Assignment 2: UR3 Manipulation & Training

Due: Sun 15th December 2024 at 23:59 (IST – Irish Standard Time)

Once completed, upload all of your code to Canvas in a single archive (zip, tar, etc.). Your submission should include all ROS packages created for this assignment.

Tasks

1. Create a CMake package named ur3_description:

   ```
   ros2 pkg create --build-type ament_cmake ur3_description
   ```

   Download the file Assignment2_Task1.zip from Canvas and extract. Copy the config, launch, meshes, models, rviz and urdf folders into the ur3_description package directory.

   Add the following to CMakeLists.txt (after "find dependencies" and before "build testing"):

   ```
   install(
     DIRECTORY config launch meshes models rviz urdf
     DESTINATION share/${PROJECT_NAME}
   )
   ```

   Add the following to package.xml:

   ```
   <exec_depend>joint_state_publisher_gui</exec_depend>
   <exec_depend>launch</exec_depend>
   <exec_depend>launch_ros</exec_depend>
   <exec_depend>robot_state_publisher</exec_depend>
   <exec_depend>rviz2</exec_depend>
   <exec_depend>urdf</exec_depend>
   <exec_depend>xacro</exec_depend>
   ```

   Build the package and execute ur3_rviz.launch.py to view the robot in RViz and familiarise yourself with its structure and workings.

   Modify your package to add controllers to the UR3 robot. Add a joint state broadcaster, in addition to a forward command controller and joint trajectory controller to control the robot joint positions. Allow full 360° movement of all joints (i.e. min/max values of $\pm 2\pi$ for your controllers) except the elbow joint, which should have min/max values of $\pm\pi$.

   Once finished, execute ur3_gazebo.launch.py to start the simulation.

   [10%]

2. Create a new package, ur3_move. Within this package, create a ROS node that moves the joints of the UR3 robot by amounts specified by the user.

The node should loop indefinitely (until terminated by the user). On each loop iteration, you should cycle through the moveable joints, prompting the user for an amount to move that joint by.

The node should then move each joint by the specified amounts and, once the movement is complete, print:

- The final actual position of each joint.
- The error in the position, relative to the desired set-point, of each joint.
- The position and orientation of the end effector relative to the base.

Note that the joint controllers of the robot accept position commands. The user enters relative, not absolute, positions, i.e. the user specifies the amount to move each joint by, not the position to move the joint to. Therefore, you need to determine the absolute position from the current position and the user input before sending the command to the joint controller.

[15%]

3. Create a ROS node (within the ur3_move package) that moves the joints of the UR3 robot as follows:

$$s_j(t) = \begin{cases} \frac{\pi}{2}\left(1 - \cos(0.2\pi t)\right) & \text{Shoulder Pan Joint} \\ \frac{\pi}{4}\left(\cos(0.4\pi t) - 1\right) & \text{Shoulder Lift Joint} \\ \frac{\pi}{2}\sin(0.2\pi t) & \text{Elbow Joint} \end{cases},$$

where $s_j(t)$ is the position of a given joint at time t, i.e. the joints should oscillate as follows:

Joint	Range (radians)	Frequency (Hz)	Period (s)
Shoulder Pan	$[\,0,\,\pi\,]$	0.1	10
Shoulder Lift	$[\,-\frac{\pi}{2},\,0\,]$	0.2	5
Elbow	$[\,-\frac{\pi}{2},\,\frac{\pi}{2}\,]$	0.1	10

See ur3_demo.mp4 for a video demonstration of the required movement.

[10%]

4. Switch controller to the joint trajectory controller. Create a client for the corresponding action server that moves the joints of the UR3 robot as described in the previous task for one full period (i.e. 10 seconds only).

[5%]

5. Create an RL environment node that may be used to train the robot to reach the cube on the table. In addition to all required publishers, subscribers, etc., your environment should also provide the following functions:

- `set_action`: define what each action will do.
- `get_obs`: return current observation/state.
- `is_done`: return true if current episode has finished, false otherwise.
- `compute_reward`: return the reward for each step.
- `reset_environment`: return robot to its initial pose (all joints at 0.0 radians).

Environment properties:

- There are six available actions corresponding to +/- movements of each major arm joint (shoulder pan, shoulder lift, elbow). The step size for each joint is 0.1.
- The observation space is the position of each major arm joint.
- The agent (robot) receives a reward of -1 for each action step except if the goal is reached, in which case the reward is 100.
- An episode is complete if the end effector reaches its goal position within a tolerance of 0.02 (i.e. if the Euclidean distance between the end effector and its target is less than 0.02), or if the maximum number of iterations (100) is exceeded.
- The goal position (cube) relative to the robot base is $(x, y, z) = (0.03, 0.385, 0.03)$.

Apply Q-learning and SARSA to train the robot.

Note that the observation space is continuous* and should be discretised in order to apply Q-learning and SARSA.

[60%]

Aside:

The task to be solved here is simply an inverse kinematics problem and obviously does not require an RL solution. Also, we haven't considered any workspace collisions (e.g. collision between the robot arm and the table) or the fact that the robot arm may end up pushing the cube off the table. Furthermore, you wouldn't typically control the movement of the robot in the way described here (independently moving each joint). Instead, you would typically perform trajectory planning using the joint trajectory controller, ROS MoveIt, or similar.

*Although each joint only moves here in a discrete step-wise manner, the final actual positions of the joints may not be exact.