ME 4451 Lab 1 Handout
RRR Serial Robot Servo Control

Anirban Mazumdar, Bryan Blaise, Nikhil Nandikanti
(1 Session Lab)

Spring 2024

1 Objectives

1. Familiarize yourself with the Robotis Dynamixel servo motors and their capabilities.
2. Establish serial communication with the servos using Matlab.
3. Perform basic position control of the servos in Matlab.
4. Write a function to move the servos to a set of given angles.
5. Test the accuracy and precision of the serial RRR robot.

2 Prelab Assignment

- Read Lab 1 Handout.
- Create a script for Lab 1 tasks (Sections 6, 7 and 8), such that you can readily implement necessary commands for each task within Matlab. Please clearly comment your code. Your TA will check this off at the beginning of your lab section.

3 Coordinate System

The position (x_e, y_e) of the end-effector and the orientation of the end-effector (ϕ_e) are shown in Fig.1. Furthermore, each joint angle, θ_i, is defined positive in the counter-clockwise direction. The label a_i corresponds to the length of the i^{th} link.

4 Dynamixel AX-12 Servo

The Dynamixel AX-12 servos are powerful DC servo motors that are much more capable than standard hobby-type servo motors. For example, the Dynamixel servos have an internal 1024-count encoder. This is superior to the standard potentiometers that are used for hobby-type

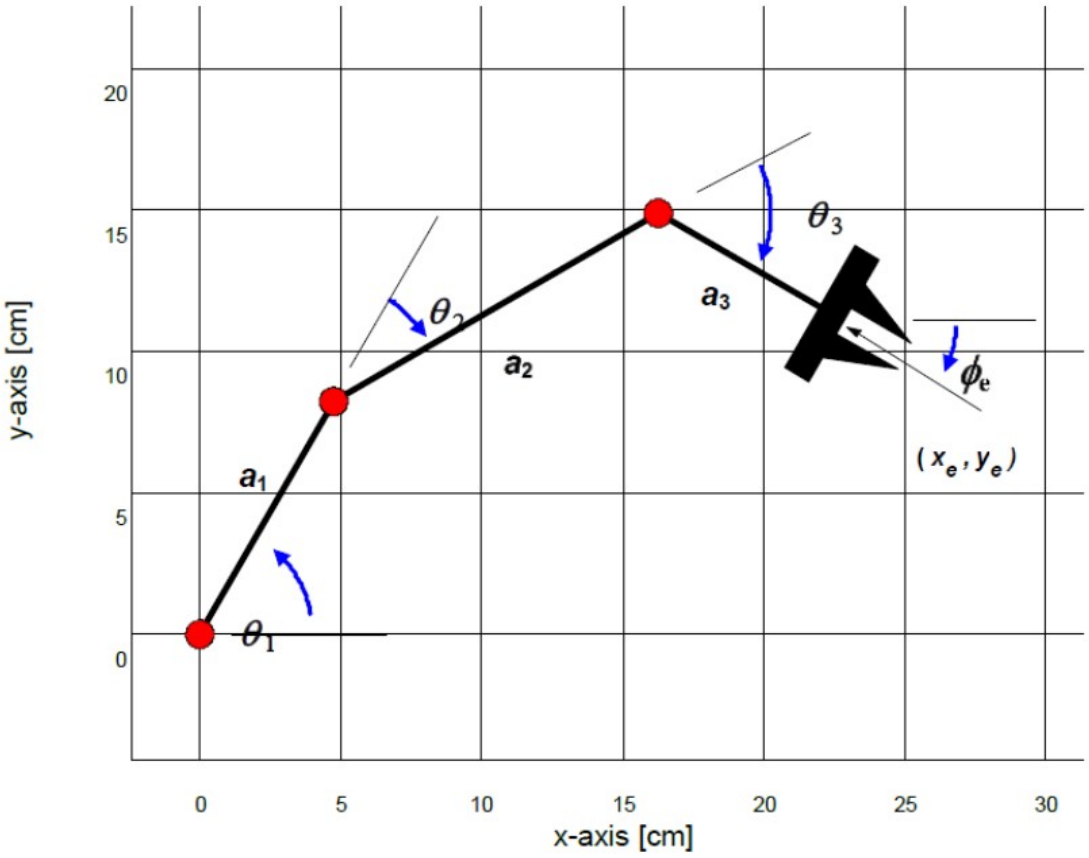

Figure 1: Coordinate frame for the RRR serial robot.

servos. Instantaneous position, velocity, load, and other parameters can be read from each servo's memory. A 3-wire serial bus connects all servos in a daisy chain, which terminates in the USB2Dynamixel serial adapter. Power is provided by a DC power adapter and is spliced into the serial bus via a small electrical board. A photo of the servo motor is provided in Fig. 2.

Figure 2: The AX-12 DC servo motor

5 Dynamixel Interface for Matlab

The Dynamixel interface consists of a set of Matlab functions that allow you to write and read parameters to individual servos. This lab will familiarize you with the main servo functions below, and subsequent labs will introduce the rest.

dxl_SerialOpen(port)
Description: Establishes a serial connection with the USB2Dynamixel serial adapter, and defines angle and torque limits to prevent robot damage. Requires the COM port name as a string. This is typically COM3 or COM4. Check Windows Device Manager if port is unknown.
dxl_SerialOpen('COM3');

dxl_GetPos(id)
Description: Returns the present position of servo *id* in degrees, measured as shown in Fig.1.
dxl_GetPos(2) (Get the position of servo 2)

dxl_SetPos(id, GoalPos)
Description: Sets the desired position of servo *id* to angle *GoalPos* in degrees. *GoalPos* must be between ±150 degrees. The servo will immediately attempt to move to this position.
dxl_SetPos(3, 60) (Move servo 3 to 60 degrees)

dxl_SetVel(id, GoalVel)
Description: Sets the move speed of the servo *id* to *GoalVel* in degrees/s. This does not cause the servo to move, but instructs the servo to use this speed for all subsequent move commands.
dxl_SetVel(1, 10) (Set the speed of servo 1 to 10 deg/s)

dxl_Grab
Description: Commands the end-effector gripper to close to a predetermined position intended for the robot pen holder. If you try this command with other objects, you may overload the servo. This function assumes that the gripper is controlled by servo 4.
dxl_Grab (Closes the gripper)

dxl_Release
Description: Releases the end-effector gripper. This function assumes the gripper is controlled by servo 4.
dxl_Release (Opens the gripper)

dxl_TorqueEnable(id)
Description: Enables torque application for the desired servo, and the servo will now hold its current position.
dxl_TorqueEnable(254) (Enables torque for all servos)

dxl_TorqueDisable(id)
Description: Disables torque application for the desired servo, allowing the servos to be moved by hand. Servos will also cease to maintain their positions.
dxl_TorqueDisable(254) (Disable torque for all servos)

dxl_SerialClose
Description: Terminates the serial connection to the USB2Dynamixel
dxl_SerialClose (Closes the serial connection)

6 Instructions and Tasks for This Lab

For the first part of this lab you will be using the basic Dynamixel interface functions described above to get a feel for using the RRR serial robot. This will include setting positions and speeds, and using the gripper. You will then write a function to command the entire robot to a given set of angles and you will test the robot for movement accuracy and precision.

6.1 Basic Interface Functions

The wiring for each robot should already be set up. First verify that the USB2Dynamixel is connected to the computer, and that the robot and power adapter are connected as well. **Make sure that the robot has plenty of space to move around in, and stay clear of it while the robot is in motion!**

6.1.1 Basic Interface Functions

Use these basic interface functions, one at a time, in the Matlab command window.

- dxl_SerialOpen('COM3'); (Open serial port, change COM if needed)
- dxl_SetVel(254,50); (Set all velocities to 50 deg/s)
- dxl_SetPos(3,15); (Move joint 3 to 15 degrees)
- dxl_SetPos(2,-20); (Move joint 2 to -20 degrees)

6.1.2 Using the Broacast ID, 254

Try changing the move speed and using the broadcast ID, 254.

- dxl_SetPos(254,0); (Move all joints to 0 degrees)

The broadcast ID 254 is used to command all servos simultaneously. Be careful when using it though. It will affect the end-effector gripper servo as well.

6.1.3 Error Handling

Sometimes you will give a command that throws and error on the servo. The Dynamixel interface is designed to notify you when this happens. Try the command below. The message on the command window should notify you that you have a Data Range error. This is because the servos have hard rotation limits of $\pm 150 degrees$. Therefore, a 160 degree command is out of the valid data range.

- dxl_SetPos(3,160); (Try to move servo 3 to 160 degrees)

Now try another command (below). This time you should see an Angle Limit error, but the servo should still have moved. This is because there are soft limits on the rotation angle, at 137 and -142 degrees to prevent link collisions for this specific type of robot. These are defined within *dxl_SerialOpen*. Thus the servo rotated as much as the soft limit allowed, and then stopped and notified you of what happened.

- dxl_SetPos(3,140); (Try to move servo 3 to 140 degrees)

6.1.4 Determine Safe Limits for Servo Motion

Determine some safe limits for servo motion using the *dxl_GetPos* function. While you just saw how the servos will barely prevent collisions and throw an error, let's find some safer limits:

- dxl_SetPos(254,0) (Return all servos to center position)
- dxl_TorqueDisable(254) (Disable motor torque to all servos)
- Now rotate link 3 by hand. Rotate it clockwise until you reach a point a little bit ahead of the collision point, and use the following command to get the position of the servo.

- dxl_GetPos(3); (Return the position of servo 3)
- Repeat the process for the counter-clockwise direction, and record what you have chosen in the table below.

	Clockwise	Counterclockwise
Angle 1 Limit (deg)		
Angle 2 Limit (deg)		
Angle 3 Limit (deg)		

Continue to play with the Dynamixel interface commands until you are comfortable with them, and then move onto the next section.

7 RRR Robot Control

Now that you understand how to control individual servos, complete the function called RRR_go_angles(theta) to move the entire arm to a given set of angles. The input, *theta*, should be an array of the desired join angles *[theta1, theta2, theta3]*. This function contains code to impose angle constraints on each servo, with a warning message to display if they are exceeded. Use the angle limits you previously found to complete that portion of the RRR_go_angles function.

- Try your new function with $theta = [0, 0, 0]$. The robot sould be fully extended and centered.
- Next try $theta = [70, -85, 30]$. Take note of the end-effector position.
- Now use $theta = [-15, 85, -55]$. What do you notice about the end-effector position, relative to the previous command?

8 Accuracy and Precision

In this final section you will test the accuracy and precision of the end-effector position. These metrics help determine how useful the robot can be for real tasks. To do this, you will write a script that starts with the end-effector at one central point, and then moves to another point somewhere else before attempting to return to the starting point. Repeating this several times with different points will show the precision of the RRR serial robot. in order to see these effects you will first need to give the robot a pen to draw with.

- Open the gripper with *dxl_Release*.
- Place the pen/pen holder into the gripper and use *dxl_Grab*.
- In the command window, open the serial connection.
- Move the robot to $theta = [102, -68, -74]$ using the *RRR_go_angles* you just wrote.

- Move to $theta = [30, -25, -45]$.

Measure the distance between these points using a ruler. The ideal distance is $14.6cm$. How far off is the distance you measured? What is the percent error? Now write a script that does the following:

- Opens the serial connection.
- Sets the move speed for all motors to $100deg/s$.
- Move the robot to angles $theta = [59, -40, -68]$ using your *RRR_go_angles*. This is the starting point.
- Move to each of the following sets of angles. **Return to the starting position in between each one**.
  $[81, -50, -45]$
  $[27, -26, -32]$
  $[24, -47, -45]$
  $[45, -67, -74]$
  $[76, -66, -95]$
- Use a *pause(2);* command after every move command to give the robot time to complete the move.
- Close the serial connection.

You should see that the robot does not return to the exact starting position each time. Draw a circle that encloses all of the starting points, and measure the diameter of this circle. What does this tell you about the precision of this robot? When you have completed all of the sections of this lab, close the serial connection and disconnect the USB2Dynamixel and the power adapter from the lab setup.