2.12: Introduction to Robotics
Project 2: Manipulation Kinematics and Control
Lab #4

1. Introduction

This lab is meant to introduce you to the Dynamixel servos provided in 2.12 as well as how to use kinematics to control them in a useful way.

Similar to hobby servos you may have used in the past, the Dynamixels are servomotors with position control, but they can also run in torque control mode as well as adjust a number of other control parameters. Unlike the hobby motors you have likely used in the past, the Dynamixels have well-tuned and very fast PID control to move them to the commanded position as well as many other interfacing parameters. 2.12 provides two models of Dynamixel servos: the MX-64, which is lighter, and the MX-106, which is higher torque. They run from a 12V supply, and they are controlled with a digital protocol that can be produced from the USB2Dynamixel dongle attached to the computers in lab. In this lab, commands will be sent through the dongle to the Dynamixels through Matlab, via serial communication, leveraging a specific API we have installed for you.

2. Dynamixel Setup

The Dynamixel Wizard is useful for debugging and testing Dynamixel servos. From the start menu, open the Dynamixel Wizard [you can hit the Windows key and search for it if it's not visible on shortcut]. From the drop-down menu in the top left, select the port to which your USB2Dynamixel dongle is connected, which should be COM4 for most of the computers on the robots in lab. Click the "Open Port" icon to the right of this menu to connect to the dongle. These features are indicated in Figure 1.

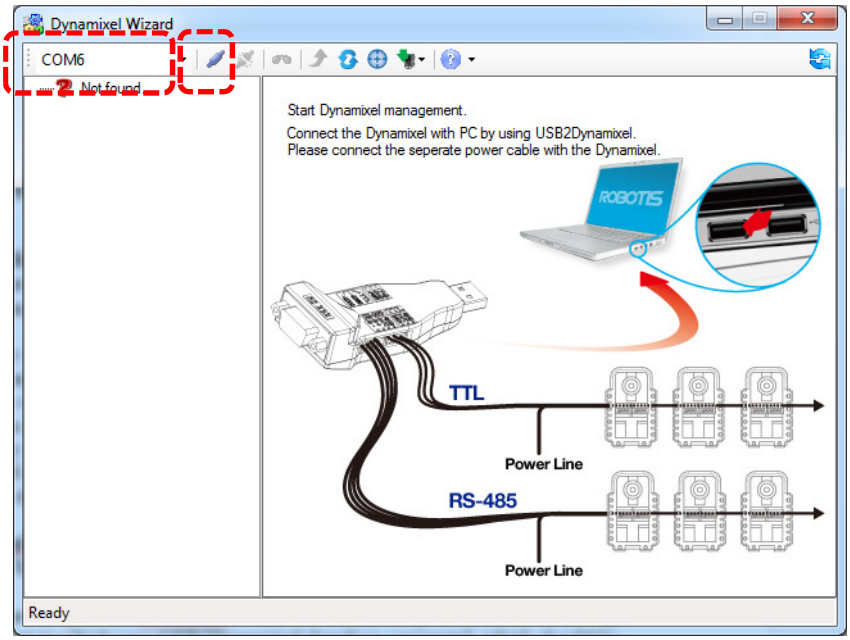

Figure 1: Dynamixel Wizard splash screen with relevant buttons indicated.

Once connected, you make the dongle search for what Dynamixels, if any, are connected to it. The Dynamixels used in lab have been initialized to a baud rate of 400kbps[1] for you already, so select "Custom Search" from the drop-down menu on the right; then, from the list of what baud rates to search, check only the box for 400000 bps (AKA Baud 4). This setup is shown in Figure 2. Click "Start searching".

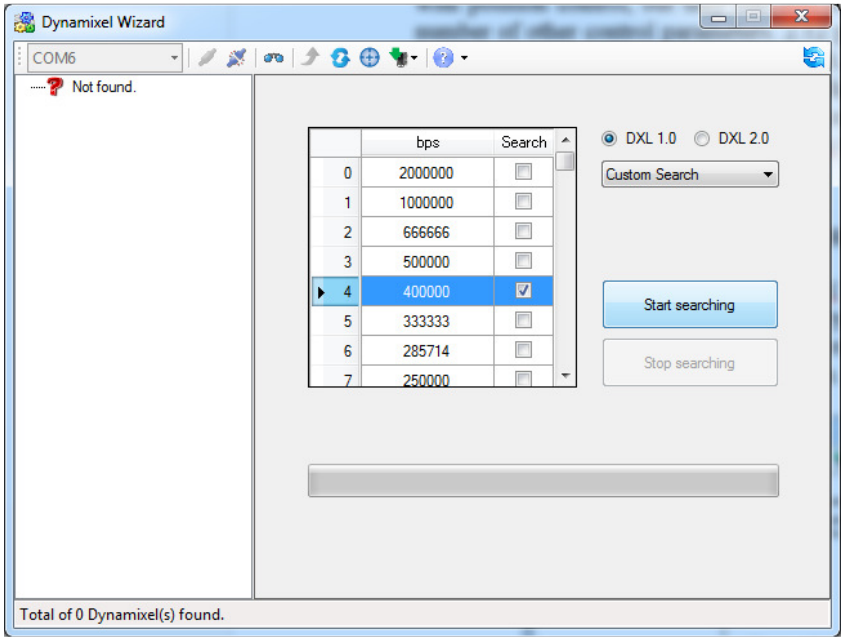

Figure 2: Dynamixel Wizard with search setup correctly

The search should return the two Dynamixels attached to the dongle. Select one by clicking on it. Listed are several parameters that can be set or read. Set its goal position to 180 degrees. Vary the position slightly, but be careful not to break the arm by causing a collision. Do the same with the second servo.

[1] A new Dynamixel motor comes with a default baud rate of 57600. Select the "Basic Search" option from the drop-down menu to a new Dynamixel connected to the computer, then manually set the baud rate to 400000 bps (4th parameter in the Dynamixel wizard) before using it with the Matlab code.

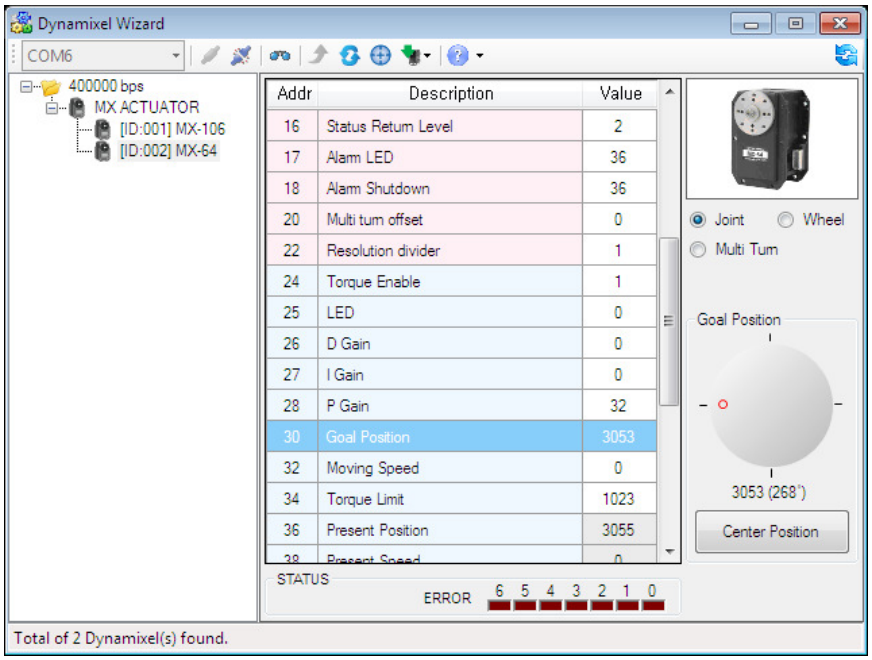

Figure 3: Selecting a Dynamixel with the Dynamixel Wizard

Click the "Close Port" icon and close the Dynamixel Wizard.

3. Control in Matlab

Next, you will control the servos from Matlab. Provided for you in Dynamixels.m is a Matlab class for interfacing with the Dynamixel servos in basic ways: connecting, setting positions, and reading positions.

For an example of the code in use, Open up Master_Slave_Dynamixel_Demo.m; this is a simple master-slave command. One servo continuously tries to match its angle to that of the other servo.

3.1 Inverse Kinematics

Also included in the lab folder is the file ArmKinematics.m, which contains a class for representing the 2-linkage arm, for checking its workspace, and for calculating inverse kinematics.

However, ArmKinematics.m is missing the definition of the method for performing inverse kinematics. **Please fill in the definition of the function *findThetas***. This function takes desired end effector coordinates *x* and *y* as inputs, and it returns joint angles *theta1* and *theta2* as outputs. You can find the place to write your code at approximately line 38 of ArmKinematics.m, at the bottom of the file.

The distance from the axis of rotation of the first servo to the axis of rotation of the second is 18 cm, and the distance from the axis of rotation of the second servo to the centerline of the hole at the end of the arm is 23.8 cm. These can be passed to the

ArmKinematics object when it is initialized, and then they can be accessed by *findThetas* as *obj.l1* and *obj.l2*.

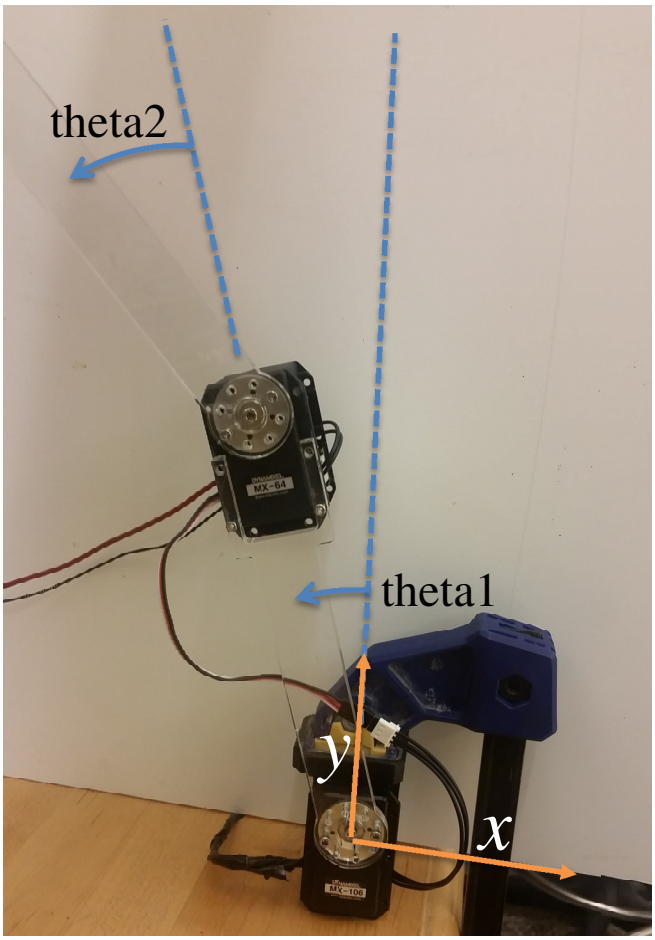

Figure 4: Coordinate systems of the arm

Once you have completed the inverse kinematics function, check that it works by plugging in some *x* and *y* coordinates and verifying that the angles correspond to what you expect.

Then, try using the inverse kinematics function to move the end effector to a particular point using the indicated section in Lab4_Student.m.

Show your arm reaching desired points to a TA for check off 1.

3.2 Safety and Valid Workspace

Some points are unreachable by the arm, and some points are reachable but would cause a collision if the arm tried to reach them. ArmKinematics includes a method *inWorkspace* for checking if a point is in the workspace of a robot. Change the definition of this method to check for these conditions and return True only if the point is within the

workspace of the robot and will not cause any collisions, otherwise the function should return False. You can find this function's definition at line 26 of ArmKinematics.m. The use of *inWorkspace* can be found in Lab4_Student.m where it is called to check if the command will be in the workspace before writing it to the motors. You can find this in line 29 of Lab4_Student.m. Show your working code to the TAs for check off 2.

3.3 Tracing a Trajectory

You can also use inverse kinematics to trace out a trajectory. Show your arm tracing a circle to a TA using the commented out section at the end of Lab4_Student.m for a bonus point.