CSE 410/510 Course Project Final Report

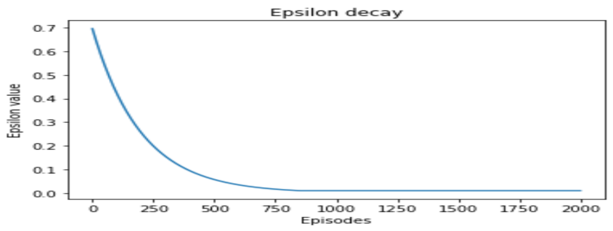
Rugved Hattekar, Nanditha Nandanavanam, and Chen Zeng[*]

I. Main motivation of the project

Machine learning techniques are becoming an essential tool in the applications of autonomous UAVs in sensing, control, path planning, mission management, etc. Among the applications, using learning model to takeover the control of the aircraft is a field awaiting explorations. This project aims to apply a powerful reinforcement learning model of DQN on Unity designed naive drone simulator which is a commercial platform to build a dynamics and control model for a quadcopter for the problem of takeoff and hovering a Drone simulator based on Unity platform.

II. Milestone progress: Learning on the lunar lander problem

For the preliminary implementation of the drone learning to fly, we used Lunar Lander environment to test our DQN implementation. Landing the agent was accomplished using DQN algorithm. The learning rate was set to 0.001, number of episodes was set to 5000, the epsilon was set to 0.7 initially with a decay factor of 0.995 over time to gradually explore less and exploit more. The discount factor was set to 0.98. The batch size for experience replay memory was set to 32. The following were the plots obtained.



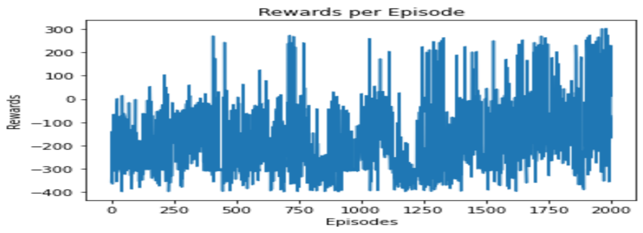


Fig. 1 Epsilon decay, Average loss and Rewards over episodes.

A. Robot Operating System Package

Robot Operating System or ROS is a primary communication mechanism between the Quad-copter controller and the simulation. It is a collection of tools, libraries and conventions that need to be installed in order to send or receive control,state, planning, actuator information messages for various quadcopter functionalities. Functional modules are denoted as ROS nodes. These ROS nodes exchange messages called as ROS topics with each other. The ROS node sending ROS topics are called publishers and the ROS nodes receiving them are called as subscribers.

B. Simulation environment

The simulation environment is based on the ROS packages and a simple PID controller which is used to control the speed of the 4 motors using a feedback signal and is designed by Unity for testing of simple RL algorithms. The ROS Architecture between the nodes is shown below. The rl_controller node is basically the controller which is in constant connection with the IMU node of the simulator, the IMU node (I) is publishing the current position of the Drone and the current velocity and the acceleration. The action space of the agent is basically the thrust(Force) it needs from the four rotors to lift the chassis to the required height. So once the velocity, acceleration and position is achieved from the IMU(I) Node from the simulator, force and velocity is calculated to reach the maximum height as a loss. The new velocity and acceleration is then implemented and sent back to the rl_controller node. We provide thrust force to the environment as an action, the action space is a n dimensional box with independent limits for each axis. The action space is basically action = [force_X,force_Y,force_Z,Torque_X,Torque_Y,Torque_Z] hence the action space is defined as Box(6,). The action space we considered is only force in X,Y,Z direction as for this project. With maximum force in

[*]Department of Mechanical and Aerospace Engineering, Univeristy at Buffalo.

Z-Direction to be 25N. Similarly, the observation space is continuous in 3-Dimensions in x,y,z. We are sampling from the observation space to make the implementation easier and considering observation space in each direction as 50m maximum. Although we only need 10m in Z-direction for this particular project.

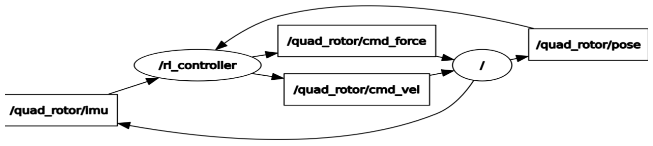
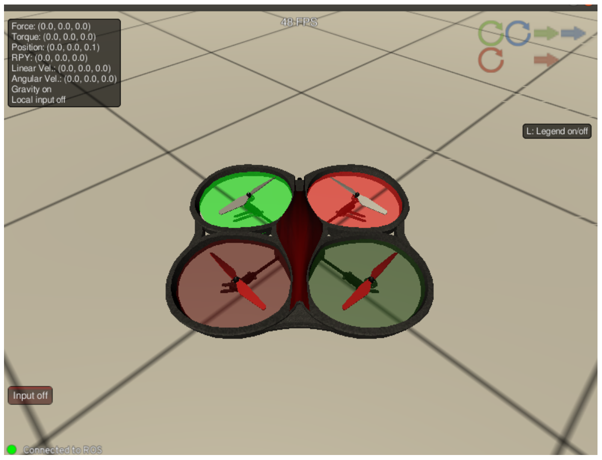
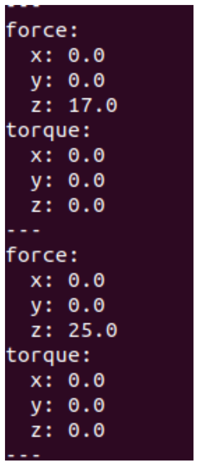


Fig. 2 Environment Architecture using ROS



(a) Simulation Environment



```
force:
  x: 0.0
  y: 0.0
  z: 17.0
torque:
  x: 0.0
  y: 0.0
  z: 0.0
---
force:
  x: 0.0
  y: 0.0
  z: 25.0
torque:
  x: 0.0
  y: 0.0
  z: 0.0
---
```

(b) Change of Thrust Force as DQN is implemented

Fig. 3 Simulator

C. DQN Algorithm

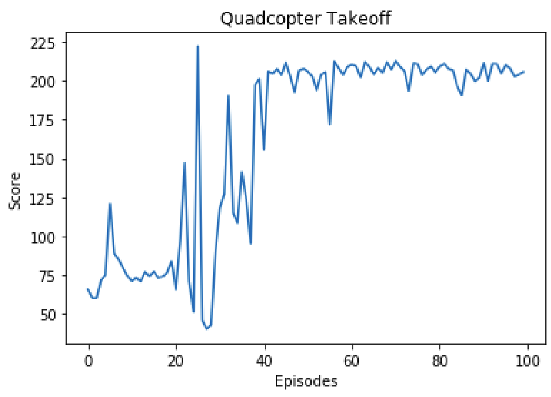
DQN algorithm is used for the learning reinforcement method. The neural network consists of 2 hidden layers with 32 and 64 nodes each activated through relu function. The Q learning process will update the target values of the deep Q network with ones generated from applying Bellman equation. During the training process, the network tries to predict actions that get better rewards by optimising Mean Squared error loss function. The learning rate was set to 0.001, number of episodes was set to 64, the epsilon was set to 1 initially with a decay factor of 0.995 over time to gradually explore less and exploit more. The discount factor was set to 0.9. The batch size for experience replay memory was set to 64.

III. Implementing Takeoff

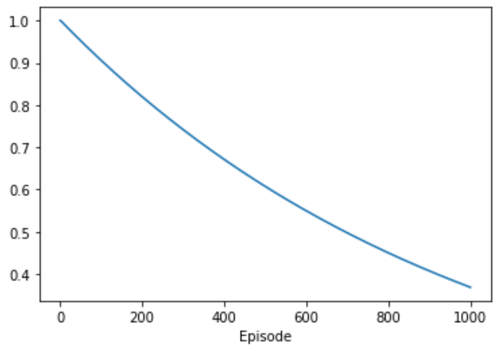
The quadcopter is trained to successfully lift off from the ground and reach a target height. The target height is to set 10 m above ground in z axis. The current position of quadcopter is subscribed from the simulation and is fed to the DQN algorithm. The DQN Algorithm then gives out the action which is a thrust force required to achieve by the four rotors. The reward function is the negative absolute distance from the quadcopter position in z axis till the target height. So, when the target height reached yields a zero and as the quadcopter is farther, the reward is negative. On reaching the target height or going beyond the height yields a bonus of +10 and the episode ends. There is a penalty of -10 if the quadcopter has still not reached the target height after a duration of 5 s has elapsed and then the episode is terminated. The action is a vector of force and torque that are published to simulation as a control command that affects that quadcopter's velocity.

IV. Results of takeoff implementation

We implemented DQN on the ROS Based simulator and the reward score has been plotted in the figure below. The reward score has a lot of fluctuations in the early iterations which explains the exploration phase of the drone, once it has achieved the higher reward it is visible that the reward the drone starts taking greedy actions and score becomes constant.



(a) Drone Reward



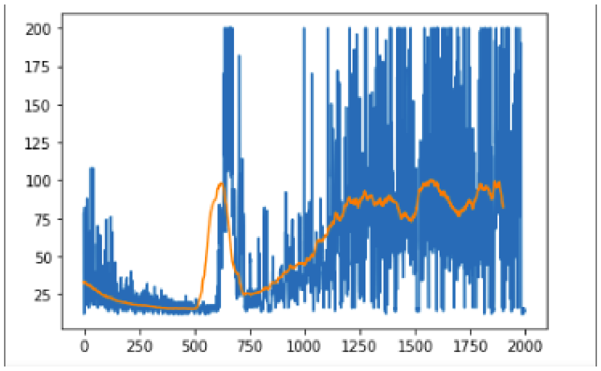
(b) Epsilon Decay

Fig. 4 Take Off DQN Implementation Results

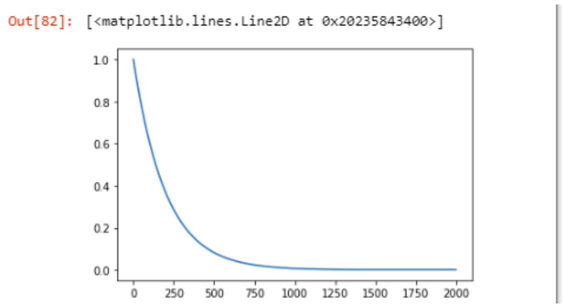
V. Implementing Hovering

The quadcopter is trained to successfully hover around a target height. The target height is to set 10 m above ground in z axis. The current position and linear acceleration of quadcopter are subscribed from the simulation and are fed to the DQN algorithm. The DQN Algorithm then gives out the action which is a thrust force required to achieve by the four rotors. The simulation environment also provides a gravity model that gives a negative thrust. So, initial velocity of 1.2 m/s is given to overcome this. The reward function is the negative absolute distance from the quadcopter position in z axis till the target height minus the linear acceleration. If this value is less than the threshold of 3 m, that yields a bonus of 10. There is a penalty of -10 if the quadcopter has not reached the hovering threshold after a duration of 5 s has elapsed and then the episode is terminated. The action is a vector of force and torque that are published to simulation as a control command that affects that quadcopter's velocity.

A. Hovering Implementation Results



(a) Drone-Hovering Task Reward



```
Out[82]: [<matplotlib.lines.Line2D at 0x20235843400>]
```

(b) Drone Hovering Epsilon Decay

Fig. 5 Hovering DQN Implementation Results

VI. Conclusion

In this project we implemented the DQN RL Algorithm for the purpose of to learn the drone fly based on the thrust for as an action. The environment which we used doesn't require the angular velocity of the rotor but it takes into consideration the total thrust force applied, hence the action space is consists of the thrust force within the limit of 25N. The ROS Architecture here is used to interact with the environment and for each timesteps. The ROS Architecture helps us to get the pose (X,Y,Z) co-ordinates of the from the simulator and the linear acceleration at the current timestep. Which is further analyzed in the DQN Framework and appropriate action is suggested. This project only takes into consideration the thrust forces along z-direction. Hence, In this project we learnt how the ROS environment can be used to test the Reinforcement Learning Algorithms. Also ROS framework is a solid framwork to implement complex and customized environements. The videos of the learning process of the drone has been provided in the submission folder.

References

1. https://medium.com/@jonathan$_hui/rl - dqn - deep - q - network - e207751f7ae4$

2.$https : //towardsdatascience.com/cartpole - introduction - to - reinforcement - learning - ed0eb5b58288$

3.$IntroductiontoReinforcementLearningbySuttonandBurto.$

4.$https : //gym.openai.com/envs/MountainCar - v0/$

5.$https : //gym.openai.com/envs/CartPole - v0/$