Robotic Manipulation – ELEC60030

Lab 2 – Planar Robot Forward Kinematics

This lab is your first chance to interact with a 'real' robotic system – a planar RR robot. RR means that it has two rotational joints. This is the classic robotic example that many robotic text books use to explain some of the concepts useful for modelling and kinematic control.

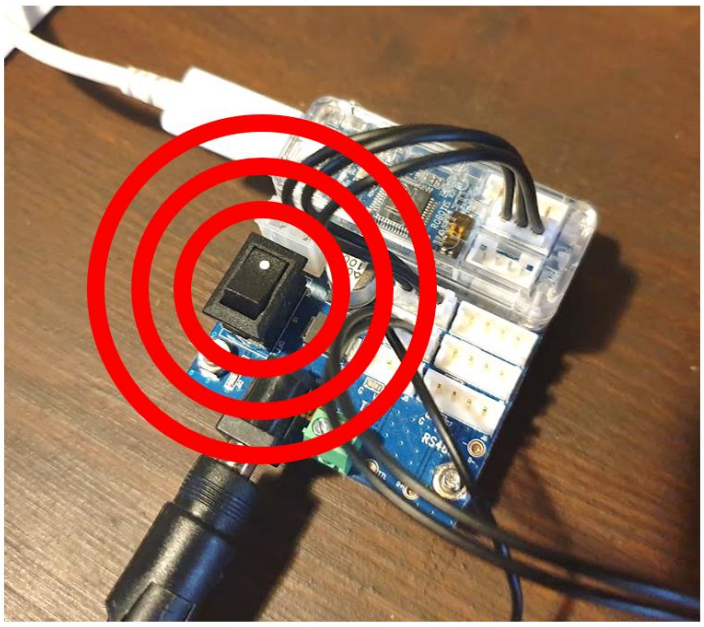
This lab will combine the practical interfacing skills of Lab 1 with the theory from Lectures 2 and 3.

IMPORTANT – The RR robot is now capable of pinching your fingers or damaging itself through self-collision of link 1 with the base.

The robot has been risk assessed and is considered safe, but please don't put your fingers near the robot when the torque is enabled and the actuators are set to move. We have some masking tape that you can use to secure the base of the robot to the desk, so you don't have to hold the base to stop it moving around.

Note that the encoders of the robot are set so that a position of 0 will command the link to move to 6-o'clock. Please do not do this for link 1. Position demands should stay within 600-3400. We have provided some code to set these limits within the on-board Dynamixel controllers.

In case of finger-trapping or collision, turn the power off immediately on the U2D2. This will disable torque to the actuators but won't negatively affect anything else. The power switch is highlighted below. Practice turning this off while the servos are moving in task 2.



Task 1 – Control Two Dynamixels Simultaneously

In this task we are going to move two Dynamixels simultaneously.

Edit your code from last week to communicate with both simultaneously. Note that the IDs of the Dynamixels are printed underneath them. Be sure to read these before you tape your robot to the desk.

1. Start by reading the encoders of both Dynamixels and printing these to the Matlab terminal

2. Add the following code after you open the port to prevent the Dynamixels from exceeding the motion limit when moving.

```
% ----- SET MOTION LIMITS ----------- %

ADDR_MAX_POS = 48;
ADDR_MIN_POS = 52;

MAX_POS = 3400;
MIN_POS = 600;

% Set max position limit
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_MAX_POS, MAX_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_MAX_POS, MAX_POS);

% Set min position limit
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID1, ADDR_MIN_POS, MIN_POS);
write4ByteTxRx(port_num, PROTOCOL_VERSION, DXL_ID2, ADDR_MIN_POS, MIN_POS);

% ---------------------------------- %
```

3. Now try sending a sequence of three position demands to both Dynamixels simultaneously (so both Dynamixels get the same position demand at the same point in the sequence). You will need to include a pause command after each position demand to make sure the Dynamixels have time to complete the motion.

4. Now try sending a sequence of three different position demands to both Dynamixels separately.

Notice how the motion seems quite ‘violent’ for big motions. You may notice that the base seems to try and move around on the table. Within your group, discuss why you think this is. Also discuss what sort of effect this might have for larger robots and how you could try and improve this.

Task 2 – Trajectory Tracking

An important feature of industrial robots is their ability to perform repetitive tasks, such as assembling the same type of car over and over again. This is achieved by getting the actuators to follow pre-determined trajectories. Before we consider how functional trajectories are generated (which will be a topic for future lectures), let us consider co-ordinated joint motion.

1. Revisit the sine wave tracking example from Lab 1. We want to get this working on both Dynamixels. This is a simple example of getting a robot to follow a set of joint trajectories. There were several ways to achieve sine wave tracking in lab 1. If you didn't get it working before, here is the method I use:
    a. Without connecting the Dynamixels, use Matlab to generate a sine wave function over a given number of time steps (for example 400 steps).
    b. The output of a sine function will naturally oscillate between -1 and 1. Modify the function to produce an output of ±500 encoder counts centred around 12 o'clock on the servos (encoder count 2046).
    c. You should store your sine function as an array of values.
    d. Use of for loop to step through each value of the sine wave, sending it to the two Dynamixels.
    e. You may notice that the motion is a bit jerky in places. Discuss with your group why you think this may be.
2. Now try and get the two actuators to follow two different trajectories. For example, a sine wave and an inverted sine wave or a sine wave and a cosine wave. Show the GTAs when you are done.

Task 3 – Frame Assignment and Forward Kinematics

Forward Kinematics is the ability to calculate the pose of different parts of a robot based on the joint angles. In this task we are going to learn how to calculate the position of the centre of the hole at the end of the robot, in Cartesian space. We will be doing this using frames and transformation matrices.

1. Create some code (based on Task 1) that reads the Dynamixel encoders and converts these into angles (in both degrees and radians). Check the datasheet to see how many encoder counts are in a revolution. You probably want to display the angle data in degrees in the Command Window while moving the robot by hand (with the torque disabled) to make sure the angle values look correct.

2. Looking at the slides of lecture 3, write out a 2D transformation matrix and name it T_0. This will be the base frame. T_0 should consist of a rotation matrix R_0 and a position matrix P_0.

3. Consider the positions of the robot frames from the lecture. Create a transformation matrix for each one, using the naming convention T_n, where n is the frame number.
    a. Consider which frames are translated from prior frames and modify their position matrix (the link lengths are written on the robot).
    b. Consider which frames are rotated from the prior frames and include the relevant theta angle in the rotation matrix (note that this should be in radians)

4. Now sequentially combine the transformation matrices to determine the position of the end of link 1 (the elbow joint) and the end of link 2 (which we can call the tool). Again, stream these to the command window as you move the robot around make sure the values make sense.

5. Now save the X-Y values of the tool over a given period of time as you move the robot around. Use the plot command in Matlab to plot X against Y (e.g. plot(x,y)). You can use the 'axis square' command to fix any linear distortion from plotting.

6. Try and draw a square (with around 7-10cm width) by manually moving the tool. Does the resultant trajectory look how you anticipated?

7. Use 'title(xxx)' to set the title of the figure to your group's name. Save your groups' best attempt at a square as a file. We'll upload them to Teams and see who did best. :-)

8. Show the GTAs when you are done.

Bonus: Task 4 – Moving in a Square

It's too early in the course to attempt inverse kinematics (where we calculate the joint angles necessary to achieve a desired tool pose) but we can do some simplified tool control.

1. Set your previous code to also display the encoder counts in addition to XY

2. Try and draw a square again by manually moving the robot with the torque disabled. In each corner pause your motion make a note of the encoder values.

3. Repeat Task 1 but for your sequence of encoder values, use the encoder values for the corners of the square.

4. How much like a square is the resultant robot motion? Are the lines the tool makes between the points straight? Discuss why the tool path appears as it does?

5. Show the GTAs and take a video of your robot moving (for uploading to blackboard / teams later)