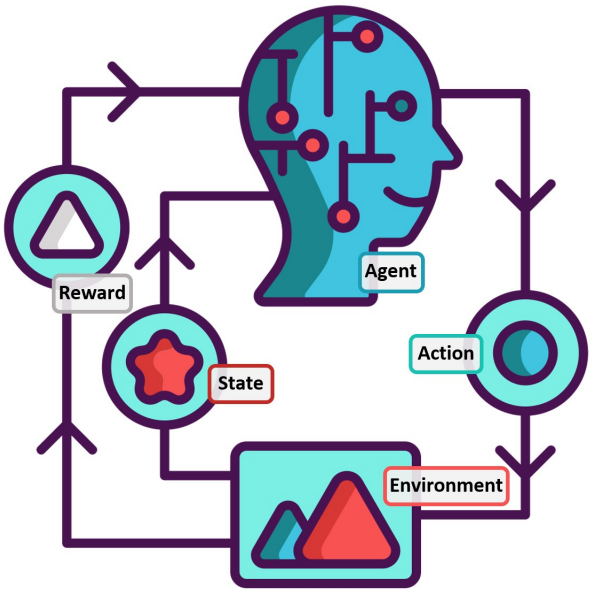
Pepper Harvesting Reinforcement Learning Overview

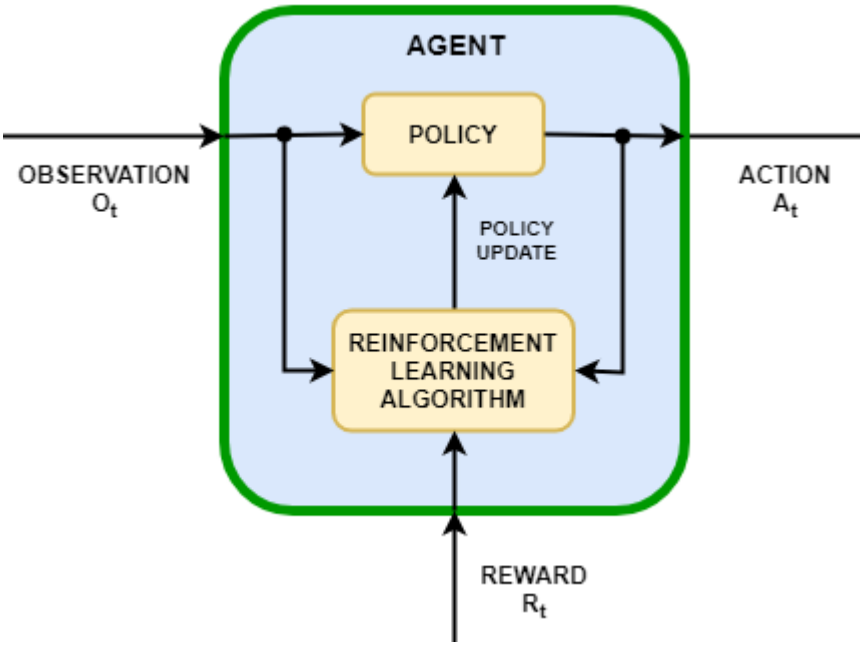
Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Fundamentals of Reinforcment Learning

The essential components of Reinforcement Learning (RL) are perception, action and goal. The idea is that an agent, having learned from past experiences, understands which actions will lead to maximize a numerical result (reward) in a given time, for any given situation (state). The typical challenge of many RL problems is to maximize not only the best current reward, but also the best future ones, which can be obtained by influencing future states through actions. The agent must be able to perceive the state of the environment and take actions that will affect it, trying to reach one or more goals related to the accomplishment of a predefined task. RL takes into account two aspects: exploitation and exploration. The first aspect consists in exploiting what the agent already knows to obtain reward, the latter means exploring other actions in order to make better selections in the next steps. Looking for a trade-off between exploitation and exploration is one of the most challenging problems concerning RL. A strategy that favors the exploitation at the cost of exploration is called greedy.



The agent contains two components: a policy (π), and a learning algorithm.



The policy is the probabiliy that the agent will perform a given action while residing in the current state. Depending on the task, the policy can be represented as a simple function or lookup table, or as a more complex deterministic or probabilistic function.

The learning algorithm continuously updates the policy parameters based on the actions, observations, and rewards. The goal of the learning algorithm is to find an optimal policy that maximizes the expected cumulative long-term reward received during the task.

The environment may be described simply as the physical world in which the agent operates. At each moment in time the agent and environemnt can be models as a state, containing all relevent information about the system. Each action performed by the agent pertubs the environment, thus changing the state. Prior to advancing to the next time step, the agents recieves a $reward$ indicating quantatively the effect the action had on the state.

The reward is a scalar feedback signal that indicates the performance of the agent during the previous iteration. The agent's ultimate goal is to maximize the expected cumulative reward it receives over time.

The expected cumulative reward the agent can accumulate in the long run, starting from a certain state and following the policy, is called the value function.

Manipulator Controlled Reinforcement Learning with Object Avoidance

Firstly, one needs to define the setup for the RL framework, that consists in building the state space S, the action space A and the reward function.

The state space of a robotic manipulator consists of the following relevant observations for training:

- Measured joint positions q
- Joint velocities $\dot{q}$
- Target point position p_t
- end-effector position p_e
- obstacle position p_o

The state space is thus defined as

$$S = \{q, \dot{q}, p_t, p_e, p_o\}$$

The action space may be defined as the following, containing the vector of rotational velocity at each joint. Specifically, the velocity control of the joints is such that, at each step, the joint has to rotate at the target velocity using the maximum torque available.

$$A = \{\dot{q}_{tar}\}$$

Finally the reward function is defined as a weighted sum of three terms: the distance between the endeffector and the target point, the magnitude of the actions, and the distance of the obstacle from the robot.

$$r = c_1 R_T + c_2 R_A + c_3 R_O$$

The distance R_T between the end-effector and the target point is computed using the Huber-Loss function,

$$R_T = \begin{cases} \frac{1}{2}\delta & \text{for} |d| < \delta \\ \delta(|d| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

where d is the Euclidean distance between the tip and the target and δ is a parameter that determines the smoothness.

The magnitude of the actions R_A performed by the manipulator is computed as the square of the norm of action vector a, as follows

$$R_A = -||a||^2$$

It gives a negative contribute to the reward, therefore smaller actions are encouraged. The distance between the robot and the obstacle R_O is computed as follows

$$R_O = (\frac{d_{ref}}{d_O - d_{ref}})^p$$

where d_{ref} is a constant parameter so that $0 < R_O < 1$, d_O is the minimum distance from the obstacle as computed by the simulation and p is for the exponential decay of negative reward. The weights c_1, c_2, c_3 are necessary in order to tune the reward function depending on what is prioritized in the training process.

Learning Algorithms

Q-Learning

Q-learning is an off-policy algorithm that directly approximates the optimal action-value function Q^* independently of the policy being followed. While selecting the action, the agent can carry on exploring the environment with probability ϵ, by randomly choosing an action and ignoring its previous knowledge. Q-learning can however become infeasible in case of continuous action problems with a large number of states. The problem could be overcome with the use of a parametric approximator of the action-value function, or Q-function. One way to build such approximator is a Deep Neural Network (DNN).

Normalized Advantage Function

Normalized Advantage Function (NAF), is a method that allows one to enable Q-learning in continuous action spaces with deep neural networks. The idea behind this method is to design the Q-function in a way that its maximum $\text{argmax}_a Q(s_t, a_t)$ can be easily computed during each update.