

## 2.12: Introduction to Robotics

### Lab 4: Motion of Robot Arms\*

Spring 2023

Assigned on: 2nd March 2023. Due by: 8th March 2023.

#### Instructions:

1. **When you are done with the lab, call a lab staff to do your check-off.**
2. **(Optional) Upload a video showing the robot arm in motion.**

## 1 Introduction

In this lab, you will go through the motion of robot arms, including:

1. Using forward kinematics (FK) to find the endpoint position given joint positions.
2. Using inverse kinematics (IK) to find joint positions given some target endpoint position (Sometimes called the “tool center point” (TCP)).
3. Scripting a complex trajectory.

**NOTE: Set your robot arm back to the home position before each run.**

## 2 Setting up the code

Please make sure your robotic arm is assembled (replace the disk with the 2nd link) and wired correctly (see assembly instructions from Lab 4), and then download Arduino code from Canvas or Github.

```
cd ~                               # make sure we are at home folder
git clone https://github.com/mit212/lab4_2023.git
```

---

\*

1. Version 1 - 2016: Peter Yu, Ryan Fish and Kamal Youcef-Toumi
2. Version 2 - 2017: Yingnan Cui, Kamal Youcef-Toumi, Steven Yeung and Abbas Shikari
3. Version 3 - 2019: Daniel J. Gonzalez
4. Version 4 - 2020: Jerry Ng, Steven Yeung, Rachel Hoffman, Kamal Youcef-Toumi
5. Version 5 - 2021: Hanjun Song
6. Version 6 - 2023: Ravi Tejawani and Kentaro Barhydt

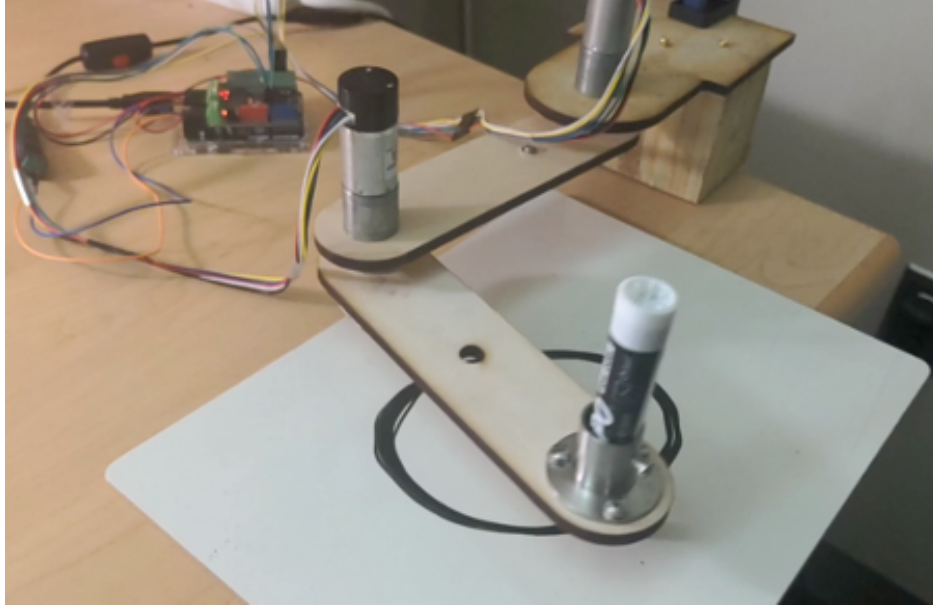


Figure 1: 2.12/2.120 two-link manipulator.

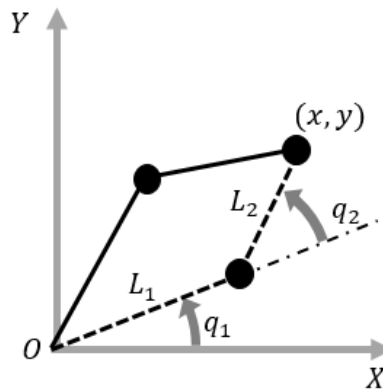


Figure 2: The kinematics of two-link manipulator. **Note that in the configuration shown,  $q_1$  and  $q_2$  have positive values.**

## 2.1 Forward Kinematics

In this section, you will be implementing the forward kinematics (FK) of the two-link manipulator. The two-link manipulator is shown in Figure 2, in which we define the length of two links ( $L_1, L_2$ ) in meters, and joint positions ( $q_1, q_2$ ) in rad, and target endpoint position  $(x, y)$  in meters. **Both  $q_1$  and  $q_2$  are defined as positive about the Z axis (out of the page).** The goal is to compute the endpoint position when the joint positions are given and we will compare the computed (true) endpoint position with the actual endpoint position.

Find the forward kinematics block in the main loop of the Arduino code and implement it following the instructions in the code. Once you are done, upload the code to the Arduino and see

if the robot moves to the expected endpoint position. You can see the actual endpoint position through the Serial Monitor. The forward kinematics equations are given below.

$$x = L_1 \cos q_1 + L_2 \cos (q_1 + q_2) \quad (1)$$

$$y = L_1 \sin q_1 + L_2 \sin (q_1 + q_2) \quad (2)$$

**Question 1** *How are the true endpoint position and actual endpoint position different? You can use a ruler to measure the actual endpoint position. Where does the difference come from?*

## 2.2 Inverse Kinematics

In this section, you will be implementing the inverse kinematics of the two-link manipulator. The goal is to compute the joint positions when a target endpoint position is given. We will compare the target endpoint position with the actual endpoint position. Find the inverse kinematics block in the main loop of the Arduino code and implement it following the instructions in the code. Once you are done, upload the code to the Arduino and see if the robot moves to the target endpoint position. The inverse kinematic solution follows that of [1]:

$$q_2 = \pm 2 \cdot \text{atan2}(\sqrt{(L_1 + L_2)^2 - (x^2 + y^2)}, \sqrt{(x^2 + y^2) - (L_1 - L_2)^2}), \quad (3)$$

and

$$q_1 = \text{atan2}(y, x) - \text{atan2}(L_2 \sin q_2, L_1 + L_2 \cos q_2), \quad (4)$$

where  $\text{atan2}(y, x)$  is a function that computes the arc tangent two variables  $y$  and  $x$ . It uses the signs of both arguments to determine the correct quadrant of the result. Note that there are two solutions for  $(q_1, q_2)$  that corresponds to elbow-up and elbow-down configurations. You may want to use the following functions in the Arduino code.

- `atan2()`, `cos()`, `sin()`, `sqrt()`
- PI: constant  $\pi$
- `sq(x)`:  $x^2$

**Question 2** *How are the target endpoint position and actual endpoint position different? Where does the difference come from?*

**Question 3** *Can you see the difference between the elbow-up and elbow-down solutions? Please take a picture of each solution.*

**Question 4** *What safety features are put in place to make sure that the robot can only be commanded to a position within the workspace?*

## 2.3 Trajectory Planning

In this section, you will make the robot arm move in a circle with a given radius and center. To do so, you are going to implement the circular path block in the main loop of the Arduino code. There will be comments in the code specifically indicating variables that need to be changed and assigned. The equation of a circle in polar coordinate will be useful to generate the circular path.

Fine-tune the following parameters to improve your circle:

- Controller gains for Motor 1 and Motor 2.
- Sampling period (variable name: `period_us`).
- Step size of waypoints (change the line `i++;` to `i += 2;` or some other integer increment).

In addition to looking at the circle drawn by the robot arm, you can use the Matlab script “SerialRead2.m” to capture and analyze the data (see Fig. (3)) so as to identify the issues. Make sure to uncomment `#define MATLAB_SERIAL_READ` and comment out `#define PRINT_DATA` before uploading the code to Arduino for this to work. Also make sure to change the serial port name in the Matlab script to match the Arduino port before running the script.

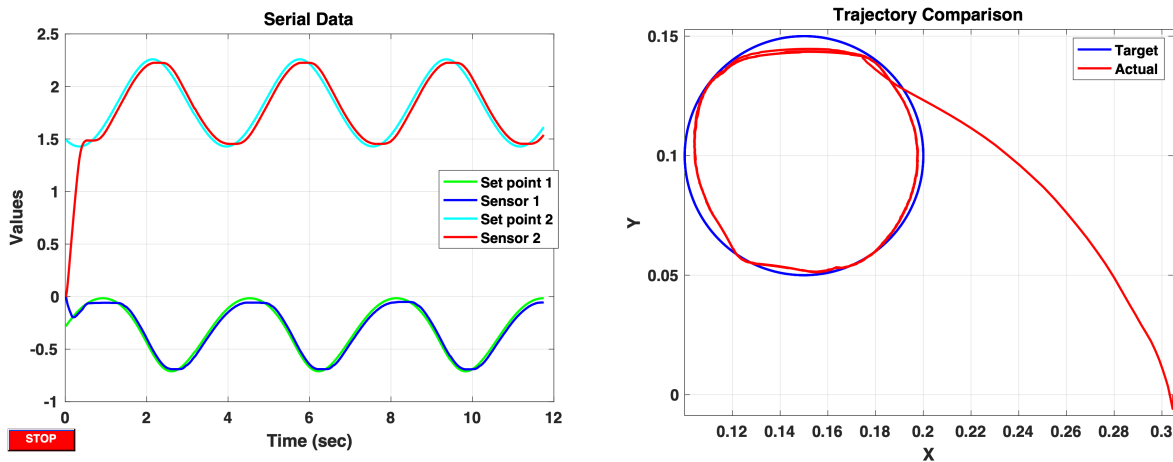


Figure 3: SerialRead.m outputs.

(Optional) Upload a video showing the robot arm in motion to this Dropbox.

If time permits, program a different trajectory for the robot to follow.

**Question 5** *Does the robot arm actually yield the commanded trajectory? Why/why not? Show a picture of the best circular path that your robot arm drew on the white board.*

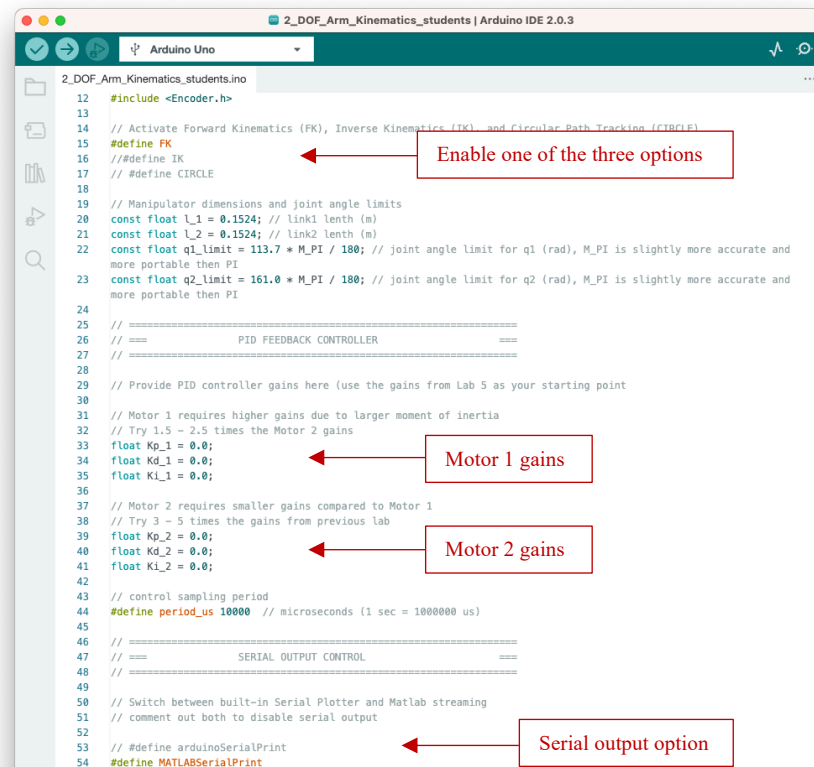
**Question 6** (Optional) *Can you make the robot follow a different shape of path? Show a picture of the path that your robot arm drew on the white board.*

## References

- [1] D. Gordon. Robotics: Forward and inverse kinematics. [Online]. Available: <http://www.slideshare.net/DamianGordon1/forward-kinematics>

## Appendix: Arduino template file

Controller gains and options:

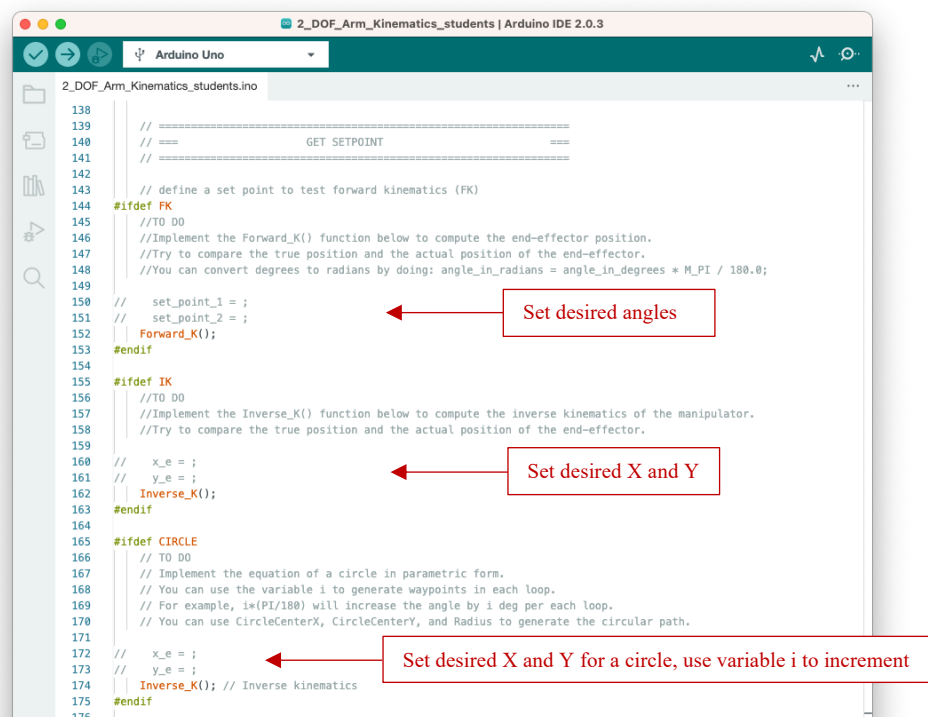


```
12 #include <Encoder.h>
13
14 // Activate Forward Kinematics (FK), Inverse Kinematics (IK) and Circular Path Tracking (CIRCLE)
15 #define FK
16 // #define IK
17 // #define CIRCLE
18
19 // Manipulator dimensions and joint angle limits
20 const float l1 = 0.1524; // link1 length (m)
21 const float l2 = 0.1524; // link2 length (m)
22 const float q1_limit = 113.7 * M_PI / 180; // joint angle limit for q1 (rad), M_PI is slightly more accurate and
    more portable than PI
23 const float q2_limit = 161.0 * M_PI / 180; // joint angle limit for q2 (rad), M_PI is slightly more accurate and
    more portable than PI
24
25 // =====
26 // == PID FEEDBACK CONTROLLER ==
27 // =====
28
29 // Provide PID controller gains here (use the gains from Lab 5 as your starting point)
30
31 // Motor 1 requires higher gains due to larger moment of inertia
32 // Try 1.5 - 2.5 times the Motor 2 gains
33 float Kp_1 = 0.0;
34 float Kd_1 = 0.0;
35 float Ki_1 = 0.0;
36
37 // Motor 2 requires smaller gains compared to Motor 1
38 // Try 3 - 5 times the gains from previous lab
39 float Kp_2 = 0.0;
40 float Kd_2 = 0.0;
41 float Ki_2 = 0.0;
42
43 // control sampling period
44 #define period_us 10000 // microseconds (1 sec = 1000000 us)
45
46 // =====
47 // == SERIAL OUTPUT CONTROL ==
48 // =====
49
50 // Switch between built-in Serial Plotter and Matlab streaming
51 // comment out both to disable serial output
52
53 // #define arduinoSerialPrint
54 #define MATLABSerialPrint
```

Annotations in the image:

- Enable one of the three options (points to lines 15-17)
- Motor 1 gains (points to lines 33-35)
- Motor 2 gains (points to lines 39-41)
- Serial output option (points to lines 53-54)

FK, IK and CIRCLE codes (to do):



```
138
139 // =====
140 // == GET SETPOINT ==
141 // =====
142
143 // define a set point to test forward kinematics (FK)
144 #ifdef FK
145 // TO DO
146 // Implement the Forward_K() function below to compute the end-effector position.
147 // Try to compare the true position and the actual position of the end-effector.
148 // You can convert degrees to radians by doing: angle_in_radians = angle_in_degrees * M_PI / 180.0;
149
150 // set_point_1 = ;
151 // set_point_2 = ;
152 // Forward_K();
153 #endif
154
155 #ifdef IK
156 // TO DO
157 // Implement the Inverse_K() function below to compute the inverse kinematics of the manipulator.
158 // Try to compare the true position and the actual position of the end-effector.
159
160 // x_e = ;
161 // y_e = ;
162 // Inverse_K();
163 #endif
164
165 #ifdef CIRCLE
166 // TO DO
167 // Implement the equation of a circle in parametric form.
168 // You can use the variable i to generate waypoints in each loop.
169 // For example, i*(PI/180) will increase the angle by i deg per each loop.
170 // You can use CircleCenterX, CircleCenterY, and Radius to generate the circular path.
171
172 // x_e = ;
173 // y_e = ;
174 // Inverse_K(); // Inverse kinematics
175 #endif
176
```

Annotations in the image:

- Set desired angles (points to lines 150-152)
- Set desired X and Y (points to lines 160-162)
- Set desired X and Y for a circle, use variable i to increment (points to lines 172-174)

Functions to implement FK and IK equations:

```
232 }
233
234 // ===== Forward Kinematics =====
235 // =====
236 //
237 void Forward_K()
238 {
239     //T0 D0
240     //Update x_e and y_e with the forward kinematics equations.
241     //You can use l_1, l_2 for the length of each link and set_point_1 and set_point_2 for the joint angles.
242
243     // x_e = ;
244     // y_e = ;
245 }
246
247 // ===== Inverse Kinematics =====
248 // =====
249 // =====
250 void Inverse_K()
251 {
252     //T0 D0
253     //In the lab2, we have learned how to implement inverse kinematics of the 2 DOF manipulator.
254     //You can use l_1, l_2 for the length of each link and x_e and y_e for the end-effector position.
255
256     q_2_ik = ; // change the sign of q_2 for another solution, here o
257     q_1_ik = ;
258
259     // position limit constraints (update set_point_1 and set_point_2 when they are within the limits)
260     if (abs(set_point_1) < q1_limit && abs(set_point_2) < q2_limit)
261     {
262         set_point_1 = q_1_ik;
263         set_point_2 = q_2_ik;
264     }
265     else
266     {
267         Serial.print("Joint limit reached!");
268     }
269 }
270
```

FK equations

IK equations