [45.0, 0.0, -124, 180.0],
28   [90.0, 0.0, 0.0, 0.0],
29   [0.0, 0.0, 168, 90]]
30
31  # define test points in the defult world frame
32  test_points = [[60, 0, 0],
33                 [0, 70, 0],
34                 [0, -210, 100]]
```

If everything works well, you should see a plot as followed, note the shape and test point locations can be different based on your arm design and choice of test points.



The simulation will also tell if the test points are inside or outside the simulated workspace. Please notice that the result can be inaccurate for marginal locations. This is because the algorithm basically takes the 50 nearest MC points to the test point and then computes the vector from the test points to the nearest MC points, then computes the angle between each 2 vectors, and finally, if the average angle is larger than 85 Deg, the test point is considered inside the workspace, otherwise outside. The result accuracy can be improved by increasing the number of overall MC points, the number of nearest MC points, and setting the threshold closer to 90 Deg.

```
Point:  [60, 0, 0]  is OUTSIDE workspace
Point:  [0, 70, 0]  is OUTSIDE workspace
Point:  [0, -210, 100]  is INSIDE workspace
```

5.3 Robot State (Optional)

Besides controlling API, it is also beneficial to have robot state feedback for higher-level robot control to better understand the condition of the robot arm. Generally, the robot state can contain information including but not limited to:

- Joint/motor position: this is important information to represent the pose of the robot
- End-effector pose: this can be derived from the joint positions using forward kinematics
- Run level: this state usually represents whether the robot is operational, busy, paused, or e-stoped.
- Joint and end-effector velocity: this information can be computed based on the history of joint and end-effector positions. However, it should be noted that if joint position measurements are noisy, filters should be applied.

There can also be other robot states such as motor torque, which is proportional to the motor current for DC motors. For the robot we are building, a joint position is provided.

6. Inverse Kinematics and Position Control in Cartesian Space

In the previous Chapter 5, we achieved joint-based control using a keyboard. In real practice, Cartesian-based control of the end-effector is commonly preferred. However, what we can actually control is still the motor rotation and joint position. Thus, we need to apply inverse kinematics, which takes the desired end-effector position and orientation as input, and outputs the joint positions of the robot arm that can reach this pose.

In general, inverse kinematics (IK) can be solved analytically or numerically. Numerical solution is generally applicable and solved by iterative methods such as Newton–Raphson method. Numerical solution is usually easier to implement but it can be computationally expensive due to the iterative solving procedure. On the other hand, analytical solution, also called closed-form solution, has better computational efficiency. However, closed-form solutions do not always exist, especially for 7 or more joints. Fortunately, for our 4-joint robot arms, close-form IK should exist.

6.1 Closed-form Solution of Inverse Kinematics

6.1.1 Manual Closed-form Solution

There are algebraic and geometric methods to solve closed-form IK (*textbook 4.1-4.6, slides week 4 page 25-67*). In the following example, we will use the algebraic method. In general, IK problem can be represented as the following equation:

$$\begin{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} & \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & 1 \end{bmatrix} = {}^{0}_{1}T \quad (\theta_1) {}^{1}_{2}T \quad (\theta_1) \dots {}^{n-1}_{n}T \quad (\theta_n)$$

The left side of the equation is the 4X4 homogenous transformation determined by the desired end-effector position so that all entries are known and constant. The right side is the symbolic forward kinematics transformation, in which the joint positions are unknown variables. Thus, each entry on the left and right can form an equation and our goal is to solve the joint positions.

It is worth noting that a homogenous transformation matrix, though has 4X4=16 entries, is actually 6 degrees of freedom (DoF). This is because the rotation matrix r11 - r33, are not linearly independent. Instead, it has only 3 DoF, which can be represented by Roll-Pitch-Yaw (RPY) angles or ZYX Euler angles (*textbook 2.5, slides week 1 page 67-69*). Thus, the homogeneous transformation is 6 DoF, 3 DoF on orientation (RPY), 3 DoF on position (x y z). In order to make both position and orientation of the end-effector have full degrees of freedom, a robot arm usually need 6 or more joints. For our 4-joint arm, we only have 4 DoF of the arm so that our arm cannot reach arbitrary positions and orientations (6 DoF) in the workspace. Thus, we can only control a maximum of 4 DoF out of the 6 DoF positions and orientations. For our robot arm specifically, because we choose the last joint (joint 4) to be a 'rolling' joint, this joint has no influence on the position of the end-effector. You can confirm this by checking the symbolic solution of the forward kinematics and you can find θ_4 is not present in the x, y, and z entries of the transformation matrix (there is no S_4 or C_4). To sum up, this means that for our robot arm, the x, y, and z position of the end-effector is determined by joint 1, 2, and 3. This is 3 DoF to 3 DoF and we can have finite solutions for most of the robot configurations within the workspace (we may have infinite solutions at singularity, which will be covered in Chapter 7). And then joint 4 can have an arbitrary rotation that gives extra dexterity on orientation.

$$\begin{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} & \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & 1 \end{bmatrix} = {}^{0}_{1}T \quad (\theta_1) {}^{1}_{2}T \quad (\theta_1) \ldots {}^{n-1}_{n}T \quad (\theta_n)$$

Thus, to solve the joint position of the end-effector, we should only use the desired position (x_d, y_d, z_d) to form equations.

6.1.2 IKBT: solving symbolic inverse kinematics with behavior tree

Solving inverse

IKBT is a package for symbolic solution of manipulator inverse kinematic equations. It is implemented with sympy (the python symbolic manipluation classes). The package is organized around a behavior tree encoding the higher level solution logic, and 'leaves' which implement specific solvers, manipulations, tests, and solution quality metrics.

You can download the IKBT package from IKBT GitHub.

6.2 Numerical Solution of Inverse Kinematics

6.3 Position Control in Cartesian Space

With the solution of inverse kinematics, we can now implement control of the end-effector in Cartesian space, which is based on our previous joint-level control but can be more intuitive for real-world tasks.

7. Jacobian Matrix, Singularity, and Velocity Control (Optional)

7.1 Derive Jacobian Matrix by Differentiation from Forward Kinematics

7.2 Singularity

7.3 Velocity Control

Appendix

Appendix 1 - Close-form inverse kinematics solution of the example robot arm

[Todo]: Need to type the solution instead of photos

$$l_2 S_1 + 0.7 l_3 S_1 - 0.7 l_4 C_3 S_1 + 0.7 l_3 S_2 C_1 + l_4 C_2 S_3 C_1 + 0.7 l_4 S_2 C_3 C_1 = X \quad ①$$

$$0.7 l_3 S_2 S_1 + l_4 C_2 S_3 S_1 + 0.7 l_4 S_2 C_3 S_1 + l_4 C_2 S_3 S_1 - l_2 C_1 - 0.7 l_3 C_1 + 0.7 l_4 C_3 C_1 = Y \quad ②$$

$$-0.7 l_3 C_2 + l_1 + l_4 S_3 S_2 - 0.7 l_4 C_3 C_2 = Z \quad ③$$

Rewrite ① and ②

$$(l_2 + 0.7 l_3 - 0.7 l_4 C_3) S_1 + (0.7 S_2 + l_4 C_2 S_3 + 0.7 l_4 S_2 C_3) C_1 = X \quad ①$$

$$(0.7 l_3 S_2 + l_4 C_2 S_3 + 0.7 l_4 S_2 C_3) S_1 - (l_2 + 0.7 l_3 - 0.7 l_4 C_3) C_1 = Y \quad ②$$

Let $k_1 = l_2 + 0.7 l_3 - 0.7 l_4 C_3$, $k_2 = 0.7 l_3 S_2 + l_4 C_2 S_3 + 0.7 l_4 S_2 C_3$

We have

$$k_1 S_1 + k_2 C_1 = X$$
$$k_2 S_1 - k_1 C_1 = y$$
$$\Rightarrow \begin{bmatrix} k_1 & k_2 \\ k_2 & -k_1 \end{bmatrix} \begin{bmatrix} S_1 \\ C_1 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} S_1 \\ C_1 \end{bmatrix} = \frac{1}{-k_1^2 - k_2^2} \begin{bmatrix} -k_1 & -k_2 \\ -k_2 & k_1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (-k_1^2 - k_2^2 \neq 0)$$

$$S_1 = \frac{-k_1 x - k_2 y}{-k_1^2 - k_2^2} \qquad C_1 = \frac{-k_2 x + k_1 y}{-k_1^2 - k_2^2}$$

$\theta_1 = \text{atan2}(S_1, C_1)$, however, there are θ_2 and θ_3 in k_1, k_2

DH	α	a	d	θ
	0	0	l_1	θ_1
	90	0	l_2	θ_2
	45	0	l_3	θ_3
	90	0	0	θ_4
	0	0	l_4	90.

(1)

We have : $k_1 S_1 + k_2 C_1 = X$

$k_2 S_1 - k_1 C_1 = y$

Take square and add up :

$k_1^2 S_1^2 + k_2^2 C_1^2 + 2k_1 k_2 S_1 C_1 + k_2^2 S_1^2 + k_1^2 C_1^2 - 2k_1 k_2 S_1 C_1 = X^2 + y^2$

$\Rightarrow k_1^2 + k_2^2 = X^2 + y^2$

$(l_2 + 0.7 l_3 - 0.7 l_4 C_3)^2 + (0.7 l_3 S_2 + l_4 C_2 S_3 + 0.7 l_4 S_2 C_3)^2 = X^2 + y^2$ ④

We have eq ③④ and rewrite into

$l_4 S_3 S_2 - (0.7 l_3 + 0.7 l_4 C_3) C_2 = Z - l_1$ ③

$(l_2 + 0.7 l_3 - 0.7 l_4 C_3)^2 + [(0.7 l_3 + 0.7 l_4 C_3) S_2 + l_4 S_3 C_2]^2 = X^2 + y^2$ ④

$③^2 + ④$, we have

$(l_2 + 0.7 l_3 - 0.7 l_4 C_3)^2 + l_4^2 S_3^2 + (0.7 l_3 + 0.7 l_4 C_3)^2 = X^2 + y^2 + (Z - l_1)^2$ ⑤

To cancel out S_3^2, we add $+ l_4^2 C_3^2 - l_4^2 C_3^2$, so that ⑤ becames.

$\frac{1}{2} l_4^2 C_3^2 - \sqrt{2} l_2 l_4 C_3 - l_3 l_4 C_3 + l_2^2 + \sqrt{2} l_2 l_3 + \frac{1}{2} l_3^2 + l_4^2 - l_4^2 C_3^2 \; \frac{1}{2} l_4^2 C_3^2 + l_3 l_4 C_3 + \frac{1}{2} l_3^2$

$= X^2 + y^2 + (Z - l_1)^2$

simplify, we have

$-\sqrt{2} l_2 l_4 C_3 = X^2 + y^2 + (Z - l_1)^2 - l_2^2 - l_3^2 - l_4^2 - \sqrt{2} l_2 l_3$

$$\theta_3 = \pm \arccos\left(\frac{X^2 + y^2 + (Z - l_1)^2 - l_2^2 - l_3^2 - l_4^2 - \sqrt{2} l_2 l_3}{-\sqrt{2} l_2 l_4}\right)$$

However, in this 2 solutions, one of the two may be a fake solution

We note $\theta_{3,1} = + \arccos(-)$

$\theta_{3,2} = - \arccos(-)$

But note that $\theta_{3,1}$ and $\theta_{3,2}$ result in same C_3. (2)

Now, θ_3 is solved and can be considered as known.

Rewrite ④:

~~$0.7l_3$~~

~~$(0.7l_3 + 0.7l_4 C_3)$~~

$$[(0.7l_3 + 0.7l_4 C_3)S_2 + l_4 S_3 C_2]^2 = x^2 + y^2 - (l_2 + 0.7l_3 - 0.7l_4 C_3)^2$$

$$\Rightarrow (0.7l_3 + 0.7l_4 C_3)S_2 + l_4 S_3 C_2 = \pm\sqrt{x^2 + y^2 - (l_2 + 0.7l_3 - 0.7l_4 C_3)^2}$$

Recall ③: $l_4 S_3 S_2 - (0.7l_3 + 0.7l_4 C_3)C_2 = z - l_1$

Let $k_3 = 0.7l_3 + 0.7l_4 C_3$, $k_4 = l_4 S_3$,

and $h_1 = \sqrt{x^2 + y^2 - (l_2 + 0.7l_3 - 0.7l_4 C_3)^2}$, $h_2 = z - l_1$.

Thus we have

$$k_3 S_2 + k_4 C_2 = \pm h_1$$
$$k_4 S_2 - k_3 C_2 = h_2$$

$$\Rightarrow \begin{bmatrix} k_3 & k_4 \\ k_4 & -k_3 \end{bmatrix}\begin{bmatrix} S_2 \\ C_2 \end{bmatrix} = \begin{bmatrix} \pm h_1 \\ h_2 \end{bmatrix}$$

$$\begin{bmatrix} S_2 \\ C_2 \end{bmatrix} = \frac{1}{-k_3^2 - k_4^2}\begin{bmatrix} -k_3 & -k_4 \\ -k_4 & k_3 \end{bmatrix}\begin{bmatrix} \pm h_1 \\ h_2 \end{bmatrix} \quad (-k_3^2 - k_4^2 \neq 0)$$

Take $+h_1$, so,

$$S_{2,1} = \frac{-k_3 h_1 - k_4 h_2}{-k_3^2 - k_4^2}, \quad C_{2,1} = \frac{-k_4 h_1 + k_3 h_2}{-k_3^2 - k_4^2}$$

Take $-h_1$, so

$$S_{2,2} = \frac{k_3 h_1 - k_4 h_2}{-k_3^2 - k_4^2}, \quad C_{2,2} = \frac{k_4 h_1 + k_3 h_2}{-k_3^2 - k_4^2}$$

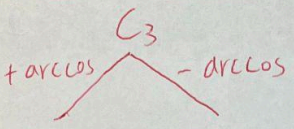
$\Rightarrow \theta_2 = \text{atan2}(S_2, C_2)$

Note that although h_1 contains θ_3 which has 2 solutions, but θ_3 is C_3 in h_1 so h_1 is unique and θ_2 has 2 solutions instead of 4.

With θ_3 and θ_2 solved, k_1 and k_2 is also known now

so θ_1 is solved.

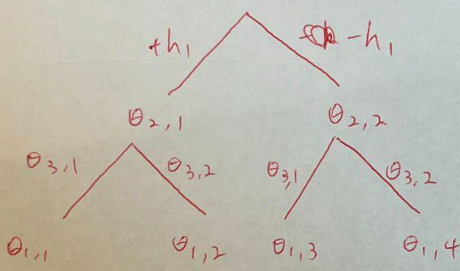
Multiple solution tree:



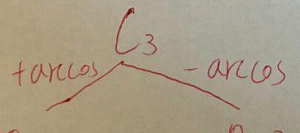
X this is wrong.

h_1 → because h_1 is $f(C_3)$, not $f(\theta_3)$

X



X



$h_1, k_{4,1}$ $h_1, k_{4,2}$ → h_1 is $f(C_3)$, not $f(\theta_3)$, but k_4 is $f(C_3)$