Mike Leske
R00183658

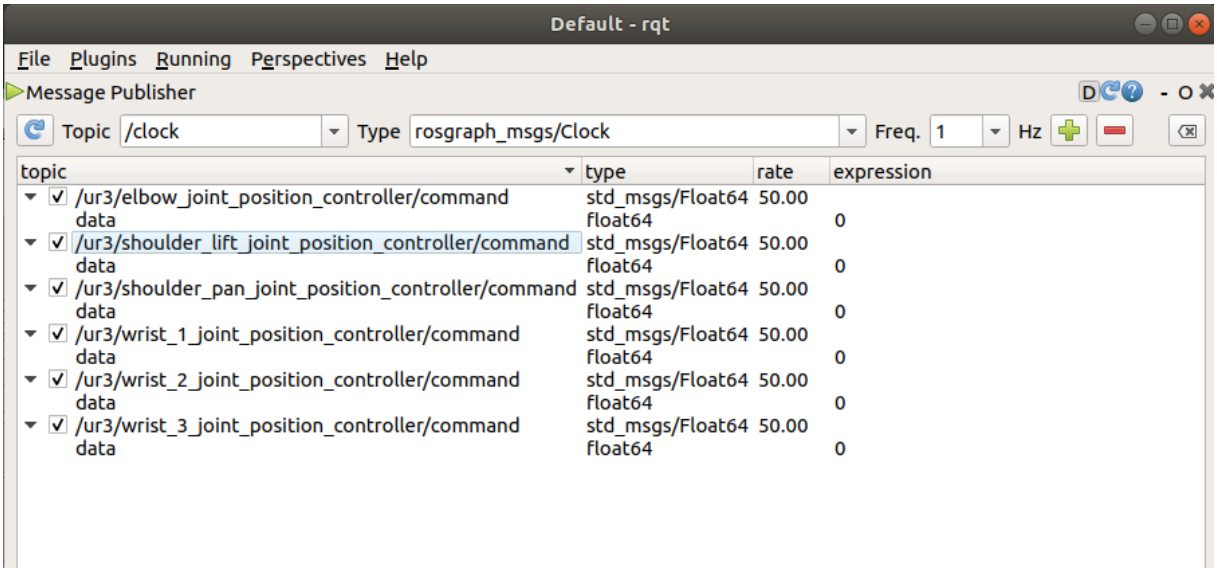
This document is intended to provide some additional guidance on the UR3 implementation and shall aid in correction the submitted assignment code.

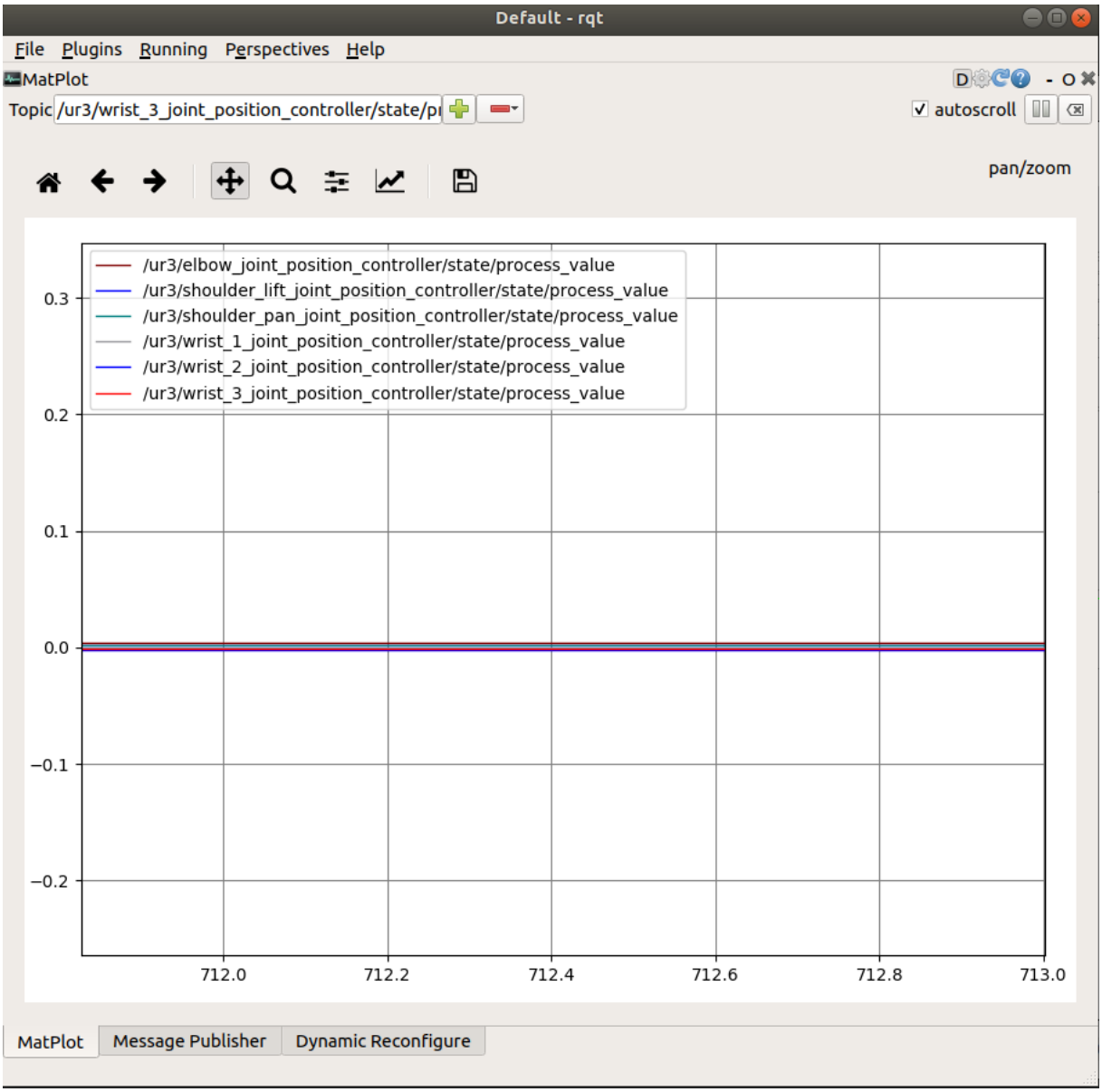
The following solution paths have been implemented as part of the assignment

1. Manual
    a. roslaunch my_ur3_reach start_training_manual.launch
    b. Allows to move the robot arm manually and accepts the following actions: 0, 1, 2, 3, 4, 5, 9
    c. Action 9 stops the training loop
2. Fixed Actions
    a. roslaunch my_ur3_reach start_training_fixed_actions.launch
    b. Runs a fixed sequence of minimal steps to reach the goal position
    c. Shall help to evaluate and grade e.g. reward, robot reset, done, etc.
3. Q-Learning
    a. roslaunch my_ur3_reach start_training_qlearn.launch
    b. Trains the robot arm with q-learning
4. SARSA
    a. roslaunch my_ur3_reach start_training_sarsa.launch
    b. Trains the robot arm with SARSA
5. DQN
    a. roslaunch my_ur3_reach start_training_dqn.launch
    b. Trains the robot arm with DQN from baselines

Each launch file points to separate config yamls and python code.

Task 1


Default - rqt
File Plugins Running Perspectives Help
Message Publisher
Topic /clock
Type rosgraph_msgs/Clock
Freq. 1
Hz
topic
type
rate
expression
/ur3/elbow_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0
/ur3/shoulder_lift_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0
/ur3/shoulder_pan_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0
/ur3/wrist_1_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0
/ur3/wrist_2_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0
/ur3/wrist_3_joint_position_controller/command
std_msgs/Float64
50.00
data
float64
0


Default - rqt
File Plugins Running Perspectives Help
MatPlot
Topic /ur3/wrist_3_joint_position_controller/state/p
autoscroll
pan/zoom
/ur3/elbow_joint_position_controller/state/process_value
/ur3/shoulder_lift_joint_position_controller/state/process_value
/ur3/shoulder_pan_joint_position_controller/state/process_value
/ur3/wrist_1_joint_position_controller/state/process_value
/ur3/wrist_2_joint_position_controller/state/process_value
/ur3/wrist_3_joint_position_controller/state/process_value
0.3
0.2
0.1
0.0
−0.1
−0.2
712.0
712.2
712.4
712.6
712.8
713.0
MatPlot
Message Publisher
Dynamic Reconfigure

Task2

1. Launch UR3: roslaunch my_ur3_description ur3.launch
2. Start script: python ur3_reach.py

The robot will appear in a “heads-down” position. Use the menu to bring the robot into its “0” position for all joints.

```
mikel@ubuntu:~/catkin_ws/src/my_ur3_reach/src$ python ur3_reach.py

Available joints:
  0  elbow_joint
  1  shoulder_lift_joint
  2  shoulder_pan_joint
  3  wrist_1_joint
  4  wrist_2_joint
  5  wrist_3_joint
  9  Exit
Select a joint to move: 0
Set position for joint: 0
```

This task calculates the error from the joint_states position info and the user-provided target position. Another option would have been to subscribe to the individual joint states and error the error from there.

Code Annotations Task 3 & 4

Register task environment

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/task_envs_list.py

```
elif task_env == 'UR3-v0':
    register(
        id=task_env,
        entry_point='openai_ros.task_envs.ur3.ur3:UR3PosEnv',
        max_episode_steps=max_episode_steps,
    )

    # import our training environment
    from openai_ros.task_envs.ur3 import ur3
```

This code was added to task_envs_list.py to make the UR3PosEnv task environment available to openai_ros.

Robot environment

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/robot_envs/ur3_env.py

```
def move_joints(self, joints_array):

    # Step 1:
    # Instruct joints to move to set position
    for pub, pos in zip(self.publishers_array, joints_array):
        joint_value = Float64()
        joint_value.data = pos
        rospy.logdebug(pub.name + " >> " + str(joint_value))
        pub.publish(joint_value)

    # Step 2:
    # Verify that all joints reach position, or release after self.move_check_num checks

    REACHED_POS = False
    # Loop for maximum self.move_check_num times
    for _ in range(self.move_check_num):

        # Give the arm time to move
        rospy.sleep(self.move_check_interval)

        # Get maximum error for the 3 joints we move to ensure all
        # Note: This excludes unused joints on purpose
        max_error = abs( np.array([self.pos[0:3]]) - np.array([self.joints[0:3]]) ).max()

        # If all the 3 joints are within error_tolerance, break
        if max_error <= self.error_tolerance:
            REACHED_POS = True
            break

    if not REACHED_POS:
        rospy.logerr("ERROR: Robot arm did not reach target position."))
```

All joints are moved to joints_array, which reflects the state of self.pos.
Next the function verifies whether all three joints of interest reached their position within a given error_tolerance, or after move_check_num validations. With the values provided in init, this check will loop for maximum 1 second (+ some overhead).

Task environment

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

```
        if action == 0:  # elbow -
            self.pos[0] -= self.position_delta
        elif action == 1:  # elbow +
            self.pos[0] += self.position_delta
        elif action == 2:  # shoulder_lift -
            self.pos[1] -= self.position_delta
        elif action == 3:  # shoulder_lift -
            self.pos[1] += self.position_delta
        elif action == 4:  # shoulder_pan -
            self.pos[2] -= self.position_delta
        elif action == 5:  # shoulder_pan -
            self.pos[2] += self.position_delta

        # Move the joints
        self.move_joints(self.pos)
```

Actions 0 to 5 add or subtract the position_delta to/from a given joint. Next, the move_joints function is called which will only close if a) the 3 joints of interest are in position or b) some maximum time expired.

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

```
    def _get_obs(self):

        obs = [ round(i, self.state_dicretization) for i in self.joints[0:3] ]
        return np.array(obs)
```

I decided to discretize the observation space in order to the algorithms to build solid policies. Otherwise, as joint movements are not perfect, the observation space, and so the policy, would become enormously large.

The parameter state_discretization is provided with the learning parameter yaml file(s).

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

```
    def _is_done(self, observations):
        done = False

        # check if robot end-effector is in range of goal
        if ((abs(self.goal_elbow - observations[0]) <= self.tolerance) and
            abs(self.goal_shoulder_lift - observations[1]) <= self.tolerance and
            abs(self.goal_shoulder_pan - observations[2]) <= self.tolerance):

            rospy.logerr("YEAH!!! Robot arm reached goal position")

            self.reached_goal = True
            done = True

            rospy.sleep(3)

        # Return done when number of steps / iterations are completed
        if self.iteration == self.max_iterations - 1:
            done = True

        rospy.loginfo("FINISHED get _is_done")

        return done
```

The _is_done function calculates whether the robot is within the tolerance range of the goal position. self.reached_goal helps to distinguish the ‘done’ state at episode end and episode end with reached goal as part of the last action.

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

```
def _compute_reward(self, observations, done):
    rospy.logdebug("START _compute_reward")

    cur_pos = np.array([self.joints[0], self.joints[1], self.joints[2]])

    reward = 1/np.sqrt(np.sum(np.square(cur_pos - self.goal_pos)))

    if self.reached_goal:
        reward += self.reached_goal_reward

    rospy.logdebug("END _compute_reward")

    self._publish_step_reward(reward)

    self.iteration += 1
    self.total_iteration += 1

    return reward
```

Each step returns a reward as specified in the assignment. Of the goal is reached, self.reached_goal_reward is added. Incrementing total_iterations is used for publishing rewards every step for monitoring intra-episode progress.

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

```
def _publish_step_reward(self, reward):
    '''
    Publish the step reward after each step.
    '''
    reward_msg = RLExperimentInfo()
    reward_msg.episode_number = self.total_iteration
    reward_msg.episode_reward = reward
    self.pub_step_reward.publish(reward_msg)
```

The reward of each step can be retrieved from '/ur3/step_reward'.

catkin_ws/src/openai_ros/openai_ros/src/openai_ros/task_envs/ur3/ur3.py

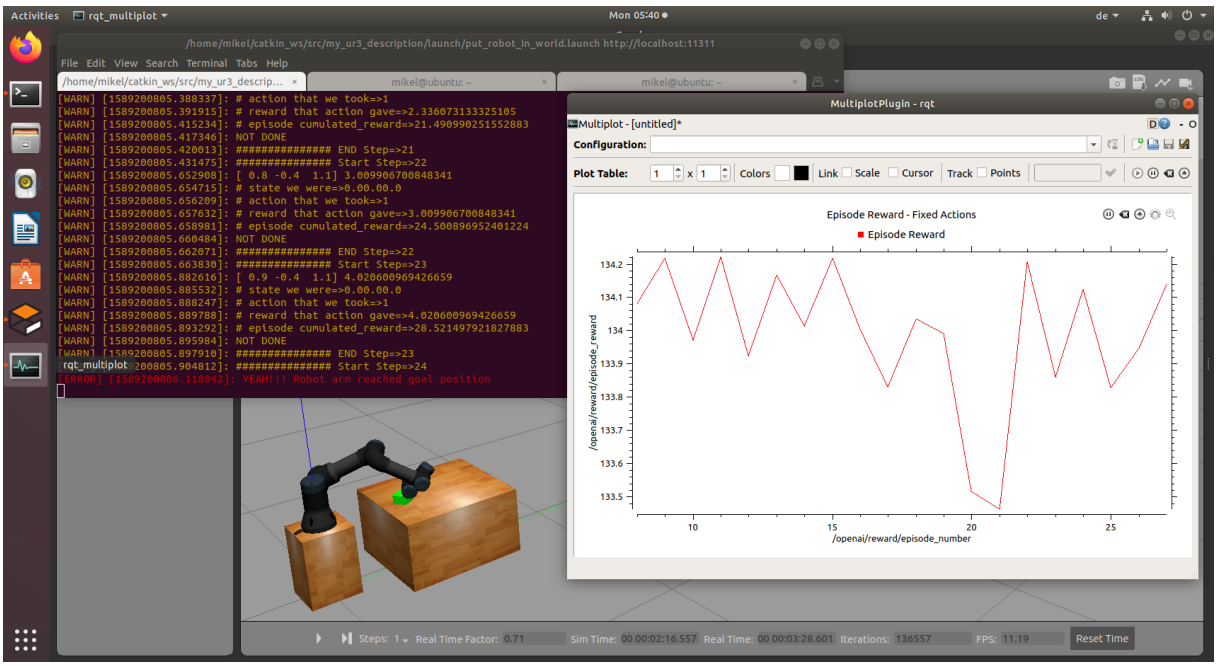
```
def _set_init_pose(self):

    self.check_publishers_connection()

    # Reset Internal pos variable
    #
    # Use .copy() to prevent overwriting self.init_pos
    self.init_internal_vars(self.init_pos.copy())
    self.move_joints(self.pos)
    rospy.sleep(2)
```

The .copy() ensure that after each episode the arm correctly moved back to its init position. Without .copy() the init position of the arm will be overwritten with each steps actual self.pos.

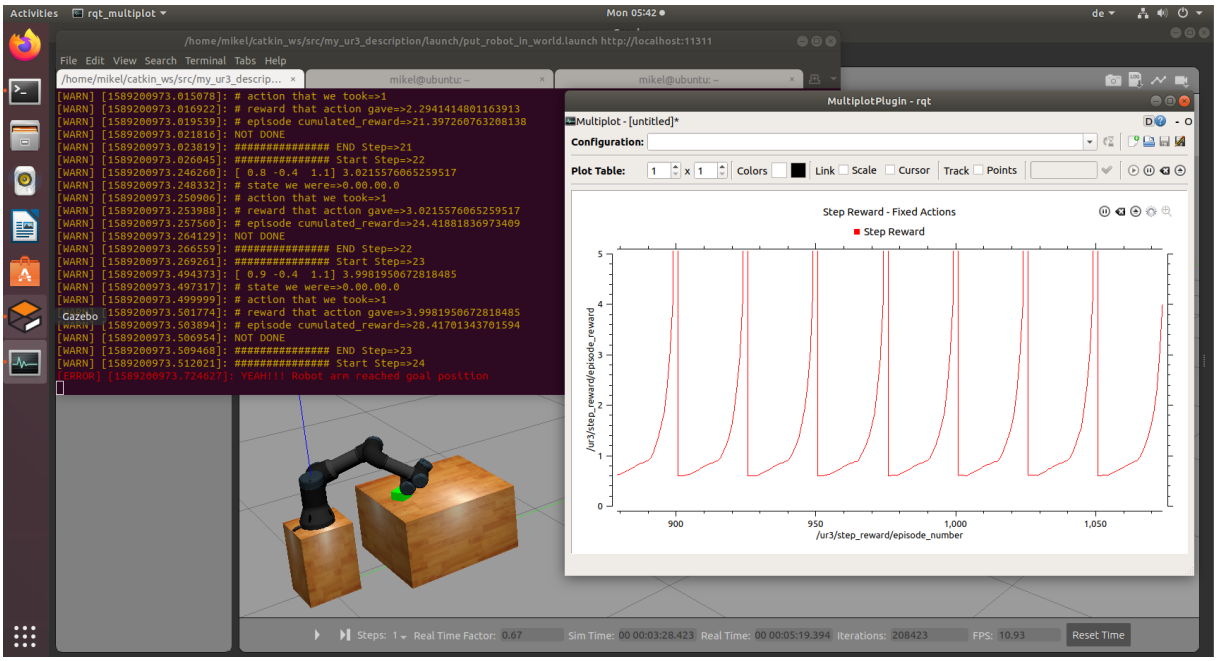
Sample Screenshots

Fixed Actions:

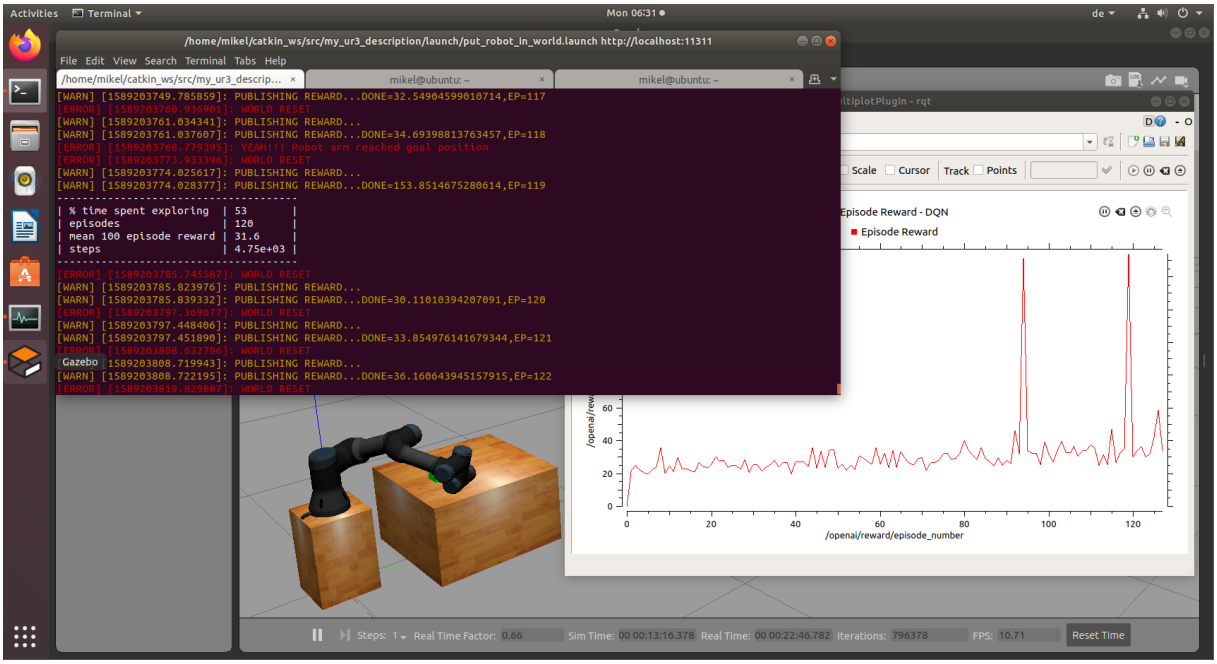
Episode Reward



Step Reward



DQN


Activities
Terminal
Mon 06:31
/home/mikel/catkin_ws/src/my_ur3_description/launch/put_robot_in_world.launch http://localhost:11311
File Edit View Search Terminal Tabs Help
[WARN] [1589203749.785859]: PUBLISHING REWARD...DONE=32.54904599010714,EP=117
[ERROR] [1589203760.936981]: WORLD RESET
[WARN] [1589203761.034341]: PUBLISHING REWARD...
[WARN] [1589203761.037607]: PUBLISHING REWARD...DONE=34.69398813763457,EP=118
[ERROR] [1589203768.779395]: YEAH!!! Robot arm reached goal position
[ERROR] [1589203773.933396]: WORLD RESET
[WARN] [1589203774.025617]: PUBLISHING REWARD...
[WARN] [1589203774.028377]: PUBLISHING REWARD...DONE=153.8514675280614,EP=119
% time spent exploring	53
episodes	120
mean 100 episode reward	31.6
steps	4.75e+03
[ERROR] [1589203785.745587]: WORLD RESET
[WARN] [1589203785.823976]: PUBLISHING REWARD...
[WARN] [1589203785.839332]: PUBLISHING REWARD...DONE=30.11010394207091,EP=120
[ERROR] [1589203797.369077]: WORLD RESET
[WARN] [1589203797.448406]: PUBLISHING REWARD...
[WARN] [1589203797.451890]: PUBLISHING REWARD...DONE=33.854976141679344,EP=121
[ERROR] [1589203808.632796]: WORLD RESET
[Gazebo] [1589203808.719943]: PUBLISHING REWARD...
[WARN] [1589203808.722195]: PUBLISHING REWARD...DONE=36.160643945157915,EP=122
[ERROR] [1589203819.829807]: WORLD RESET
Scale
Cursor
Track
Points
Episode Reward - DQN
Episode Reward
/openai/reward/episode_number
Steps: 1
Real Time Factor: 0.66
Sim Time: 00 00:13:16.378
Real Time: 00 00:22:46.782
Iterations: 796378
FPS: 10.71
Reset Time