CSCI 4302 - LAB1 - INVERSE KINEMATICS

The goal of this lab is to implement inverse kinematics and trajectory control for the UR5e robot, mimicking the built-in functions “MoveJ” and “MoveL” of the UR robot. “MoveJ” implements inverse kinematics (IK) and moves the robot in joint space. “MoveL” also solves the IK problem, but lets the robot move in Cartesian space. This requires to compute waypoints in Cartesian space, solving the IK problem for each of them, and then moving the robots along them. As a “collaborative” robot, the UR robot also implements some low-level safety feature. The user can specify multiple planes that no part of the robot will cross.

Specific goals:

- implement a solution to the basic IK problem
- understand how join speed affects trajectory following
- implement a basic waypoint generator
- implement basic collision avoidance.

OVERVIEW

A UR5e robot is positioned to be able to collect beverage cans that are presented on a conveyor belt. The robot is positioned such that it can reach the entire width of the conveyor belt, allowing it to capture cans no matter where they are on the conveyor belt. The goal of this lab is to implement the motion primitives that are necessary to move the robot appropriately. This lab needs to be implemented in three steps. First, implement all the tooling that allows Webots to read the robot’s joints and set them. Second, implement the forward kinematics of the robot to obtain it’s actual position as well as the position of all its joint. Finally, implement a basic IK solver that computes the target joint angles for a given pose. You will need to implement a loop that waits until the robot is close to the desired location. This will be your “MoveJ” command. Second, generate a trajectory that allows the robot to approximately move on a line between two points using “MoveJ”. This time, draw the next waypoint as soon as the robot is close enough. This will be your “MoveL” command.

This is a 3-week lab. Plan for satisfying the “preliminaries” in Week 1, implementing the MoveJ command in Week 2, and implementing MoveL in Week 3.

REQUIRED FILES

- Download files from **Github**

Inspect the Webots world and think about the motions that would be necessary to captur`e every can on the conveyor belt. What would happen if the robot would be allowed to move to these positions any way it wants? Why would you need only commands like MoveJ, but also MoveL?

Think about the end-effector's pose relative the base of the robot. What sensor information do you need to compute the pose of the robot's end-effector?

How many solutions will you possibly get to achieve the same pose using a six degree-of-freedom robot?

PRELIMINARIES

- Implement functionality to control and read the position of all six joint angles of the UR5e in Webots. Open the robot's Proto file to find out how all its joins are called. Note that it takes a couple of seconds for the robot to "boot" and provide accurtate joint angles. You will therefore need to run robot.step() a couple of times.
- Play with different poses of the robot by manually using its joints. Which joint angles should be "forbidden" to prevent damage to the robot or the environment?

Think about how you would implement a function that checks whether a set of joint angles is allowable or not. Is it sufficient to simply specify a range for each joint? Why not? Consult the UR5e manual to find out how a user can specify "keep out" zones.

- Implement the forward kinematics for the UR5e robot. You can copy and paste code that you find online, but do not use a library as you will need to deliver a single file. Output the homogeneous transform for selected joint angles.
- Write a function that tests whether a given set of joint angles will result in any point of the robot being left or right of a given plane. Notice the word "any". Your function needs to be called "isJointPosSafe(joints,plane)" and take as inputs the joints angles of the robot (six values) as well as a plane given by four values a,b,c,d (four values). Your function needs to return "True" if all joint centers are above the plane and "False" otherwise.

Hints: a plane is defined as $a*x+b*y+c*z+d=0$. Compute a, b, c and d from three given points, then check wheter a point (x,y,z) is above (negative) or below (positive) the plane.

MOVEJ

- Consult the UR5e reference manual to understand the MoveJ command.
- Implement an inverse kinematics solver for the robot. You can copy and paste code that you find online, but do not use an IK library.
- Compute the inverse kinematics for a desired pose (hint: you can obtain suitable rotation matrices from the forward kinematics by manually moving the robot in the desired pose)
- Reject all joint positions that lead to an unsafe position. Use "plane" as a global variable.
- Implement the MoveJ command to servo to a desired pose. Use the maximum angle that the robot has to move to compute the number of timesteps required to perform that motion. Disretize the joint angles into the appropriate number of bins and incrementally move the joints there. Test whether a configuration is "safe" before you move there. Throw an exception otherwise.

Hint: unlike other objects in the environment, the robot will not fall (due to the parameter "staticBase" set to TRUE). You can therefore delete all the objects that are in its way (as well as lift it a little bit) to observe how illegal motions do look like when fully executed.

MOVEL

- Use the forward kinematics to calculate the current position of the robot
- Determine the Euclidian distance between the current and the desired position of the robot
- Assume a maximum Cartesian speed of the robot, e.g. 1cm/s, and generate a list of waypoints from the current to the desired position of the robot
- Create a control loop that calls MoveJ to move from point to point. For smoother motion, you can pull the next waypoint, once the robot gets reasonably close to the next waypoint.
- Observe what happens if you try to move too far or in other ways that are kinematically not feasible.

Note: You can use the orientation of the robot at the beginning of the motion and stick to it.

SUMMARY

1. Feed the joint angles to the FK. Does the position and rotation make any sense?
2. Feed the result (homogeneous transform) to the IK. Do you get the same joint angles?
3. Subtract 20cm from the X-value in the homogeneous transform and feed to IK. Does the robot move to the right?
4. You can set the joint angles directly at first (wait 200 timesteps) and see what happens
5. Instead of moving all at once, implement MoveJ as a for-loop moving step by step. Now implement MoveL to move through Cartesian Space.

DELIVERABLES

- A Python module with blocking implementations of isJointPosSafe, MoveJ and MoveL. Both functions will need to execute until the robot reaches a desired location or return failure. Assume "plane" to be a global variable that defines the robot's keep-out zone.
- Please only deliver one Python file, that is all code for inverse kinematics needs to be contained therein. Do not submit iPython notebooks, but convert them into a stand-alone controller (see below). Only provide variable and function definitions. Make sure you comment out all your other code/tests, so that your submission can be tested by simply importing the file.

NOTES

If you are using Jupyter lab, you can use the following commands to turn your notebook into a stand alone controller:

```
try:
   !jupyter nbconvert --to script ik_solver.ipynb
   %load_ext autoreload
   %autoreload 2
except:
    pass
```

This assumes that your notebook is called “ik_solver.ipynb” and will create a script “ik_solver.py”