Lab Session Robotics

October 2023

1 General Notes

1. This lab session will cover the experimental application of all the three assignments involving Kinematics, Jacobians and Trajectories. This task accounts for 1 point (70% lab +30% report).
2. To grade your lab work, please SHOW TA the motion of your real robot arm to gain 70% of the lab work, the left 30% point will be graded by your mini-report. Please hand in it no later than 23:59 on 27th October.
3. Please send your lab report to *BRIGHTSPACE*, named using the following syntax GroupNumber_labreport.zip.

2 How to work with the robotic arm

Connect the robot to your computer and start MATLAB. Firstly, add the toolbox into the workpath and the RTB toolbox (all materials can be downloaded from brightspace):

```
>> addpath('add your local path here...\rvctools\common')
>> addpath('add your local path here...\rvctools\robot\interfaces')
```

Then check the port COMx of the robot arm:

```
>> arb = Arbotix('port','COMx','baud', 57600)
```

This command corresponds to the connection of the robot #x. To adapt it for your robot, change the last number in the command above. Now you have a connection to the robot through the workspace class variable *arb*.

Let's start with something innocuous, taking the temperature of the servo motors

```
>> arb.gettemp

ans =

    32    34    33    31    31    34
```

which is the temperature of each servo motor in degrees Celsius. Now let's get the actual angle of joint 1, the waist joint

```
>> arb.getpos(1)

ans =

    0.7414
```

which is the angle in radians. Of course, for your robot you will most likely get a different answer. You can get the joint angle for the other joints in a similar way. We can get all the joint angles simultaneously.

```
>> arb.getpos()

ans =

    0.7414   -1.9124   -0.2096    0.0511    0.1841   -0.0869
```

Now let's try a cautious motion, we will move the gripper

```
>> arb.setpos(6,1)
```

And back

```
>> arb.setpos(6,0)
```

Now let's move the whole robot to the zero configuration

```
>> arb.setpos([0 0 0 0 0 0])
```

The robot is quite rigid since the Dynamixel servos are doing their job very well. We can relax the robot by

```
>> arb.relax
```

which puts all the servos into a zero torque mode and we move the joints by hand. The only force we feel is due to the friction in the Dynamixel gearbox.

To send the whole trajectory to the robot you can use the command

```
>> dt = 0.01;
>> arb.setpath(qTraj,dt);
```

where $qTraj$ is a $N \times 6$ vector of set-points and dt is a time step between two sequential set-points. To realize model-based control strategies we need to have the formal representation of the robotic arm. We can create a kinematic model of the robot by

```
>> mdl_arbotix
```

which creates the workspace variables $mArb$ (the robot)[1]. Although the robot has 6 servo motors, since one is the gripper, this robot has only five joints.

```
>> mArb

mArb =

Arbotix (5 axis, RRRRR, modDH, slowRNE)

+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|          0|          0|      1.571|
|  2|         q2|          0|          0|     -1.571|     -1.571|
|  3|         q3|          0|         17|          0|      1.571|
|  4|         q4|         11|          4|      1.571|      1.571|
|  5|         q5|          0|          0|      1.571|          0|
+---+-----------+-----------+-----------+-----------+-----------+

grav =    0  base = 1   0   0  25   tool =   0   -1    0    0
          0         0   1   0  25            0    0    1    9
       9.81         0   0   1  17           -1    0    0    0
                    0   0   0   1            0    0    0    1
```

Let's plot the robot in its zero angle pose

```
>> mArb.plot([0 0 0 0 0])
```

and we see it is standing straight up. We can drive the graphical robot around using the virtual teach pendant

[1]You can find the details on how a robotic arm model is created, via the Peter Corke open source robotics toolbox, into the toolbox manual available in BrightSpace.

```
>> mArb.teach
```

and manipulating the sliders causes the graphical robot to move. Note the coordinate frames of the world and of the tool.

3 Lab Task $(1 = 0.7\%$ lab $+0.3\%$ report $)$

The main goal of the joint space control is to design a feedback controller such that the joint coordinates track the desired motion as closely as possible. Firstly, the desired motion, which is described in terms of end-effector coordinates, is converted to a corresponding joint trajectory using the inverse kinematics of the manipulator. Then the feedback controller determines the new set-points for joints local controllers to move the manipulator along the desired trajectory specified in joint coordinates starting from measurements of the current joint states. Due to its simple structure, this kind of control schemes offers many advantages. For example, by using independent-joint control, communication among different joints is saved. Moreover, since the computational load of controllers may be reduced, only low-cost hardware is required in actual implementations. Finally, independent-joint control has the feature of scalability, since the controllers on all the joints have the same formulation.

3.1 Description of the task:

1. Define two positions in the workspace $p_a = [x_a, y_a, z_a]^\top$ and $p_b = [x_b, y_b, z_b]^\top$ and determine the appropriate configurations of joints. (Hint: try the command mArb.ikine $(\cdot)$)
2. Plan a smooth trajectory between these points (Hint: try the command $j\,$traj$(\cdot)$)
3. Save the trajectory points as a matrix in the workspace variable and check your result based on mathematical model of the robot. Plot the resulting trajectory of the end-effector (Hint: try the command mArb.fkine $(\cdot)$).
4. Apply this trajectory for the real robotic arm. Compare the performance of robot arm and your simulation result.