Reinforcement Learning for Altitude Hold and Path Planning in a Quadcopter

Karthik PB
Dept. of Electronics and Communication
PES University,
Bengaluru, India
e-mail: karthikpk23@gmail.com

Keshav Kumar
Dept. of Electronics and Communication
KIET Group of Institutions,
Ghaziabad, India
e-mail: dipsitekeshav22@gmail.com

Vikrant Fernandes
eYantra
Indian Institute of Technology, Powai
Mumbai, India
e-mail: vikrant.ferns@gmail.com

Kavi Arya
eYantra
Indian Institute of Technology, Powai
Mumbai, India
e-mail: kavi@cse.iitb.ac.in

***Abstract*—The control and stability of drones is a challenging problem. There is need for a more dynamic and robust control that the drone can use to adjust itself to an unknown environment directly. This paper presents a framework for using reinforcement learning to control altitude of a drone. We use PID to stabilize *x* and *y* axis of the drone. The drone is trained using Q-learning of Reinforcement Learning in a simulated environment. The trained model is then tested in the real world. Furthermore, a comparative analysis of reinforcement learning and PID algorithm is presented. Finally, an application of way-point navigation from one given point to other in an environment filled with obstacles at different points formulated as a 3-dimensional grid is presented using Q-learning of Reinforcement Learning.**

Keywords-reinforcement learning; Q-learning; ROS; PID; real world simulation; V-REP; path planning; navigation; localization; whycon; nano drone

I. Introduction

Reinforcement Learning (RL) has been successful in solving many complex problems. They are being extensively used in Atari and various other computer games to teach computers to win the game [1]. In a traditional RL approach, an agent takes actions on its environment and earns rewards based on the outcome of that action. Formulating a control problem in RL terms is a difficult task, but it is advantageous over traditional optimal control as initially the agent does not require information on the environment. By taking random actions on the environment, the agent learns about the environment. This is deduced by its behavior to that particular action in that environment and the reward it gets based on the outcome of the action. This makes RL a more robust and flexible approach.

Training a RL model is time consuming, hence simulators with physics and robot models close to the real world are used for training purposes. The trained model is then implemented on a real world robot [2]. This approach is suitable over training the model on a real world robot, especially in the case of drones. In this study, nano drone Pluto X [3] developed by DronaAviation is the agent. An episode is defined as the amount of time the drone stays in the environment without getting damaged or loitering out of the frame. Learning in RL happens based on a reward system i.e. higher is the reward if it takes actions towards a set goal and lower is the reward if it takes actions that take it away from the set goal. As a result, the agents' priority is to maximize the reward.

There are two methods in reinforcement learning, model-based learning and model-free learning. Since the environment is not known, we use model-free learning method. When an episode ends, the drone sets to its initial position and the new episode starts. There are two techniques in model-free method, namely Monte-Carlo and Temporal difference. In Monte-Carlo, the learning happens after end of every episode based on average of all the observations in that episode. This is not suited for a drone, as every state it observes is important. Hence, temporal difference has been chosen in which the learning happens in each and every state. Again, there are two techniques that can be used, that are SARSA and Q-Learning [4]. SARSA is on policy and Q-Learning [5] is maximum reward irrespective of long term consequences. The basic block diagram of RL model looks as shown in Figure 1.

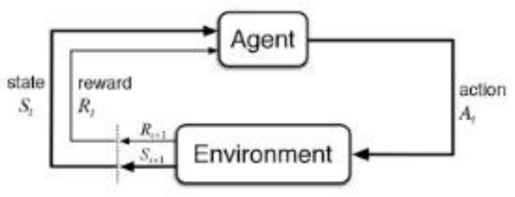
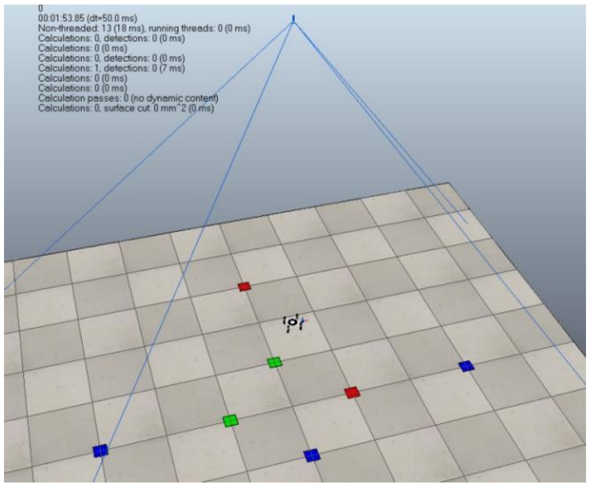


Figure 1 RL Block diagram.

For path planning, PID control [6] has been used to navigate from coordinate to coordinate as the main goal was to get navigation [7] points through RL, avoiding all the obstacles.

This paper is divided as follows: Section II discusses the experimental setup of the study. Section III talks about the problem formulation of holding pose and deciding way points as formulated in terms of reinforcement learning. Section IV discusses the technique of RL used to solve the problem. Finally we present our results in Section V. Conclusion and Future work is discussed in Section VI and VII respectively.



```
0
00:01:53.85 (dt=50.0 ms)
Non-threaded: 13 (18 ms), running threads: 0 (0 ms)
Calculations: 0, detections: 0 (0 ms)
Calculations: 0 (0 ms)
Calculations: 0, detections: 0 (0 ms)
Calculations: 1, detections: 0 (7 ms)
Calculations: 0 (0 ms)
Calculations: 0 (0 ms)
Calculation passes: 0 (no dynamic content)
Calculations: 0, surface cut 0 mm^2 (0 ms)
```

Figure 2 V-REP environment.

II. Experiment Setup

The project proposes a method for position hold and 3D path planning of the Pluto drone in a defined environment using RL. For this, the environment included standing pipes of various heights that acted as obstacles that the drone had to avoid. The environment objects and drone were localized using an overhead camera as shown in Figure 3. The camera had a resolution of 1920x1080 and ran at 30 frames per second (fps). The position of the Pluto drone was estimated using WhyCon [8] fiducial marker fixed over the drone. The observation states are values of *3D* Cartesian space in *x*, *y* and *z* in the whycon frame of reference. The position coordinate of the obstacles in whycon frame of reference were recorded for the RL algorithm before execution. A custom built ROS package for the pluto drone was used for communication over Wi-Fi.

Since training takes lot of time, the RL model cannot be directly trained on a real drone because of limited power supply and very high chances of damage. Hence, an environment is created in a simulation, setting its physics near to the real world and also defining dynamics of the drone that we use. Disturbances in the form of noise are added to the environment making it even more robust. The simulation has been done on V-REP [9] as shown in Figure 2 and the controlling of Pluto-X drone is done with the help of Robot Operating system(ROS) [10] which has been interfaced with the V-REP simulator so as to observe and manipulate the movements and task performed by the drone.

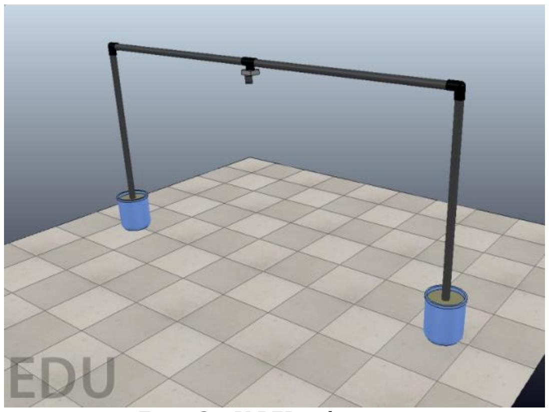


Figure 3. V-REP with camera.

III. Problem Formulation

In order to train the Reinforcement Learning model, an simulated environment for pluto drone was written in python as a class according to the OpenAI Gym [11] standards which has functions to choose actions, give rewards, get current states, etc. In our study, the drone is the agent and states are coordinate units of *x*,*y*,*z* in the whycon frame of reference tracked by the overhead camera. The reward system is written in the environment.

A. Altitude Position Hold

The focus of holding altitude required giving appropriate throttle commands to the drone using Reinforcement Learning. To this end, the observation space is the drones' altitude. The whycon *z* value gives us the drones distance from the camera that is interpreted as the drones' height from the ground. It is observed that the whycon *z* value varies between 20 to 40, with 20 being closer to the overhead camera i.e. higher drone altitude and 40 being closer to the ground i.e. lower drone altitude. Additionally, these values are discretized to its nearest integer value. The action space is PPM values given to the drone. These values range from 1200 to 1800. However, values 1200, 1800 and values closer to them are extreme values which would speed up the drone in the set direction. To prevent that, we set softer lower and upper bounds as 1300 and 1700 in order to allow the drone to correct itself without sudden speed up with correction values given in steps of 10. Hence, in total there are 20 observation states and 40 action states. During training phase, leaving the roll (*x*) and pitch (*y*) degree of freedom unchecked could affect the model getting trained. To prevent this, in our study we applied PID on the roll and pitch of the drone so as to hold its position at (*x*, *y*) = (0, 0) as per the whycon frame of reference.

B. Path Planning

For path planning, the observation space is *x*, *y* and *z* coordinates as the focus is on using Reinforcement Learning to compute a path avoiding obstacles. For this there are six actions - move forward, move back, move right, move left, go up and go down. These actions translate to movement of one unit in the corresponding axis in the whycon frame of reference. Here *x* and *y* varies between values -6 and 6 and *z* value varies between 25 and 35. Even here, the values of *x*, *y* and *z* are discretized. Hence, totally there are 6 actions and 1440 states i.e. 12 states of *x*, 12 states of *y* and 10 states of *z*.

IV. Q-Learning

The simplest method in Q-Learning is forming a Q-Table with discretized states and actions. The table maps states to actions through values known as Q values. These Q values tell how good the action is to a given particular state. In a particular state, that action is best which has the maximum Q value.

In Q-Learning, we maintain exploration and exploitation strategy using epsilon greedy policy [12]. Exploring is done by taking random action and finding how good that action is for that state and exploiting is done by taking that action which has the maximum Q value in that state i.e. the actor becomes greedy and takes that action which gives immediate maximum reward irrespective of long term consequences. Epsilon value is taken as 1 initially and decayed gradually as a function of episodes. The decaying can be linear, logarithmic or exponential based on the requirement. A random number is generated and if it turns to be less than epsilon value, then a random action is taken. If the generated number is more than the epsilon value, a greedy action is taken. Greedy refers to taking that action which has the maximum Q value for that state. In this project, for first 1000 episodes the epsilon is kept 1 so that the agent just explores. Later, it is exponentially decayed and around 2000 episodes it almost equally explores and exploits. After that the agent just starts exploiting.

For altitude hold, let z_{hold} be altitude to be held and *z* be the current state. The reward function is defined as given below:

```
if ($|z - z_{hold}| < 1$) :
    reward = 100
else if ($|z - z_{hold}| < 2$) :
    reward = 20
else if ($|z - z_{hold}| < 3$) :
    reward = 10
else if ($|z - z_{hold}| < 4$) :
    reward = 5
else if ($|z - z_{hold}| < 5$) :
    reward = -5
else if ($|z - z_{hold}| < 6$) :
    reward = -10
else:
    reward = -50
```

The reward system for path planning is the same, with difference being that instead of *z*, the radial distance is taken from the destination point as given in (1).

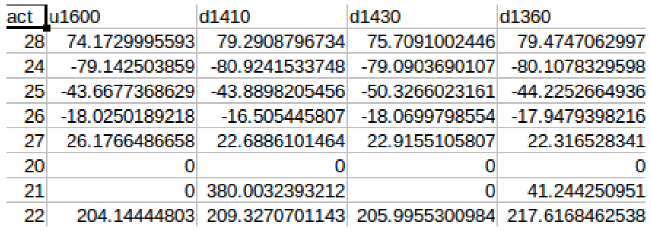
$$R_t = \sqrt{(x - x_{dest})^2 + (y - y_{dest})^2 + (z - z_{dest})^2} \qquad (1)$$

In the learning process, let the drone be in a state s_t. The agent takes an action a_t from the action set. After the action, the drone will go to a next state say s_{t+1} and it gets a reward R_t based on the reward system mentioned above. Now, the Q value corresponding to state s_t and action a_t is updated from bellman's equation given in (2).

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha[R_t + argmaxQ(s_{t+1}, a_t)] \qquad (2)$$

where α is the learning rate.

The above process runs in a loop for defined number of episodes. Once the learning is done in a simulated environment, the model is implemented on the drone. It then always takes only greedy actions from the table. A part of learnt Q table is shown in Figure 4.



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	217.6168462538

Figure 4. Part of Q-table.

In the table, as shown in Figure4, the rows denote the states, i.e. value of whycon *z* coordinate. The columns denote actions. For instance '*u1600*' denotes up-throttle with PWM value 1600 and '*d1410*' denotes down-throttle with PWM value 1410. Equal throttle is given to all motors as only altitude matters. The values in the table are Q-values for those corresponding states and actions. Based on the selected Q-value, a corresponding action is taken.

The Q-table for path planning looks similar with state denoted as *x_y_z* and 6 actions as mentioned before. If there is an obstacle in the path, it is considered as terminal state and the episode ends. A very high penalty (negative reward) is given for crashing into the obstacle. This way it tries to avoid those coordinate points while exploiting.

V. Results

As mentioned before, the model is trained on V-REP simulator first and is then tested on the pluto drone. Since the exact physics of the real world cannot be simulated in the simulation environment, the performance of the drone in the real world is not as perfect as in the simulator.

Graphs in Figure 5 and Figure 6 shows the comparison of Performances of continuous states PID vs discretized states RL respectively.

It has been observed that the deviation from the set point is slightly less in RL than PID though states were discretized for RL. Continuous state RL is not possible using Q-Table because there will be infinite states. Hence, approximators like neural networks have to be used to make it continuous state space for RL which clearly tells that RL will perform much better than PID.

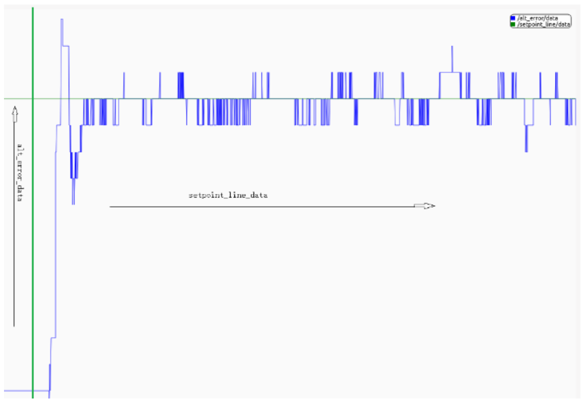


Figure 5. Steady state error with PID control.

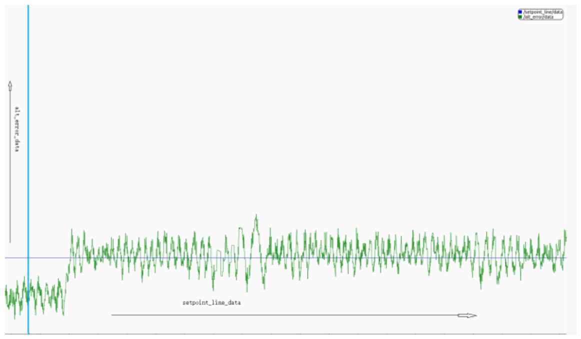
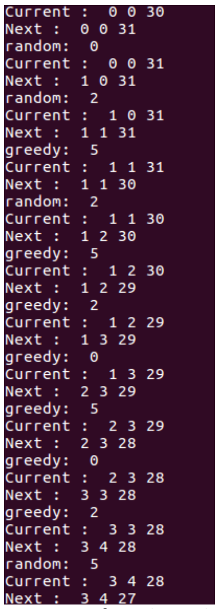


Figure 6. Steady state error with learnt RL model.



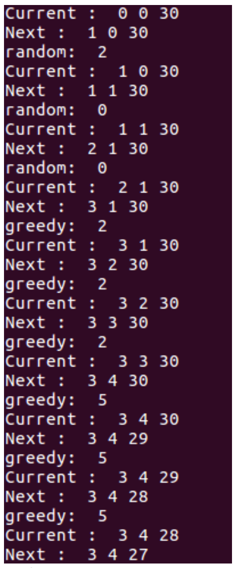
```
Current :  0 0 30
Next :  0 0 31
random:  0
Current :  0 0 31
Next :  1 0 31
random:  2
Current :  1 0 31
Next :  1 1 31
greedy:  5
Current :  1 1 31
Next :  1 1 30
random:  2
Current :  1 1 30
Next :  1 2 30
greedy:  5
Current :  1 2 30
Next :  1 2 29
greedy:  2
Current :  1 2 29
Next :  1 3 29
greedy:  0
Current :  1 3 29
Next :  2 3 29
greedy:  5
Current :  2 3 29
Next :  2 3 28
greedy:  0
Current :  2 3 28
Next :  3 3 28
greedy:  2
Current :  3 3 28
Next :  3 4 28
random:  5
Current :  3 4 28
Next :  3 4 27
```

Figure 7. Way-points from (0,0,32) to (3,4,27).

For path planning, the way points calculated using RL to reach destination point (3, 4, 27) with some obstacles at different coordinate points are given in Figure 7.

Now, adding additional obstacles at coordinates (1,3,28), (2,3,28), (3,3,28) and retraining the model makes the algorithm take different way-points as its path, thereby completely avoiding (2,3,28), (3,3,28) from previously given way points as shown in Figure 7. The new way points are shown in Figure 8.

The entire set up in the real world with the drone hovering using RL is shown in Figure 9.



```
Current :  0 0 30
Next :  1 0 30
random:  2
Current :  1 0 30
Next :  1 1 30
random:  0
Current :  1 1 30
Next :  2 1 30
random:  0
Current :  2 1 30
Next :  3 1 30
greedy:  2
Current :  3 1 30
Next :  3 2 30
greedy:  2
Current :  3 2 30
Next :  3 3 30
greedy:  2
Current :  3 3 30
Next :  3 4 30
greedy:  5
Current :  3 4 30
Next :  3 4 29
greedy:  5
Current :  3 4 29
Next :  3 4 28
greedy:  5
Current :  3 4 28
Next :  3 4 27
```

Figure 8. Way points from (0,0,32) to (3,4,27) avoiding additional obstacles given.

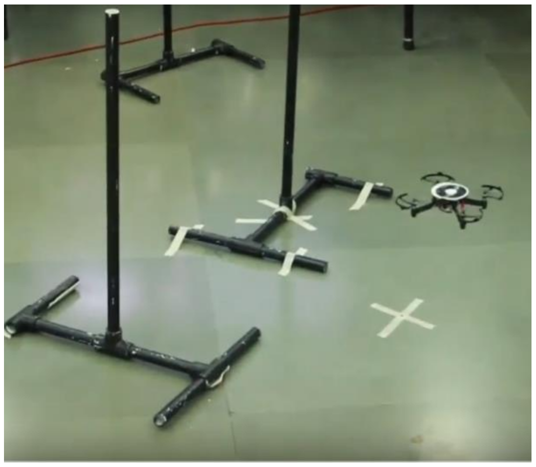


Figure 9. Real world set up and drone hovering using RL.

VI. Conclusion

This paper presented a technique to train a drone to learn to hold a given altitude using Q-Learning and also to plan a path avoiding obstacles using Q-Learning and PID. Hence, these techniques can be used in the environments that are actually not known well. Models learnt from reinforcement learning gives a very robust performance in these unknown environments. Even if the environment changes a little, Q-Table can constantly keep updating as per environment which makes it more robust. From the graphs in results, it can be clearly seen that sampling rate for RL is much faster than PID because PID involves calculations of the parameters (K_p, K_d and K_i) but in Q-Table it is just searching. Therefore, RL response is faster than PID. Also, by seeing the rising time and overshoot from graphs, the transient response of RL is much better than that of PID. Discretized RL performed slightly better than PID, hence we can say making a continuous state space for RL will even give a much better performance than PID.

VII. Future Work

As mentioned before, the Q-Table can be approximated by a neural network so that the state space is continuous. These can be implemented as deep Q-networks. Even in deep Q-networks, the policy is still epsilon greedy.
This may not be a very good strategy for exploring and exploiting. Hence, we can use actor critic models in which the policy is also updated and improved while learning. Also, in this project PID has been applied to roll and pitch axis in order to avoid disturbances. The final target is to completely eliminate PID and produce a complete stable and robust model just using Reinforcement learning.

Acknowledgment

We acknowledge the support of e-Yantra project staff especially Omkar Manjrekar and Rishikesh Madan. We would also like to thank Drona Aviation for their support with the Plutodrone.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wier-stra, and M. Riedmiller, “Playing atari with deep reinforcement learn-ing,” arXiv preprint arXiv:1312.5602, 2013

[2] P. Kormushev, S. Calinon, and D. Caldwell, “Reinforcement learning in robotics: Applications and real-world challenges,”Robotics, vol. 2, no. 3, pp. 122–148, 2013.

[3] “DronaAviation.” [Online]. Available ttps://www.dronaaviation.com/

[4] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 1st ed. Cambridge, Massachusetts and London, England: The MIT Press, 2017.

[5] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, “Q-learning algorithms: A comprehensive classification and applications, ”IEEE Access, vol. 7, pp. 133 653–133 667, 2019.

[6] J. G. Ziegler and N. B. Nichols, “Optimum settings for automatic controllers,” trans. ASME, vol. 64, no. 11, 1942.

[7] H. Pham, H. La, D. Feil-Seifer, and L. Nguyen, “Autonomous uav navigation using reinforcement learning,” 01 2018.

[8] Nitsche, T. Krajnik, P. Cizek, M. Mejail, T. Duckettet al., “Whycon: an efficient, marker-based localization system,” 2015.

[9] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, “Virtual robot experimentation platform v-rep: A versatile 3d robot simulator,” in Simulation, Modeling, and Programming for Autonomous Robots, N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, and O. von Stryk, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 51–62.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T.Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in ICRA workshop on open source software, vol. 3, no. 3.2. Kobe, Japan,2009, p. 5.

[11] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, “Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo,” arXiv preprint arXiv:1608.05742, 2016.

[12] C. Sun, “Fundamental q-learning algorithm in finding optimal policy,” in 2017 International Conference on Smart Grid and Electrical Automation(ICSGEA). IEEE, 2017, pp. 243–246