Control of Robotic Manipulators using Q-Learning

Kabilan N, Sabarishwaran G, Gokul prazath S

Centre for Computational Engineering and Networking (CEN),
Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Coimbatore.

Introduction:

There are different robotic manipulators based on their requirements, some of them are Cartesian coordinate robot, cylindrical robot, PUMA Robot, Polar Robots, KUKA robots and so on. Control of robotics is one of the vast areas for research. With the advancements in modern technologies in artificial intelligence, such as deep learning, and recent developments in robotics and mechanics, both the research and industrial communities have been seeking more software-based control solutions using low-cost sensors, which have less requirements for the operating environment and calibration. A useful robot is one that is able to control its movement and the interactive forces and torques between the robot and its environment. Controlling a system requires the availability of a mathematical model and some sort of intelligence to act on the model. The mathematical model of a robot is obtained from the basic physical laws governing its movement. Intelligence, on the other hand, requires sensory capabilities and means for acting and reacting to the sensed variables. These actions and reactions of the robot are the result of controller design. Robotic manipulators can also be learned to move on their own provided the available constraints using Reinforcement learning. In this project we used Q-Learning to control a 2R Robotic manipulator.

Objective:

Control the trajectory of end-effector of the 2R manipulator using Q-Learning

Robotic Manipulator:

A Robotic manipulator is a device used to manipulate materials without direct physical contact by the operator. As robotic manipulators are inspired by the human arm kinematics, their end effectors may be inspired by the human hand autonomy. They have been used in a diverse range of applications including welding automation, robotic surgery and in space. Robots are programmed

in a way that can perform a series of actions automatically or semi-automatically. The main objective of Robotics is to create robots in the way it assists humans in various ways. They are capable of doing work faster with higher accuracy compared to humans.
Artificial Intelligence can be applied to robots to make them intelligent which can perform the task with their intelligence. With the help of AI algorithms robots can perform more complex tasks than usual. Robots are not always like the one shown in movies. Robots are mostly used for automating industrial works. Thus, they can be two link bots or just with a hand.

2R Robotic Manipulator:

Figure 1 shows the schematic diagram of a simple two-joint robotic arm mounted on a stationary base on the floor.

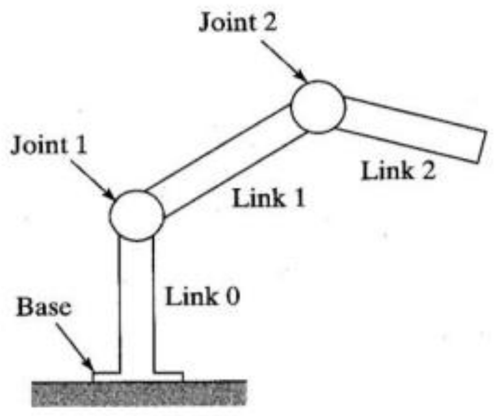


Fig 1: Simplified scheme diagram of mechanical components of a 2R robot.

Joints are similar to joints in the human body, which provide relative motion between two parts of the body. In the robotic field, each joint is in a different axis and provides an additional DoF of controlled relative motion between adjacent links, as shown in Figure 1. In nearly all cases, the number of degrees of freedom is equal to the number of joints.

1. An end-effector is an accessory device or tool which is attached at the end of the chain, and actually accomplishes the intended task. The simplest end-effector is the gripper, which is capable of opening and closing for gripping objects.

2. Links are the rigid or nearly rigid components that connect either the base, joints or end effector, and bear the load of the chain.
3. An actuator is a device that converts electrical, hydraulic, or pneumatic energy into robot motion.

Deep Reinforcement Learning:

Reinforcement learning is a subfield of machine learning, concerned with how to find an optimal behavior strategy to maximize the outcome through trial and error dynamically and autonomously. This autonomous self-teaching methodology is actively studied in many domains, like game theory, control theory, operations research, information theory, system optimization, recommendation system and statistics.

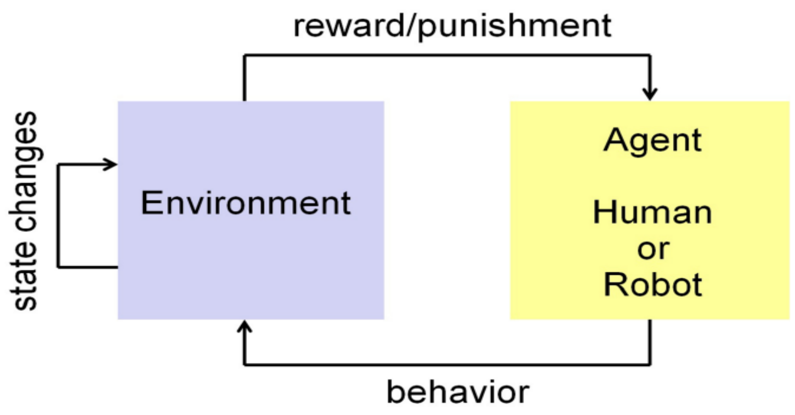


Fig 2: Universal model of reinforcement learning.

The method of reinforcement learning is inspired by learning through reward for every change in state in the environment. When an agent is in a state, it chooses an action according to its current policy and then it receives a reward from the environment for executing that action. By learning from this reward, it transitions to a new state, chooses a new action and then iterates through this process.

Deep Reinforcement Learning is among the most promising algorithms for this purpose because no predefined training dataset is required, which ideally suits robotic manipulation and control tasks. A reinforcement learning approach might use input from a robotic arm experiment, with different sequences of movements, or input from simulation models. Either type of dynamically generated experiential data can be collected, and used to train a Deep Neural Network (DNN) by iteratively updating specific policy parameters of a control policy network. Deep Reinforcement Learning is the combination of deep learning and Reinforcement Learning, combines both the technique of giving rewards based on actions from reinforcement learning, and the idea of using a neural network for learning feature representations from deep learning.

Deep Reinforcement Learning in Robotic Manipulator Control.

The main goal of DRL in context to robotic manipulator control is to train a deep policy neural network. The input of the network is the current state with information including position, velocity, acceleration, target pose, sensor information etc. The network trains with the information of current state and produces an optimal policy that gives out the optimal action to be taken in each actuator, such as torque or velocity. When the robotic manipulator accomplishes a task, a positive reward will be generated. With these delayed and weak signals, the algorithm is expected to find out the most successful control strategy for the robotic manipulation.

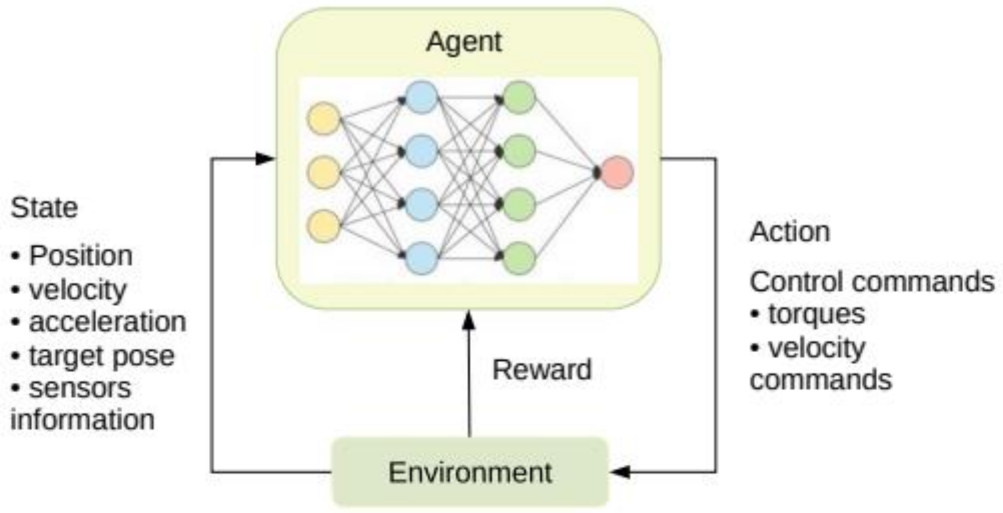


Fig 3: A schematic diagram of robotic manipulation control using DRL

Q-Learning

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It’s considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward. The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

When q-learning is performed we create a matrix called a q-table that follows the shape of [state, action] and we initialize our values to zero. We then update and store our q-values after an episode. This q-table becomes a reference table for our agent to select the best action based on the q-value. The next step is simply for the agent to interact with the environment and make updates to the state action pairs in our q-table Q [state, action].

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as exploiting since we use the information, we have available to us to make a decision.

The second way to take action is to act randomly. This is called exploring. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process. We can balance exploration/exploitation using epsilon (ε) and setting the value of how often you want to explore vs exploit.

The updates occur after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent. A terminal state for example can be anything like landing on a checkout page, reaching the end of some game, completing some desired objective, etc. The agent will not learn much after a single episode, but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values or q-star (Q∗).

3 Basic Steps in Q-Learning:

1. Agent starts in a state (s1) takes an action (a1) and receives a reward (r1)
2. Agent selects action by referencing Q-table with highest value (max) OR by random (epsilon, ε)
3. Update q-values

Terminologies in Reinforcement learning:

1. **Learning Rate:** lr or learning rate, often referred to as alpha or α, can simply be defined as how much you accept the new value vs the old value. Above we are taking the difference between new and old and then multiplying that value by the learning rate. This value then gets added to our previous q-value which essentially moves it in the direction of our latest update.
2. **Gamma:** gamma or γ is a discount factor. It's used to balance immediate and future reward. From our update rule above you can see that we apply the discount to the future reward. Typically, this value can range anywhere from 0.8 to 0.99.
3. **Reward:** reward is the value received after completing a certain action at a given state. A reward can happen at any given time step or only at the terminal time step.

Trajectory:

A trajectory or flight path is the path that an object with mass in motion follows through space as a function of time. Motion Planning would be the planned motion of a system to achieve a goal, this would have values even for a system at rest. Whereas Trajectory Generation would be the potential trajectories of a system, and when at rest would be zero.

Trajectory Planning:

Trajectory planning is moving from point A to point B while avoiding collisions over time. This can be computed in both discrete and continuous methods. Trajectory planning is a major area in robotics as it gives way to autonomous vehicles.

Trajectory planning is sometimes referred to as motion planning and erroneously as path planning. Trajectory planning is distinct from path planning in that it is parametrized by time. Essentially trajectory planning encompasses path planning in addition to planning how to move based on velocity, time, and kinematics.

Robot Trajectory Control:

The trajectory of the end effector can be controlled using some controllers. One of the controllers used in our work is the PD controller. In PD controlled control system output varies in proportion to the error in the predicted trajectory as well as with the derivative of the error. This type of controller provides combined action of both proportional and derivative control action.

Methodology:

In order to design a controller for the robotic arm, it is essential to have a mathematical model for that arm.

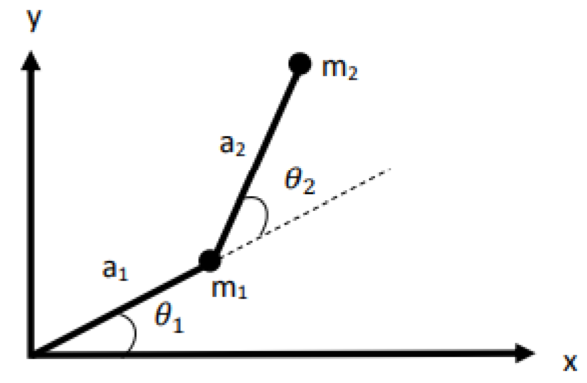


Fig 4: 2R Manipulator

Where m1 and m2 = Mass of first and second links respectively, a1 and a2 = Length of first and second links respectively. Using lagrangian formulation, dynamic of an n-joint robot manipulator with revolute joints can be formulated as:

$$\tau = B(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q)$$

$$B = \begin{bmatrix} a_1 + 2a_2\cos(q(1)) & a_2\cos(q(1)) + a_3 \\ a_2\cos(q(1)) + a_3 & a_3 \end{bmatrix}$$

$$C = \begin{bmatrix} -a_2\sin(q(1))[q(1)^2 + 2q(0)q(1)] \\ a_2\sin(q(1))[q(0)^2] \end{bmatrix}$$

$$G = \begin{bmatrix} a_4\cos(q(0)) + a_5\cos(q(0) + q(1)) \\ a_5\cos(q(0) + q(1)) \end{bmatrix}$$

•a1 (Total Moment of Inertia about origin)

•a2 (Torque about revolute Joint 1)

•a3 (Moment of Inertia on Link2)

•a4 (Total Gravitational Torque)

•a5 (Gravitational torque for Link2)

Where **B(q)** is the inertia matrix, **C (q, $\dot{q}$)** is the centrifugal & Coriolis force matrix, and **G(q)** gravitational torque matrix. To find the dynamic response of joint variables q & q_dot using inverse dynamics.

$$\ddot{q} = \mathrm{inv}(B(q))\,(\tau - C(q,\dot{q}) - G(q))$$

Using the $\ddot{q}$ from the equation, q and q_dot is updated using the updation equations given below.

$$q' = q + \left(\frac{\dot{q} * \text{step\_size} + \ddot{q} * \text{step\_size}^{2}}{2}\right)$$

$$\dot{q}' = \dot{q} + \ddot{q} * \text{step\_size}$$

We aggregate all the state features in a state structure, it contains q and q_dot values of each horizon (We are limiting it to finite horizon problems. So, we limit it to 10) along with Kp (proportional gain), Kd (derivative gain), a_position (current position) and a_velocity (current velocity).

State = (q, q_dot, Kp, Kd, a_position, a_velocity)

An experimental trajectory data is dynamically using the double differentiated equation of $\ddot{q}$ to train a Deep Neural Network (DNN) by iteratively updating specific policy parameters of a control policy network. Finding $\ddot{q}$ for all horizons with the function given below which return updated $\ddot{q}$.

$$\ddot{q}_d(t) = [-\pi\cos(st)\dot{s}^2 - \pi\sin(st)\ddot{s} - \pi\sin(st)\dot{s}^2 + \pi\cos(st)\ddot{s}]$$

Q-Network, a feedforward network is created with 2 layers: hidden layer and output layer. With tanh activation for the hidden layer loss function used is loss function and optimized using gradient descent optimization.

Reward

Here if the error is more our reward will tend to 1 if it is less it will tend to -1 in order to make our robot direct in the correct direction, we have to maximize negative reward. Actions are incremented in a discrete fashion Position & velocity are incremented independently with position [+5, +3, 0, -3, -5] and velocity [+1, +0.5, 0, -0.5, -1]

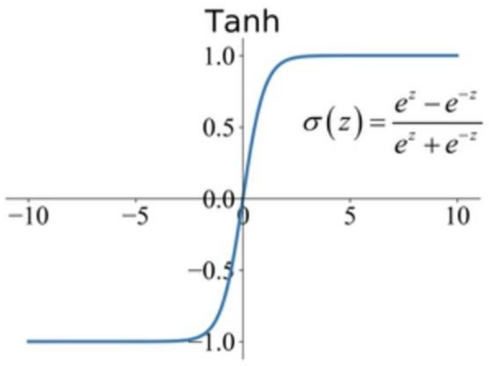


Fig 5: tanh activation function

$$r(s) = e^{-\|e_q\|^2} + e^{-\|\dot{e}_q\|^2}$$

Gain Saturation:

Since, it is difficult to cover the exact dynamic model. Then the closed loop controller ensures the dynamic errors. Kp is constrained to values between 10 and 200. Kd is constrained to values between 2 and 40.

Updating

The state variables are extracted from the current state, reward is calculated for time t, Kp and Kd are updated based on the current position and velocity of the trajectory. auxiliary control input (u) is calculated using the equation

$$u = K\_p(q(t) - q) + K\_d(q'(t) - q)$$

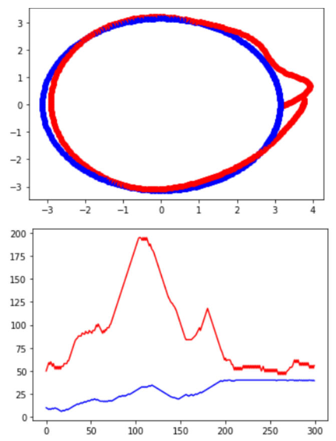
q and q_dot are updated using the forward dynamics. The updated values are updated in the current state.

The future trajectory is calculated for a particular horizon using q_double_dot. The optimal combination of velocity and position is predicted using the Q-Network. The state variables are updated using forward dynamics to get positional parameters of the next point.

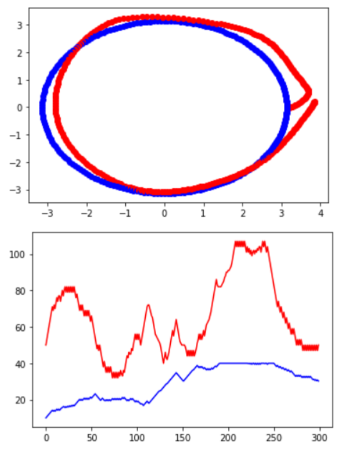
Epsilon value is updated for every 3 episodes, as initially exploration is favored more and gradually the framework gives more importance to exploitation. The error is calculated between the original trajectory and the predicted trajectory using mean square error. The model is trained to reduce that error.

The output of the RL framework is showcased below. This clearly shows the agent improved itself and converges to the required trajectory. This runs for 50 epochs with initial epsilon value as 0.9 favoring exploration.

Output:



```
Episode : 37------Total Loss : 798.891253971341------Epsilon : 0.008720010936629989
```



```
Episode : 50------Total Loss : 485.31193596515044------Epsilon : 0.00209367462588486
```