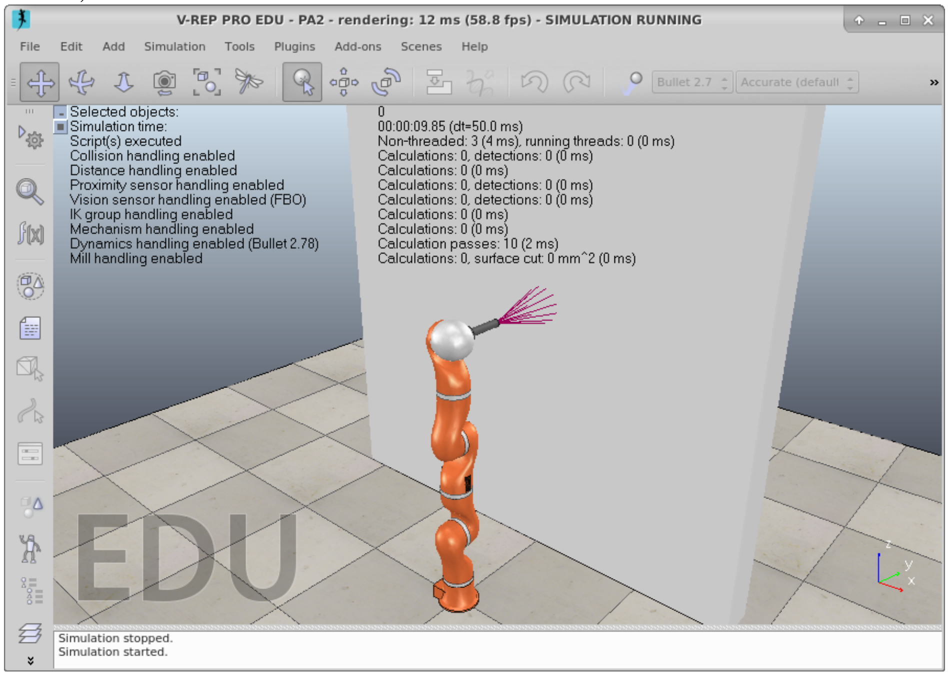
CSCI 445 Programming Assignment 2 — Manipulation

Prof. Ayanian, University of Southern California Spring 2017

In this assignment you will learn how to use a robotic arm in a spray-painting application.

1 Getting Familiar with the Robotic Arm

Extract the archive (`PA2.zip`). Open `PA2.ttt` in V-REP. You should see a robotic arm as used in industrial applications. Change the code (`pa2.py`) such that the end-effector, which is equipped with a spray-painter, points in the direction of the wall. As usual, you can run your code using `python3 run.py --sim pa2`. If successful, V-REP should visualize the arm as shown below:



```
Selected objects:                          0
Simulation time:                           00:00:09.85 (dt=50.0 ms)
Script(s) executed                         Non-threaded: 3 (4 ms), running threads: 0 (0 ms)
Collision handling enabled                 Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                  Calculations: 0 (0 ms)
Proximity sensor handling enabled          Calculations: 0, detections: 0 (0 ms)
Vision sensor handling enabled (FBO)       Calculations: 0, detections: 0 (0 ms)
IK group handling enabled                  Calculations: 0 (0 ms)
Mechanism handling enabled                 Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)    Calculation passes: 10 (2 ms)
Mill handling enabled                      Calculations: 0, surface cut: 0 mm^2 (0 ms)
```

```
Simulation stopped.
Simulation started.
```

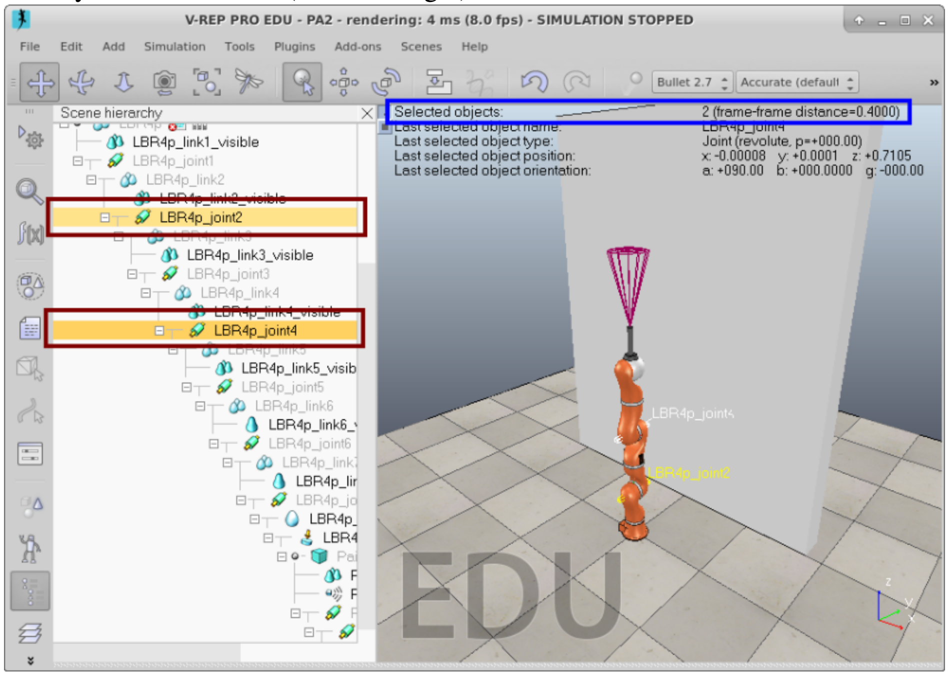
2 Forward Kinematics

Even though our robotic arm is in 3D, we will just use it in a plane to mimic the 2D version discussed in class. Find the two relevant joints such that the robot roughly follows the model in class and for each one find its angular range. Implement forward kinematics for the full 3D position $[x, y, z]^T$ (since we do everything in a plane one of the coordinates will be a constant). Let your robot go to four different joint angle pairs and print the estimated coordinates on the screen. Make sure that those coordinates match the ones shown by V-REP when `LBR4p_joint7` is selected (up to a small epsilon due to numerical inaccuracies).

The output of your program should be formated as follows (the numbers will **not** match):

```
Using joint 42 (range 0 to 90 degrees)
Using joint 88 (range -180 to 0 degrees)
Go to 45,-90 deg, FK: [0,0,0]
Go to 90,-70 deg, FK: [1,4,0]
Go to 20,-180 deg, FK: [2,5,0]
Go to 45,-10 deg, FK: [3,6,0]
```

Hint: You will need to estimate the length of the links. The easiest way to do so is to use V-REP: if you select two objects in the "Scene hierarchy", the text on the top-left corner will include the distance between the two objects. Below is an example where joint2 and joint4 are selected (red rectangle) and the computed distance by V-REP is $0.4\,\mathrm{m}$ (blue rectangle):



V-REP PRO EDU - PA2 - rendering: 4 ms (8.0 fps) - SIMULATION STOPPED

```
Selected objects:                      2 (frame-frame distance=0.4000)
Last selected object name:             LBR4p_joint4
Last selected object type:             Joint (revolute, p=+000.00)
Last selected object position:         x: -0.00008   y: +0.0001   z: +0.7105
Last selected object orientation:      a: +090.00    b: +000.0000   g: -000.00
```

3 Inverse Kinematics

In order to compute the joint angles for a desired position we need to solve the inverse kinematics problem. Given the coordinates on the wall (since the robot is parallel to the wall we again need only 2 coordinates) we need to compute the desired angles of our joints. Implement inverse kinematics for our robotic arm. Test your implementation with at least three different points and verify that the coordinates you specify match those of V-REP.

Note that in the equations for inverse kinematics there are two possible solutions. Make sure to select the appropriate solution automatically.

The output of your program should be formated as follows (the numbers will **not** match):

```
Go to [0,0], IK: [-90 deg, 90 deg]
Go to [0,1], IK: [-60 deg, 50 deg]
Go to [0,2], IK: [-70 deg, 40 deg]
```

4 Drawing

We are now ready to use the spray-painter to draw various shapes. You can set the color (RGB) of your spray can by using

```
self.arm.set_color(0.0, 0.0, 1.0)
```

and you can enable the spray by calling

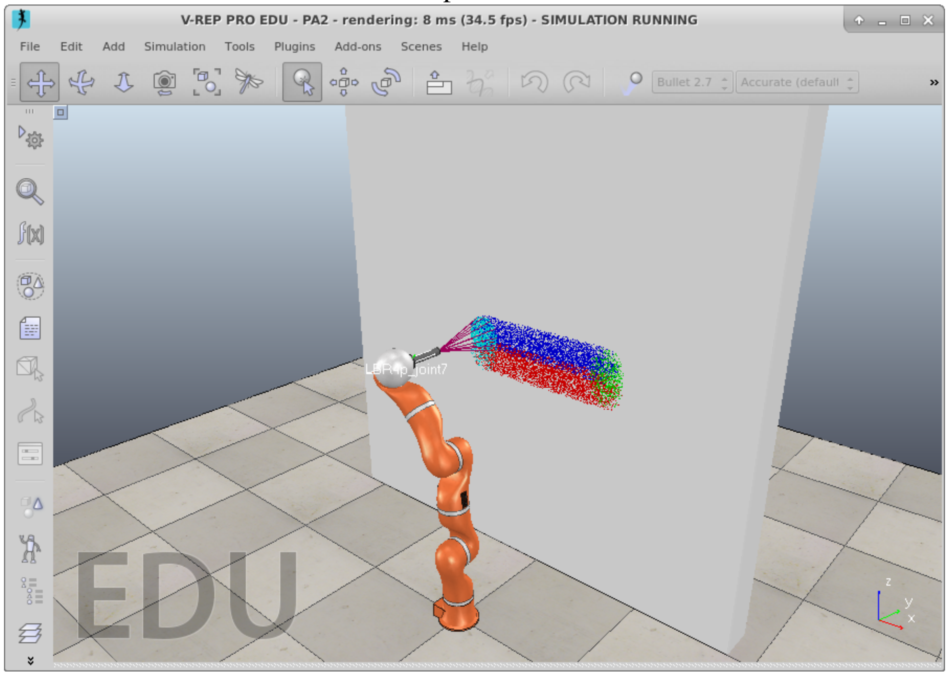
```
self.arm.enable_painting()
```

4.1 Rectangle — Attempt 1

Attempt to draw a rectangle by just providing the vertices to your inverse kinematics function. Use the following vertices coordinates (in the xz-plane): $[-0.3, 1.0]$, $[0.3, 1.0]$, $[0.3, 0.9]$, $[-0.3, 0.9]$.

4.2 Rectangle — Attempt 2

Improve your previous attempt such that the drawn figure looks more like a rectangle. Furthermore, each edge should have a distinct color. The output should look like below:



4.3 Custom Drawing

Be creative and let your robot draw for you.

5 Submission

Please create a zip-file named `firstname_lastname.zip` containing `pa2.py` and submit the archive using BlackBoard. All files should be in a working condition, i.e. we should be able to execute your code on our side using Python3 during grading.