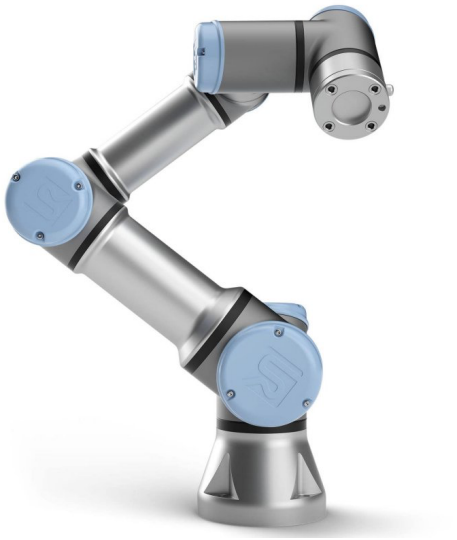
2023 WiSe Reinforcement Learning – Final Examination – Coding Task

6-axis Robot



The inverse kinematics of a robot with six axes is to be trained by means of a Reinforcement Learning Agent. The robot is to be considered to be the Environment, with the joint variables as input. The Environment is fully deterministic, i.e. there aren't any external factors that influence the robot's response to the input.

The state of the Environment is observed as the coordinates of the Tool Center Point (TCP) and its orientation in a 3D space, i.e. $(x, y, z, \alpha, \beta, \gamma)$ which can be calculated from the Direct Kinematics when the six joint angles are known..

All joints of the robot are rotary. The joint variables are the joint angles. The Denavit-Hartenberg matrix for the Direct Kinematics of the robot is as follows:

JOINT TRANSFORMATION	a_i [M] (LINK LENGTH)	d_i [M] (LINK OFFSET)	α_i [RAD] (LINK TWIST)	θ_i [RAD] (JOINT ANGLE)
1	0	0.15185	1.570796327	θ_1
2	-0.24355	0	0	θ_2
3	-0.2132	0	0	θ_3
4	0	0.13105	1.570796327	θ_4
5	0	0.08535	-1.570796327	θ_5
6	0	0.0921	0	θ_6

The Agent has the following action possibilities for each joint angle separately:

ACTION PRO JOINT	$\delta\theta_i$ [DEGREE]
DECREASE	- 0.1°
KEEP	0.0°
INCREASE	+ 0.1°

Please observe that the actions are given as "deltas" and not as absolute angles for the joints, that means that the Agent must decide if the angles increase, decrease or keep a joint angle. However, that must be defined for all six joints, meaning that the Agent can take at every Time Stamp 3^6 possible Actions (= 729)! Every joint angle is bounded to $-180° < \theta_i < +180° \ \forall i \in [1,2,3,4,5,6]$.

The Agent has to learn to perform a 3D trajectory in space, i.e. the trajectory may not be only on a the xy-plane, or the yz-plane nor on the zx-plan: The trajectory must show to change all three coordinates.

Your Agent should be trained using 3-step Bootstrapping to learn a trajectory. The trajectory that your Agent has to learn is a Helix with a radius of 3 cm and 2 turns. The robot has to keep the same orientation which travelling through the trajectory: You can define the orientation but you have to keep it constant all the time! It is free for you to decide where the trajectory is in the 3D-space. The Environment of a robot like this is huge, and therefore, you need to define a voxel in the 3D-space where the task will be solved: That voxel will be the Observation Space. The voxel has to be large

enough to create boundaries for the learning processes, … but not too large to blow the computer's memory space!

Your team has to:

1. Define the 3D-trajectory mathematically.
2. Define the Observation Space in the form of the voxel where the trajectory will be.
3. Code the Environment with the given direct kinematics and the constraints of the join angles and the chosen voxel. Define a suitable Reward Function for the Agent, as well as the strategy how to deal with occasions where the TCP leaves the trajectory and/or the voxel.
4. Code the defined algorithm to come to an optimal Policy.
5. Verify the performance vs. time of the Learning process showing that your Agent is in fact learning and the decisions are becoming "better" after each training.
6. Once your Agent learned the optimal Policy, test the Policy showing that the robot's TCP is indeed following the trajectory. Compute the mean square error (MSE) from the ideal trajectory and what the Agent is doing in reality.
7. Propose new ideas how to reduce the MSE … you don't need to code them, but your ideas must be plausible and realistic: Substantiate them!
8. Make sure that your code has some tools to allow for easy verification of the performance of your Agent.