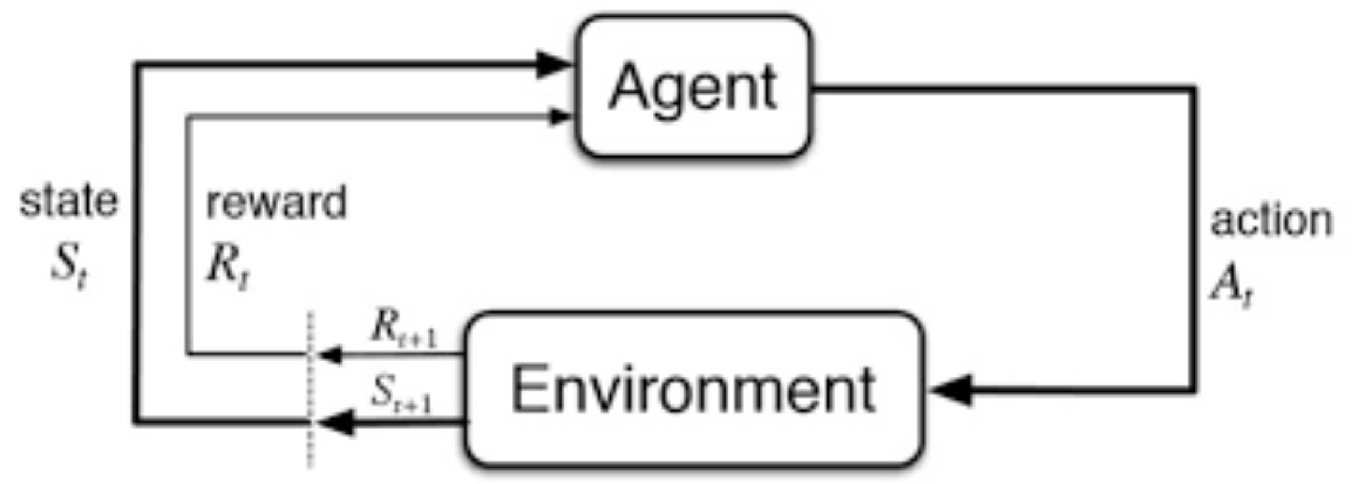
Reinforcement Learning in Drones

Karthik.P.B, Keshav Kumar, Mentor: Vikrant Fernandes

Abstract :

Reinforcement learning is a technique in which the agent is unaware of the environment it needs to act in and hence take some random actions and gets information about the environment using a reward system that is given by environment based on agent's actions. An episode is defined as reaching terminal state or the agent is out of observation space.

There are several techniques of reinforcement learning that are monto-carlo and temporal difference. Temporal difference can be implemented using SARSA(On Policy method) or Q-Learning(Off policy method). Here, in the project we have used Q-Learning.



Q-Learning :

The action value function defines how good is that action is for a given state of agent. while learning, it is updated using **Bellman's equation** given by :

$$Q_{new}(s_t, a_t) = (1-\alpha)[Q_{old}(s_t, a_t)] + \alpha[R_t + argmax\, Q(s_{t+1}, a_t)]$$

In Q-Learning, unlike SARSA, based on experiences the agent gets greedy gradually i.e. it always goes for immediate maximum reward.

To maintain exploration and exploitation strategy, we use epsilon greedy policy.

Path planning using Q - Learning :

The regular trajactory planning algorithms just depend on the intial position, final position, intial velocity and final velocity. But in reinforcement learning, the agent takes random actions and visits most of the points in space and then finds obstacles and try to avoid them in next episodes by which it learns to go to a target point.

Path planned with some obstacles :

In fig.1, path is planned with some obstacles, now three points from these waypoints are made obstacles that are [1,3,28], [2,3,28], [3,3,28] and fig.2 shows path planned excluding the above points.

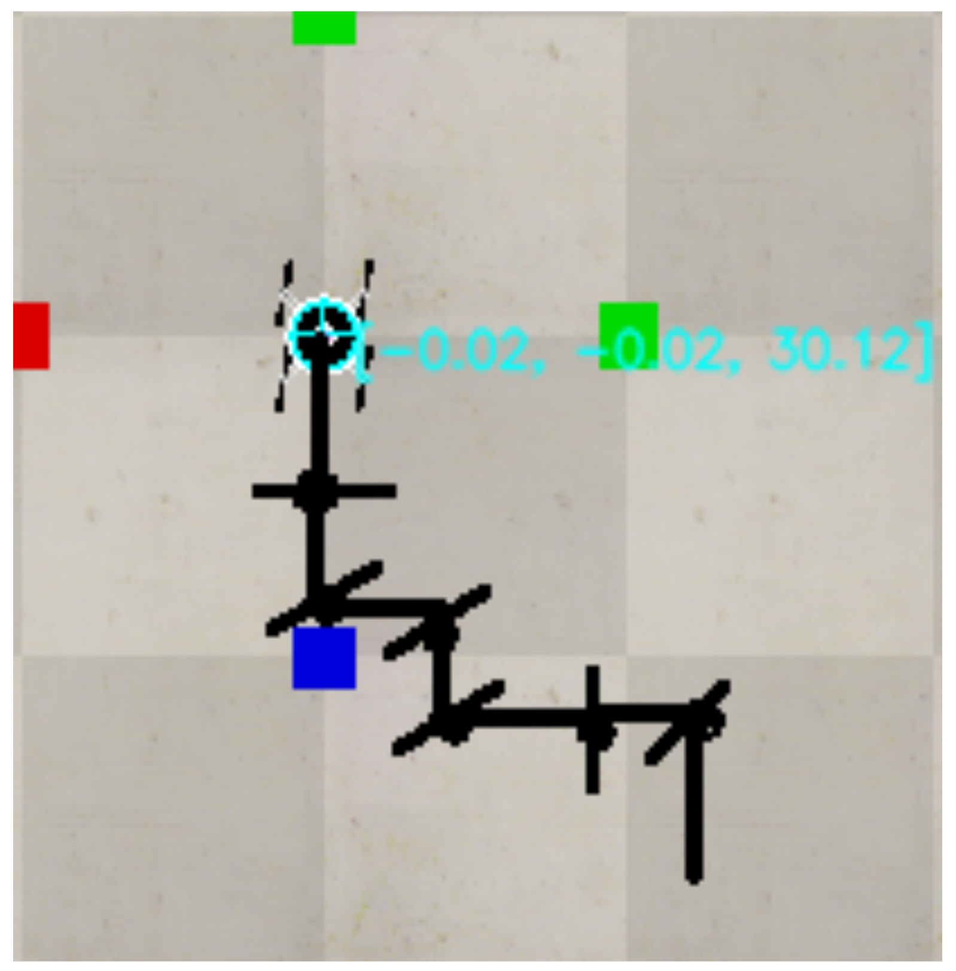


Figure.1

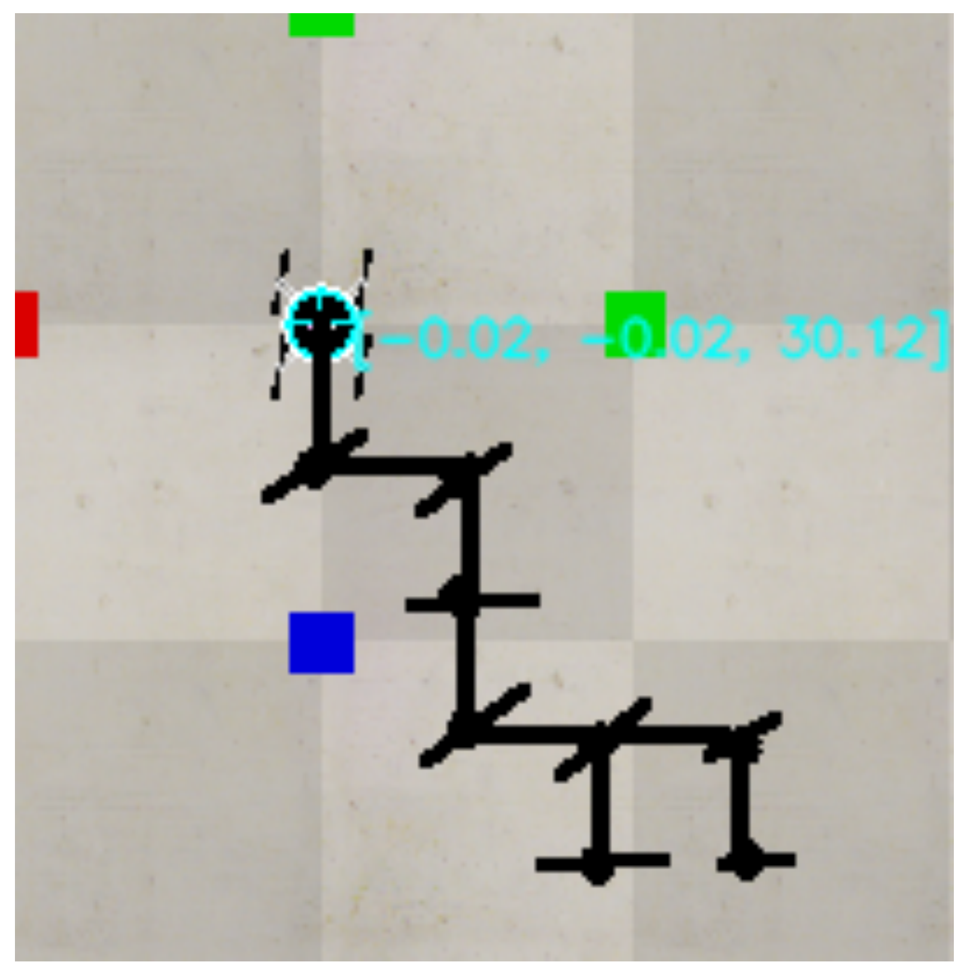
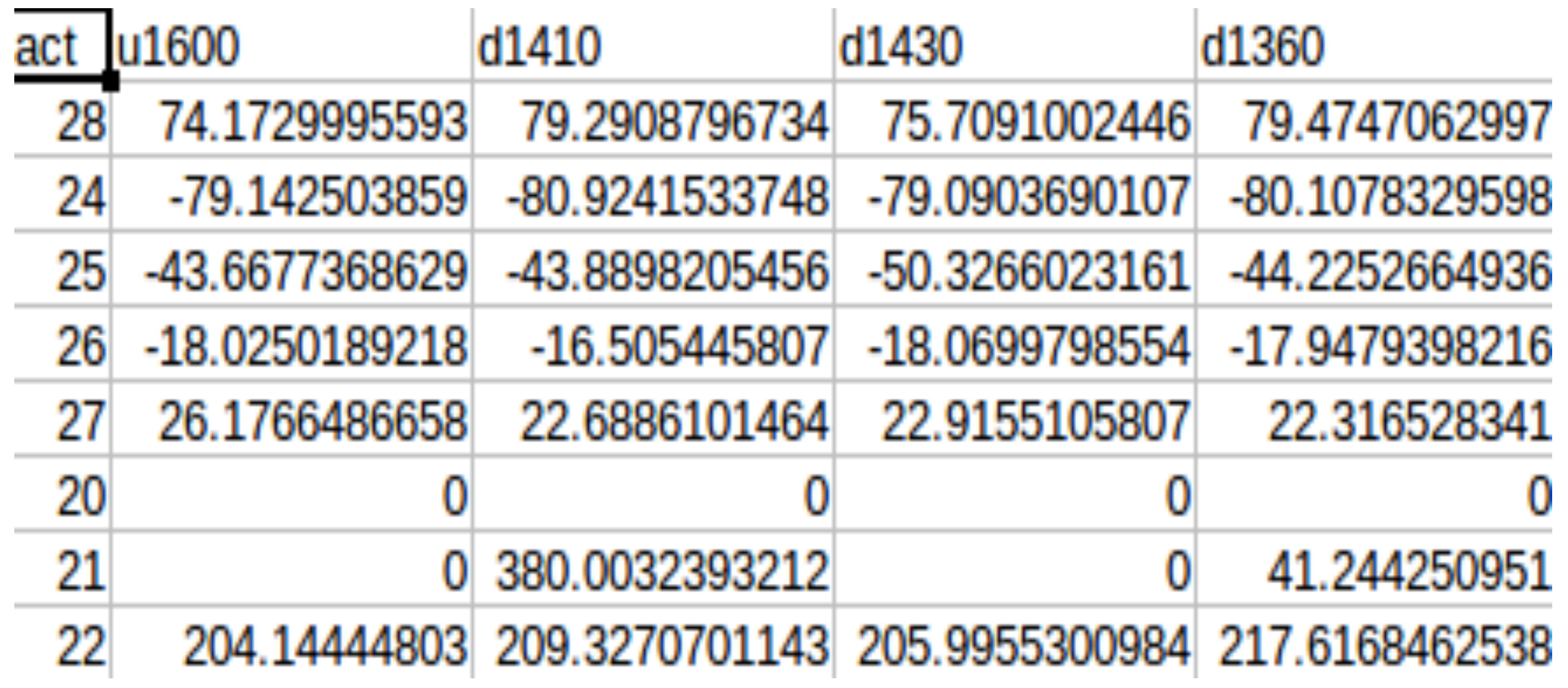


Figure.2

Position hold using Q - Table method :

While learning, a Q-table is constructed as shown in figure below which maps from states to actions. While learning these values are updated using Bellman's expectation equation.

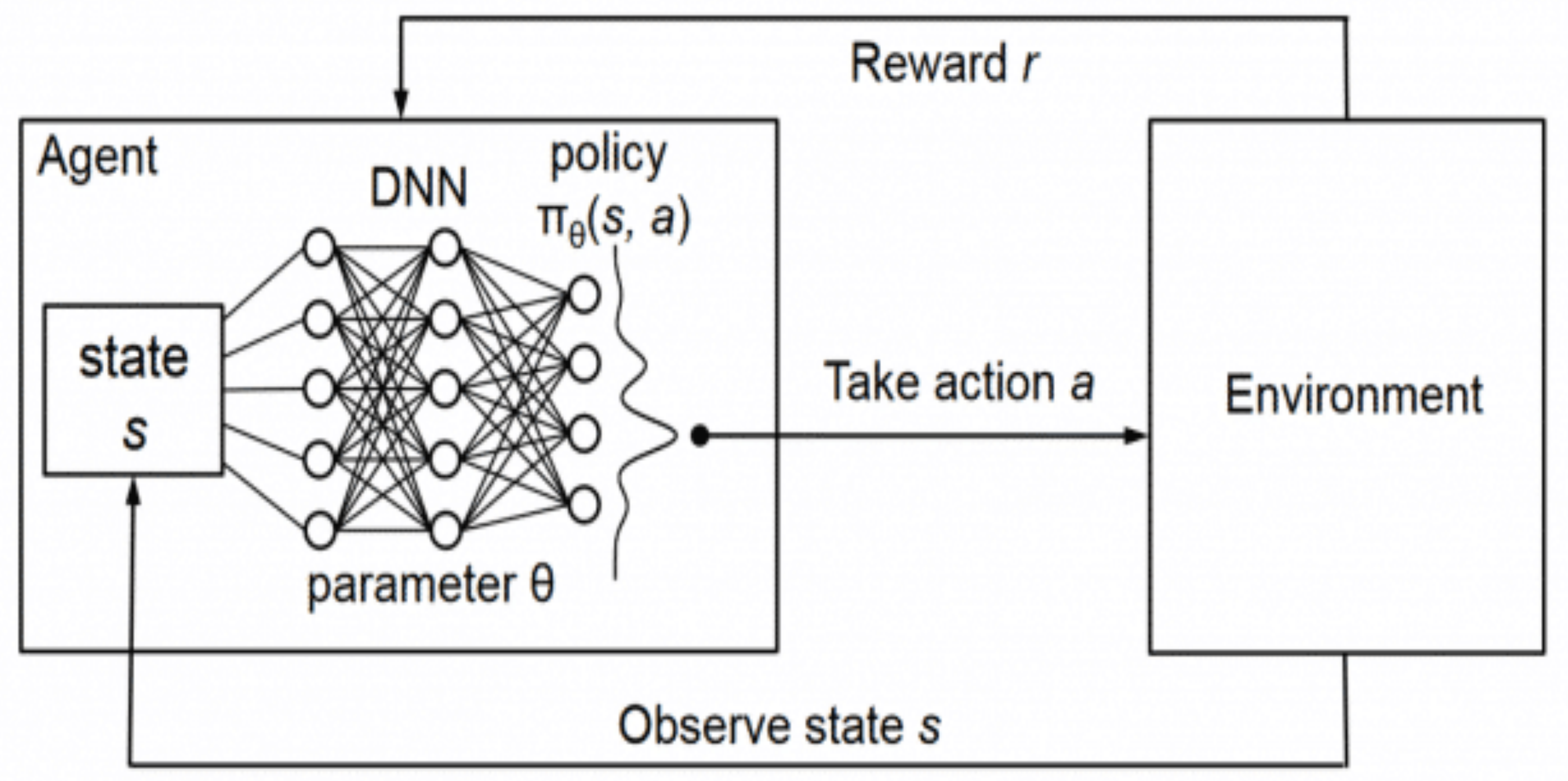
After learning, the agent(Drone) gets what state it is in currently and takes that action which has maximum value for that state. First column is z coordinates and first row is actions



act	u1600	d1410	d1430	d1360
28	74.1729995593	79.2908796734	75.7091002446	79.4747062997
24	-79.142503859	-80.9241533748	-79.0903690107	-80.1078329598
25	-43.6677368629	-43.8898205456	-50.3266023161	-44.2252664936
26	-18.0250189218	-16.505445807	-18.0699798554	-17.9479398216
27	26.1766486658	22.6886101464	22.9155105807	22.316528341
20	0	0	0	0
21	0	380.0032393212	0	41.244250951
22	204.14444803	209.3270701143	205.9955300984	217.616846253

Position hold using Deep Q Networks :

Figure below shows deep Q networks for continuous observation space. It helps in avoiding oscillations. This is basically like approximating the Q table with a Neural Network.


Reward r
Agent
DNN
policy
$\pi_\theta(s, a)$
state
s
parameter θ
Take action a
Environment
Observe state s