İstanbul
Bilgi Üniversitesi

RL Approach for Autonomous Control and Landing of a Quadrotor

by

Hamza Abuzahra,
Ahmed Sameh Ahmed,
Maximilian Cevdet Arat,

Supervised by

Elena Battini Sonmez

Submitted to the
Faculty of Engineering and Natural Sciences
in partial fulfillment of the requirements for the

Bachelor of Science

in the
Department of Computer Engineering

June, 2023

Abstract

This study looks deeper into how a drone functions with the aim of optimizing autonomous flying and landing with the aid of reinforcement learning and image processing. The demand for quadrotors has increased recently in various fields, which brings to light the issues expected to be faced. The two main problems faced are the need for resources to train professional pilots for the quadrotors and the various hindrances that could be faced affecting the communication between the control station and the quadrotor. The research project will be split into mainly two phases; the training for navigation purposes and training for landing using markers. Both phases will be conducted using Gazebo and ROS. The results of the research proved feasible using DDPG algorithm for both phases with room for improvement in future works.

Table of Contents

List of Figures

List of Abbreviations

RL	Reinforcement Learning
ROS	Robot Operating System
UAV	Unmanned Arial Vehicle
AI	Artificial Intelligence
DQN	Deep Q-Network
DDPG	Deep Deterministic Policy Gradient

1 Introduction

In this day and age quadrotors have become quite a popular technological advancement used in a wide variety of fields and for multiple purposes. The quadrotors control mechanism is mainly split into three types: Manual, Automated, and Autonomous. Manual means the quadrotor is controlled using a workstation via satellite connectivity, radio waves or the internet. Automated refers to providing a defined set of parameters and the device will follow a set of preset rules. Autonomous on the other hand means that the device constantly monitors its dynamic environment and takes in that information to adapt [1].

The process of autonomous flying and landing of a quadrotor is proven to be a very difficult task which can be tackled using numerous methods. In this paper, we will be using Reinforcement Learning (RL) to train the quadrotor to follow a path using image processing. An open-source framework that can be used is Robot Operating System (ROS) which is a set of libraries and tools that aid us in simulating robot applications. In addition, gazebo is a simulator generally used with ROS to create a suitable simulation environment to test the functionality of a drone. The end goal is to optimize autonomy for quadrotors to eliminate the need for resources in training professional pilots and to reduce any effect of the environment on communication with the quadrotor.

This paper is further split into the following sections. Section 2, the Related Works, touches on the literature review along with the scope of the study. Section 3, the Methodology, contains the necessary information on the algorithms and the training and testing phases. Section 4, the Design and Experimental Setup, contains details of the algorithm, applications used for simulation and training and the finalized hardware for the testing phase. Section 5, the Experiments and Results, contains the experiments conducted in the study along with their results and, lastly, Section 6, the Conclusion, concludes the current state of the study.

2 Related works

For the related works, several different research has already been conducted that aid the current study, some of which worked on the supporting functions of the earlier studies, which will be addressed in this section. Sensor-fusion techniques require costly sensors to full fill the task without any major problems but it's not suitable for the consumer drone market. Additionally, the majority of these techniques rely on GPS signals, which are frequently absent in congested real-world situations. Due to the utilization of external sensors, device-assisted approaches provide an accurate posture assessment of the UAV throughout the entire landing maneuver [2].

The conference paper [3] explains that the AI Drone does not require any type of sensors or other hardware beyond the embedded cameras. When RL (Reinforcement Learning) algorithm is used to drive the vehicle to the designated marker, They have seen that using the identification technique by using the markers is significantly more reliable even though the embedded camera's low image quality.

One way or another the UAV must avoid crashing into obstacles and autonomous UAV have to make judgments by reacting appropriately to unanticipated situations in real time without direct human interference. To estimate and implement the methodology, the capability of the solutions is used [4]. This study describes a self-trained controller for autonomous navigation in a static and dynamic environment that is trained by using RGB and RGB-Depth images. The Gazebo simulator algorithm has been trained in dynamic and static situations, and the findings demonstrate that the system is able to comprehend an optimized solution for high-level control of a UAV. The training is performed with a UAV, which is employing a variety of rewards, including obstacle avoidance, motivation, reaching goals, and area rewards, in a multi-objective approach [5].

Another study demonstrates how checkpoints improve research findings on drone delivery. These checkpoints can be used to improve the test results and by allowing the model to be restored to a previously saved state, rather than starting the training process from scratch. This can be particularly useful in deep reinforcement learning, where the training process can be time-consuming and involve many iterations. Additionally, using checkpoints can help to prevent overfitting by restoring the model to a point where it performed well on the test data rather than continuing to rain [6].

As for the landing task several papers took a low cost approach to accurately identify and land on a specific spot, a downward camera was used to extract information about the exact position of the landing spot by image processing techniques.
Moreover, [3] approached the landing task by allowing the drone to explore the xy-plane and trigger descending when it thinks it is in the right spot. In [7] and [8], descending was done automatically in a steady rate while allowing the agent to change its position in the xy-plane.
[2] utilized ArUco markers to allow the drone to recognize the landing spot more accurately and in a robust way.

Overall, our proposed approach represents a significant advance in the field of autonomous UAV control and has the potential to enable more effective and efficient use of these vehicles in a variety of applications. Further research and development are needed to fully realize the potential of this approach, but our initial results are promising and demonstrate the feasibility of our approach.

3 Methodology

3.1 Workflow of the Project

Figure 1 shows the workflow of the project. Actor network representing the policy of the agent decides on the actions to be taken, with a possibility to explore a new action. The action is then executed by the drone then receives a reward that is used as a label to update the weights of the critic network. The critic network takes state of the agent and the action it would take as input, to evaluate the value of the action that is then used to update the weights of the actor network.

3.2 Reinforcement Learning

3.2.1 What is Reinforcement Learning

Reinforcement learning is an active research field, which branches from machine learning, that is concerned with enabling an agent, generally a robot, to learn from experience by trial and error through interacting with its environment. The environment gives back feedback signals to the agent according to the actions taken. In case the sequence yielded success it takes a positive reward, otherwise, it gets a negative or no reward. In this way, setting a particular reward function allows us to train the agent to do the desired task.

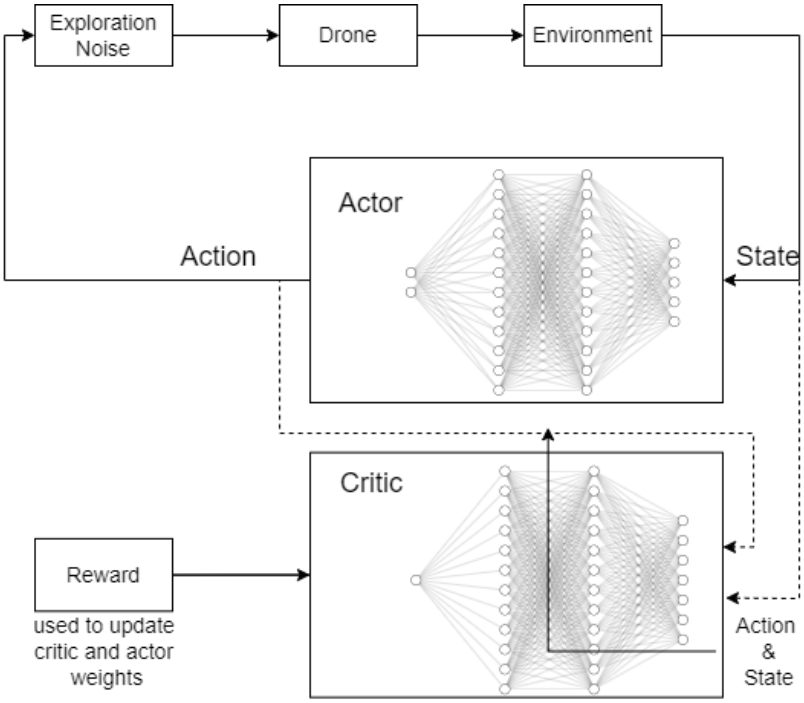

Figure 1: Flow Chart Diagram

Figure 2 shows the feedback loop between the agent and the environment.

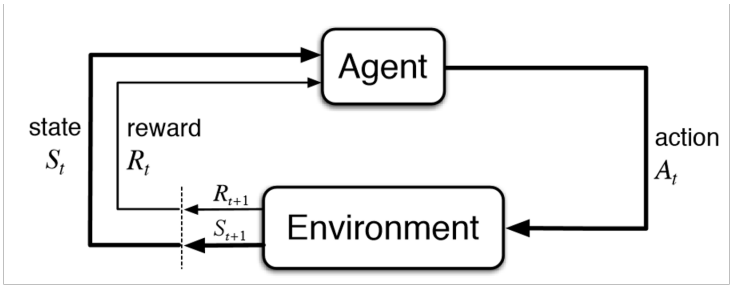

Figure 2: Reinforcement Learning Model, adopted from [9]

3.2.2 Formulation of Reinforcement Learning Problems

Some useful definitions used in RL:
Environment: the space where the agent operates, could be anything from our physical world to any virtual environment such as an Atari game.
State: the state of the agent describes the situation of the agent in the context of the environment.

Policy: in other words the mind of the agent, is a method that determines the next action to be taken in a certain state.
Reward: A feedback signal from the environment indicating how good the action taken was.
Value: the future reward an agent would get if a certain action is taken.

RL problems are mainly described using a Markov Decision Process. Which is a mathematical framework that consists of a set of finite environment state S, a set of actions in each state A, a reward function R and a transition model P. Transition models are used in case the dynamics of the environment are known. In the case of a non-deterministic environment such as our real world, a model-free approach is used instead.

RL algorithm mainly focuses on optimizing the policy of the model, the value function, or a combination of the two. During the training phase, the agent faces a choice between exploring new actions and exploiting the current best-known actions, this is known as **exploitation vs exploration** trade-off. Many methods are proposed to tackle this issue, which is mainly to accept short-term sacrifices to be able to obtain better long-term results.

3.2.3 Practical Applications of RL

RL requires a lot of data, hence domains where simulated data is widely used, such as computer gameplay and robotics, make the best use of this method. AlphaGo Zero [10] is an example of RL in games, it is trained to play games such as the ancient Chinese game of GO, Backgammon, chess and some Atari Games. In robotics, RL is used for automation as a main focus. It enables the robot to learn from its own experience how to optimize control systems. [9]

4 Design and Experimental Setup

4.1 Simulation

4.1.1 Gazebo

Gazebo is an open-source 3D simulator which provides multiple features like physics engine, rendering, sensor simulations and actuator control for robotic application and simulation. It supports multiple physics engines including ODE, which is the default, and Simbody. It can mimic the real-life environment with its rendering capabilities including realistic lighting, shadow

and textures. Moreover, it can simulate accurate sensors with vision capabilities similar to laser range finders, cameras, and kinetic sensors. Gazebo utilizes OGRE engine for 3D rendering [11]. Figure 3 shows a scene from the simulator.

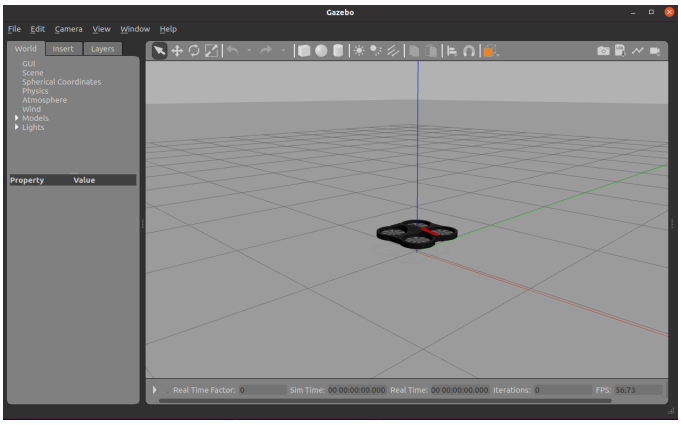

Figure 3: Gazebo Simulator

4.1.2 ROS

Robot Operating System (ROS) is an open-source set of libraries and tools that allow you to create robot applications. ROS Noetic Ninjemys is a version of ROS targeted mainly towards Ubuntu 20.04 hence the team decided to run Ubuntu virtually on Windows 11 using Vmware on one machine and on 2 other devices on dual boot.[12]

4.1.3 Environment

The drone will be tested in 2 types of environments, empty and outdoor prototypes. An example of the empty environment that will be used is shown in Figure 4 while figure 5 shows an example of an outdoor environment.

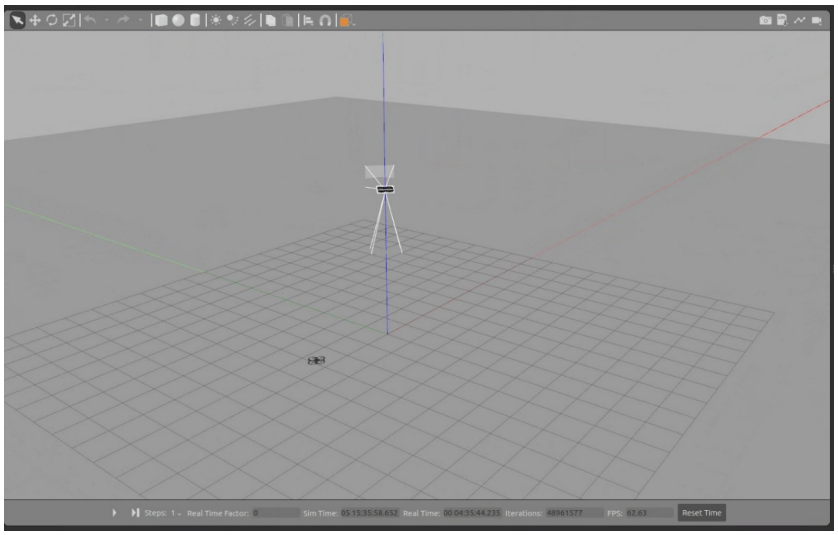

Figure 4: Empty World in Gazebo Simulator

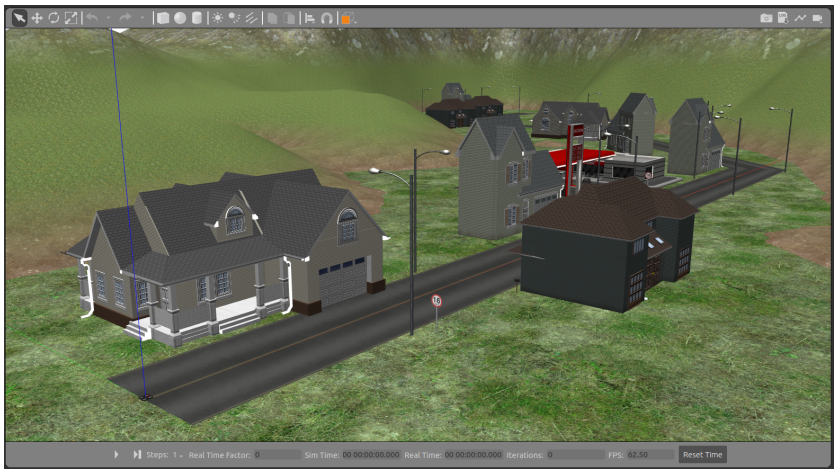

Figure 5: Outdoor world in Gazebo Simulator

4.1.4 The Quadrotor

Experiments will be done using AR Parrot 2.0 drone. The drone has a front camera and a downward camera built-in, light weight and affordable. In addition, it has ready to use models and packages in Gazebo and ROS, which made it more popular in the community and widely used in experiments especially those involving RL. A photo of the drone is shown in figure 6.

Figure 6: AR Parrot Drone 2.0

4.2 Training

The language of choice is python since the team is proficient in this language and it seems the most efficient and widely used language for reinforcement learning.

4.2.1 Actor Critic

Model-free based RL algorithms are policy-based or value-based. Policy-based algorithms directly optimize the policy of the agent explicitly, whereas value-based algorithms try to find the optimal value function. The policy is implicitly found by choosing the action with the highest value. Another approach is to combine the two into a method called Actor-Critic. The Actor part of the agent chooses an action to be taken, and the Critic on the other hand evaluates how good the action is. The actor learning is done by optimizing the policy using policy gradient, while the critic learning is done by finding the optimal value function. [13]

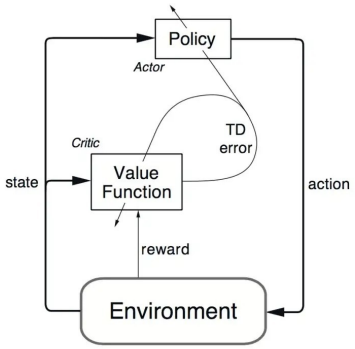

Figure 7: Actor Critic Algorithm, adopted from [13]

4.2.2 Deep Q-Network

Q-learning is a method that finds the optimal value function in a finite state-action space. Starting with a Q-table, that maps states with actions, initialized with 0s. And the table is updated for each action taken and the received reward, according to the bellman equation shown below. [14]

$$Q(S_t, A_t) = (1 - \alpha)Q(S_t, A_t) + \alpha(R + \gamma * max_a Q(S_{t+1}, a)$$

This method is not possible to use in our case as it requires to save an entry in the table for every state-action pair so that an infinite length table would be needed. Q-learning ensures finding the optimal value function. However, it only works on finite state-action space. Approximation methods are used to find a near-optimal value function. [15] introduced Deep Q-Networks which uses a Neural Network to map from state inputs to action-value pairs. A CNN can be used to be able to get state information from images. In this way, the table lookup is avoided and Q-learning updates the weights of the neural network instead of table entries. Figure 8 demonstrates the mapping from states to Q-values of actions.

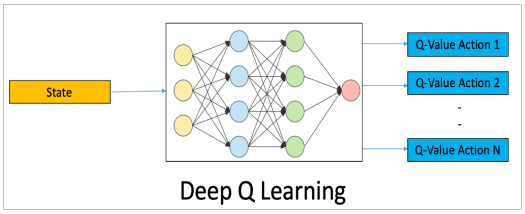

Figure 8: Deep Q Algorithm, adopted from [16]

This research project takes the actor-critic approach with Deep Q-Networks

as critics. Along with a deterministic policy network as an actor. And that makes up the Deep Deterministic Policy Gradient (DDPG) algorithm [17]. Figure 9 shows the differences between DQN and DDPG.

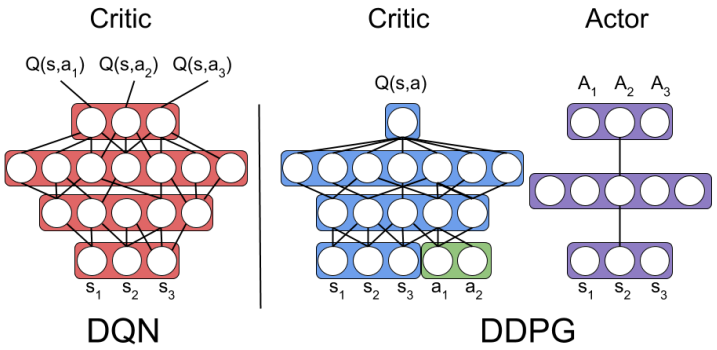

Figure 9: Deep Deterministic Policy Gradient algorithm, adopted from [18]

DDPG has two additional networks called target networks. Updating the weights of the actor and the critic is done according their corresponding target network. A target network updates are done less frequently than the updates of the main networks. This is implemented to add stability to the networks, which allows the agent to converge to a policy. Without target networks, and because the actor and the critic are learning at the same time, the actor would almost always fail to converge on a policy that describes the values given from the critic network, because the objective is constantly changing.
Pseudocode of the DDPG algorithm is shown in Algorithm 1.

Algorithm 1 Deep Deterministic Policy Gradient (DDPG)

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor network $\mu(s|\theta^\mu)$ with weights θ^Q with weights θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{M}$
Initialize exploration rate ϵ and decay parameters $\epsilon_{\min}, \epsilon_{\text{decay}}$
for episode = 1 to M **do**
  Receive initial observation state s_1
  for t = 1 to T **do**
    Select action $a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \mu(s_t|\theta^\mu) + \mathcal{N}_t & \text{otherwise} \end{cases}$
    Execute action a_t and observe reward r_t and new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample a random mini-batch of N transitions
    (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
    Compute target values $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the mean squared Bellman error:
      $\mathcal{L} = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
      $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
    Update the target networks:
      $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
      $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    Decay exploration rate: $\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \times \epsilon_{\text{decay}})$
  end for
end for

5 Experiments and Results

5.1 Preliminary Work

The team found the AR Drone model for gazebo and are successfully capable of manually controlling the quadrotor using the keyboard shown in figure 10 and a sample of the hovering quadrotor is seen in figure 11. The training worlds have been chosen.

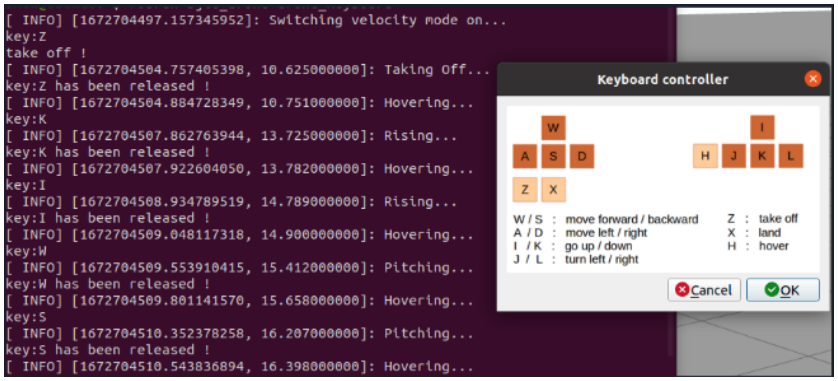

Figure 10: Quadrotor Control Keyboard

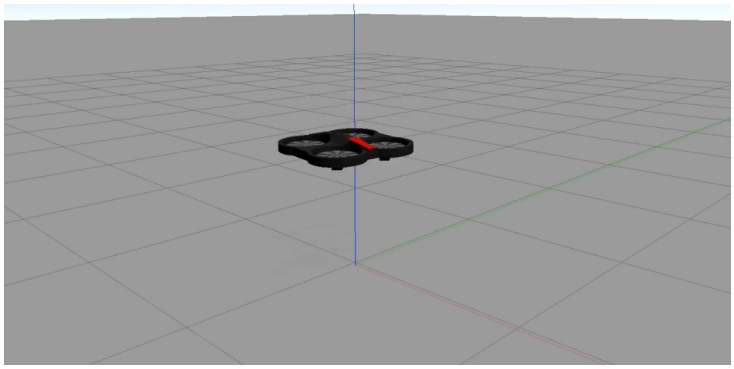

Figure 11: Sample Hovering Quadrotor

5.2 Navigation Experiments

5.2.1 The Setup

The movement of the drone is being automated using a move function which controls the Twist message that controls the drones coordinates as shown in figure 12.

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Figure 12: Twist message documentation from ROS [19]

The input state is

$$S = (d_x, d_y, a_z, ta_z, crashrange)$$

with dx being the difference between the quadrotor and the target on the x plane, dy being the difference in the y plane, az being the quadrotor orientation around z axis according to the worlds origin, taz being the target angle relative to the world and crash range being how close the quadrotor's sonar is to an object.

The agent has a 2 dimensional action function which is represented by

$$A = (lx, az)$$

lx determines the agent moving forward and backwards in the x plane while az determines the rotation of the agent to change its orientation.

The reward function used is represented by

$$\sum_{i=1}^{4} c_x r_x = c_1 r_1 + c_2 r_2 + c_3 r_3 + c_4 r_4$$

where $c_1 = 10$, $c_2 = 10$, $c_3 = 15$, $c_4 = 1$, multiple combinations of constants were tested and the mentioned combination is the most fitting. Meanwhile, for the rewards r_1 is for if it crashes where the sonar range is between 0 and 5 and the reward's highest value is 0, r_2 being the distance reward which

is positive the closer it gets to its target and negative the further away it gets, r_3 being the angle reward which gives reward based on difference of the orientation of the agent and the target, and r_5 being the time reward which gives negative reward for every timestep. The equations for the rewards are shown below:

$$r_1 = -1 + \frac{min(crashRange, 5)}{5}$$

$$r_2 = previousDistance - currentDistance$$

$$r_3 = agentAngle - targetAngle$$

$$r_4 = -0.01$$

The agent starts at a randomised starting position and a randomised target for each episode for the purpose of generalisation so the agent can then succesfully navigate to any given target from any given start position. The episode terminates if any of the following 4 conditions are met: the agent crashes, the agent reaches its target, the agent goes away further than 15 meters from its target, and if it reaches the max number of steps per episode which is set to 350 steps.

5.2.2 Phase 1 - Empty Environment

The agent was first trained in the empty environment in figure 4 for E number of episodes with varying hyper-parameters in order to find the perfect values for α, β, τ, and batch_size(bs).The table below displays a few of the hyper-parameters tested which served as checkpoints in figuring out the perfect combination.

Test	α	β	τ	bs
1	0.003	0.003	0.001	64
2	0.0001	0.0003	0.001	1024
3	0.0001	0.0003	0.1	1024
4	0.00001	0.00003	0.001	512
5	**0.00001**	**0.00001**	**0.001**	**512**

Results: after testing with multiple hyper-parameters, the hyper-parameters of *Test5* proved to be the most successful with an average reward per 100 episodes converging at 500 and the training was completed in 723608 steps in 4500 episodes. The evaluate graph shown in figure 13 displays the average reward gained every episode.

Figure 13: Reward graph for navigation in empty environment

5.2.3 Phase 2 - Outdoor Environment

Transfer learning was used hence the trained agent from the previous phase is then trained in the outdoor environment in figure 5 so the agent learns to dodge obstacles. The same combination of hyper-parameters were used from the previous phase. Figure 14 shows the average reward per 100 time-steps during the training, which shows convergence to an optimal policy.

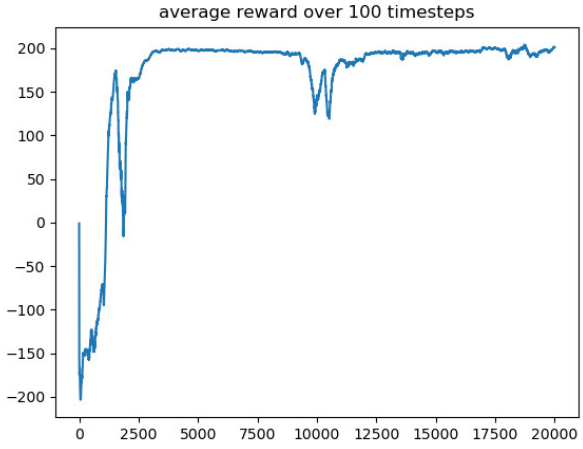

Figure 14: Reward graph for navigation in outdoor environment

5.3 Landing Experiments

5.3.1 The Setup

For the landing the ArUco marker approach is being used in which when the model picks up on the marker with the downward camera and starts decending. The ArUco marker used is in the figure 15. The model extracts the features from the image which is used to determine the location of the marker relative to the image captured by the camera.

Figure 15: ArUco marker used for landing

The input state is

$$S = (d_x, d_y, inframe)$$

with dx being the difference between the center of the image and the marker on the x plane, dy being the difference in the y plane, and inframe is a bool which changes based on whether the marker is in the frame of the quadrotor's camera or not.

The agent has a 2 dimensional action function which is represented by

$$A = (lx, ly)$$

lx determines the agent moving forward and backwards in the x plane while ly determines the movement in the y plane.

The reward function used is represented by

$$R = 1 - ((\frac{d_x}{0.6667})^2 + (\frac{d_y}{0.5000})^2)$$

The agent starts every episode at a randomised starting position above and

around the landing position. The episode terminates if any of the following 2 conditions are met: the agent lands or the marker is no longer in the frame of the camera.

5.3.2 The Experiments

The agent was trained as shown in figure 16 for E number of episodes with varying hyper-parameters in order to find the perfect values for α, β, τ, and batch_size(bs).The table below displays a few of the hyper-parameters tested which served as checkpoints in figuring out the perfect combination.

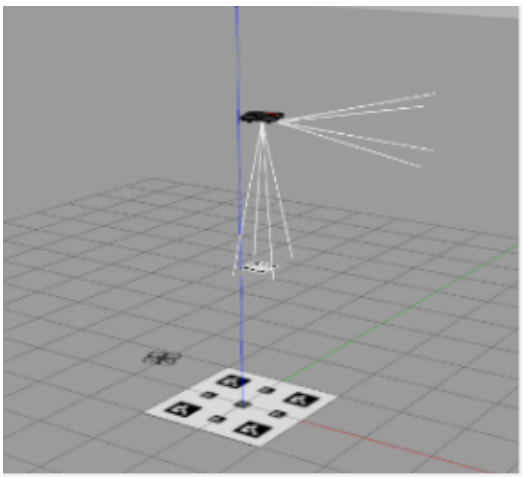

Figure 16: Setup for the landing training

Test	α	β	τ	bs
1	0.003	0.0003	0.01	1024
2	0.001	0.001	0.01	256
3	0.00001	0.00001	0.001	512
4	0.0001	0.0001	0.01	256
5	**0.0001**	**0.0001**	**0.001**	**512**

Results: after testing with multiple hyper-parameters, the hyper-parameters of *Test5* proved to be the most successful with a successful landing rate of 60% and an average reward per 100 episodes converging at 15 and the training was completed in 264940 steps in 10000 episodes. The evaluate graph shown in figure 17 displays the average reward over 100 episodes.

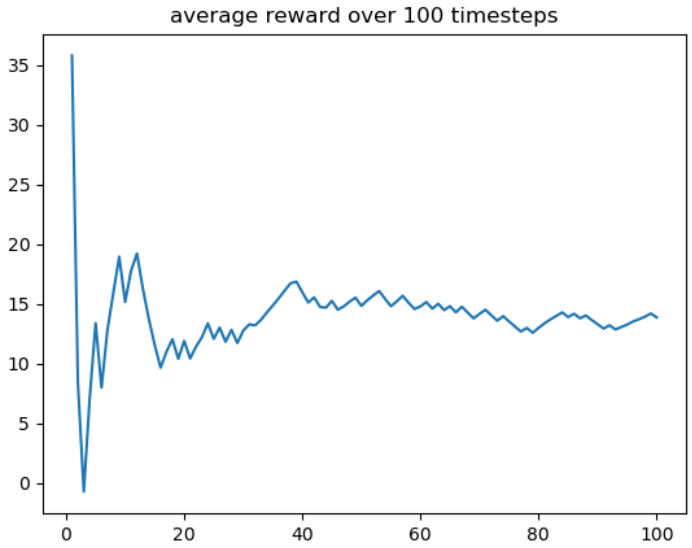

Figure 17: Evaluate graph for landing over 100 eps

6 Current Conclusion & Future Work

6.1 Conclusion

In conclusion, this research vehemently pushes this field further since no previous research used the same architecture and algorithm for both the navigation and landing with just a change in the reward system for the cases. DDPG was successfully implemented for the determined purpose of this research, which is to optimize the autonomy of navigation and landing of a quadrotor. Suitable results were achieved with a very positive reward for generalized navigation with obstacle-avoiding capabilities and successful landing on ArUco markers.

6.2 Future works

For the future of this research there are three main paths to be taken and tested. The first one being the plan to implement the model into an actual quadrotor in order to test if the simulation is functional for a real life test. The second path would be to test other RL algorithms and try to compare the results to find out which algorithm would be a perfect fit for our purpose. Lastly, more research can be done to generalize the algorithm for all quadrotors not just AR drone.

References

[1] S. Matteson, "Autonomous versus automated: What each means and why it matters," *TechRepublic*, 2019.

[2] J. Jiang, Y. Qi, M. Ibrahim, J. Wang, C. Wang, and J. Shan, "Quadrotors' low-cost vision-based autonomous landing architecture on a moving platform," in *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2018.

[3] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for uav autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018.

[4] A. Ramezani Dooraki and D.-J. Lee, "A multi-objective reinforcement learning based controller for autonomous navigation in challenging environments," *Machines*, vol. 10, no. 7, 2022.

[5] A. T. Azar, A. Koubaa, N. Ali Mohamed, H. A. Ibrahim, Z. F. Ibrahim, M. Kazim, A. Ammar, B. Benjdira, A. M. Khamis, I. A. Hameed, and G. Casalino, "Drone deep reinforcement learning: A review," *Electronics*, vol. 10, no. 9, 2021.

[6] G. Muñoz, C. Barrado, E. Çetin, and E. Salami, "Deep reinforcement learning for drone delivery," *Drones*, vol. 3, no. 3, 2019.

[7] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, and H. Bang, "Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 108–114, 2018.

[8] M. Piponidis, P. Aristodemou, and T. Theocharides, "Towards a fully autonomous uav controller for moving platform detection and landing," in *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems (VLSID)*, pp. 180–185, 2022.

[9] Bhatt and Shweta, "Reinforcement learning 101," Apr 2019.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and et al., "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354–359, 2017.

[11] "Ros gazebo: Everything you need to know," 2021.

[12] "https://www.ros.org/."

[13] D. Karunakaran, "The actor-critic reinforcement learning algorithm," Sep 2020.

[14] M. Wang, "Deep q-learning tutorial: Mindqn," Oct 2021.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529–533, 2015.

[16] H. Nguyen, "Playing mountain car with deep q-learning," Mar 2021.

[17] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning* (E. P. Xing and T. Jebara, eds.), vol. 32 of *Proceedings of Machine Learning Research*, (Bejing, China), pp. 387–395, PMLR, 22–24 Jun 2014.

[18] D. Karunakaran, "Deep deterministic policy gradient(ddpg) - an off-policy reinforcement learning algorithm," Nov 2020.

[19] "http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html."