Learning Control Policies for quadcopter Navigation with Battery Constraints

Rohit Murthy, Harikrishnan Suresh and Chris Song
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{rohitmur, hsuresh, qsong}@andrew.cmu.edu

Abstract

quadcopter UAVs are being heavily deployed in autonomy tasks due to its small size and high maneuverability, thereby enabling them to execute complex trajectories efficiently. However in the context of long range autonomy, these robots are limited in terms of performance due to their battery constraints. This brings forth the need to generate efficient planners that can enable the quadcopter to operate in a real environment for extended periods with maximum productivity. Through this project, we aim to develop an agent that has learned to navigate to a goal using an energy efficient control policy. The agent is trained in a reinforcement learning setting using the Deep Deterministic Policy Gradient algorithm. We evaluate our algorithm in a simulated environment and show that it performs better than classical control strategies.

1 Introduction

Quadcopters have become increasingly popular in recent times for research and consumer applications. They unlock the ability to traverse large expanses of space with few obstacles impeding its path. One major drawback of these systems is the small battery system that they can carry. Hence it is important to develop strategies that allow quadcopters to maximize the area that they can cover given the battery constraint.

In this project, we attempt to enable a quadcopter agent to learn to navigate to a goal location using energy efficient control policies. This will result in lower battery consumption which in turn will improve the long range navigation capabilities of quadcopters. This has another effect of allowing quadcopters to increase the payload that they carry for the same distance traveled. This is invaluable in the real-world to autonomous drone delivery systems such as Amazon Prime Air.

2 Related Works

Deep Reinforcement Learning has been used to solve complicated, non-linear problems beginning with video games in [1]. Since then there has been important work done to extend these approaches to continuous action and state spaces which are very relevant to the field of robotics. Model-based Reinforcement Learning methods have been successful in control of very unstable control systems such as quadcopters as shown in [2] and [3]. David Silver et al. [4] extended policy gradient methods to continuous action spaces which allow the use of many of the tricks used in the Deep Q-Networks implementation in [1]. [5] successfully shows that Reinforcement Learning can be used for the low-level control of quadcopters to solve complex maneuvers such as stabilizing after throwing the quadcopter from a hand upside-down.

There have also been attempts to improve the energy-efficiency of quadcopters to counter the battery constraints. The focus however has been to plan efficient paths [6] or the model of the quadcopter [7] rather than developing low-level control policies to improve the energy-efficiency. Drawing inspiration from both classical energy-efficient planning strategies and learning-based low-level control, we train an end-to-end learning agent to navigate to a goal while considering battery constraints.

3 Method

3.1 Problem Formulation

The problem we are trying to solve requires a quadcopter with a battery constraint to navigate to a goal state by learning an appropriate low level control policy. For the low level control problem, we are only considering position control for simplicity purpose and hence our input state includes the current state of the system. To ensure that the quadcopter is able to generalize a policy based on goal state and battery level we include these in the *state* as well. Based on the current position, the current battery level and the target position, the algorithm computes the required instantaneous velocities for the quadcopter which is the *action*. Hence our problem has a continuous state space as well as a continuous action space. The state space used by the agent is given in Eqn. 1 and the action space is given in Eqn. 2 .

$$S = [x \;\; y \;\; z \;\; \textit{goal\_x} \;\; \textit{goal\_y} \;\; \textit{goal\_z} \;\; \textit{battery\_level}]^T \tag{1}$$

$$A = [V_x \;\; V_y \;\; V_z]^T \tag{2}$$

Each episode is terminated if the quadcopter becomes unstable or runs out of battery, eventually crashing into the ground.

3.2 Battery Drain Formulation

The key contribution of this project is to incorporate the battery constraints of a quadcopter. We wanted the agent to conserve its battery usage while navigating to its goal position. The battery drain is modeled according to Eqn. 3

$$\Delta battery = - \left\| V \right\|^{1.1}, V = [1+ \mid v_x \mid, 1+ \mid v_y \mid, 1+ \mid v_z \mid] \tag{3}$$

Since the velocity of the quadcopter is usually below 1m/s, we added a *+1* term to ensure that the velocity monotonically increases with an increase in velocity. Also, when quadcopters don't have velocity they are still hovering which should drain the battery. The velocity in x, y and z are treated as equal and penalized in the same fashion. The exponent was crucial because the formulation became nonlinear. An increase in speed will cause the battery to drain faster. The agent must find a middle ground between getting to its goal location as quickly as possible so that its goal reward will not be heavily discounted and controlling its velocity so that it doesn't run out of power in the middle of its journey.

3.3 Reward Shaping

A major challenge in Reinforcement Learning, especially for Robotics applications, is coming up with a reward function that enables the agent to learn the desired behavior. We experimented with several reward functions to come up with the most promising reward function for the agent to learn a stable policy. Our most naive approach was purely based on the distance and heading of the agent to the goal. This reward function has several shortfalls one of which was that it allowed the agent to learn a less than optimal behavior of circling around the goal to accumulate as much rewards as possible without ever terminating. The main inspiration for our final reward function shown in Eqn. 4 was drawn from Andrew Ng, Harada and Russell (1999) [12]. They suggest that the reward function should take the form of a potential function.

$$R = \begin{cases} \|X_g - X_{t-1}\| - \|X_g - X_t\|, & \text{for incremental distance from goal} \\ -\frac{\|V\|^{1.1}}{100}, & \text{for battery drain} \\ +50, & \text{for reaching the goal within a threshold of 0.5m} \\ -5, & \text{for crashing or dying out of battery} \end{cases} \tag{4}$$

With this reward function, we only give out a reward if the agent is making progress towards the goal with the relative measure of it's distance to the goal rather than absolute. Furthermore, we also decrease the reward based on how much battery was used in the process. The battery drain was scaled down by 100 to ensure that all rewards were of similar magnitudes. The final goal reward was given to be 50. It was made to be significantly bigger because we want to ensure that the agent is able to understand that its importance is higher that the positive reward at each step. If the agent was successful in reaching the goal, a large enough reward must be used to reflect that and reinforce the learning process. A negative reward of -5 is given if the agent becomes unstable and crashes into the environment. We chose a moderate penalty for crashing because a big penalty makes it difficult to learn any positive behavior at the start of training.

3.4 Implementation

For our RL problem setting, it was important to deal with continuous action space as the agent must learn a policy along the lines of a position controller for navigation. Our main algorithm is the Deep Deterministic Policy Gradient (DDPG) [4] as detailed in Alg. [1]. DDPG is an actor-critic algorithm that is also model free, off-policy and utilizes some of the features presented in Deep Q-Networks (DQN)[1]. Similar to a DQN, the DDPG employs experience replay as well as target networks during training for better stability. With experience replay, we are able to break apart temporal correlation during training and remove any compounded variances from bad predictions. Target networks are a great tool for normalizing the network while training and providing stability. The weights of the target network are made to slowly track the learned networks with the soft target parameter.

To ensure sufficient exploration during training, noise is added to the actor policy during training based on the Ornstein-Uhlenbeck process that generates temporally correlated exploration for exploration efficiency in physical control problems with inertia. The noise is sampled from the distribution with $\sigma = 0, \rho = 0.3$ and *decay rate* = 0.6. We also added an additional exploration term that decreases linearly from 1 at the rate 1e-5. During testing, no noise is added.

The network hyperparameters are listed in the table below. During training, the learned policy is frozen every 50 episodes and then evaluated for 10 episodes to find the best model. The activation function used for the output layer of the actor is *tanh* as the agent requires both positive and negative velocities. The network architecture used in this project for the main task is shown in Fig. 1. The network architecture and hyperparameters were similar for most of the tasks, any changes will be explained in the corresponding results section.

Table 1. List of hyperparameters

Batch Size	32
Replay Buffer Size	5000
Actor Learning Rate	$1e^{-4}$
Critic Learning Rate	$1e^{-3}$
Target network tracking parameter, τ	0.125
Discount Factor, γ	0.98
# episodes	2500

3.5 Simulation Environment

The quadcopter is simulated using the Gazebo simulation engine, with the *hector_gazebo*[9] ROS package modified to our needs. To use this simulator for reinforcement learning we developed a custom *OpenAI Gym*-like environment as a wrapper to the simulation to perform all the required functions like *step, reset, sample, etc.* The quadcopter state observations required for the algorithm are obtained from the simulator, which produces the required control actions and the behavior during

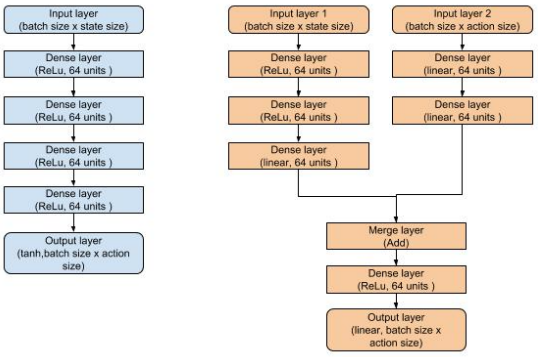


Figure 1: Actor (left) and Critic (right) network architectures

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q\left(s, a \mid \theta^{Q}\right)$ and actor $\mu\left(s \mid \theta^{\mu}\right)$ with weights θ^{Q} and θ^{μ}
Initialize target network $Q^{'}$ and $\mu^{'}$ with weights $\theta^{Q^{'}} \leftarrow \theta^{Q}$, $\theta^{\mu^{'}} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode = 1,M **do**
  Initialize a random process N for action exploration
  Receive initial observation state s_1
  for t=1,T **do**
    Select action $a_t = \mu\left(s_t \mid \theta^{\mu}\right) + N_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, r_t, a_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_t, r_t, a_t, s_{t+1}) from R
    Set $y_i = r_i + \gamma Q^{'}\left(s_{i+1}, \mu^{'}\left(s_{i+1} \mid \theta^{\mu^{'}}\right) \mid \theta^{Q^{'}}\right)$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i \mid \theta^{Q})\right)^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i} \nabla_a Q\left(s, a \mid \theta^{Q}\right) \mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right) \mid_{s_i}$$

    Update the target networks:

$$\theta^{Q^{'}} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q^{'}}$$
$$\theta^{\mu^{'}} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu^{'}}$$

  end for
end for

training is rendered in Gazebo. It is important to note that there is some stochasticity in the initial state(which causes both RL agents and classical control agents to fail) as well as in the model of the quadcopter which we have treated as a black box since our model-free approach does not need to consider it. We have chosen to use an empty Gazebo environment because our project focuses more on the ability to learn control policies to decrease battery utilization rather than obstacle avoidance which has been shown in several previous works to be an easy Reinforcement Learning problem. The environment has also been limited to $\pm 15m$ in x and y and between 0.5m to 5m in the z-direction.

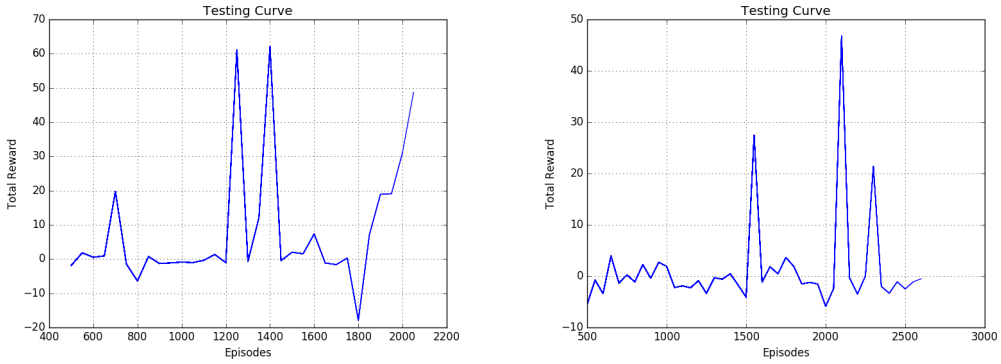
4 Results

Following the methodology explained in the previous section, we successfully trained our DDPG agent to navigate from a start state to a goal state. To quantify the results we generate several reward plots of the agent as well as compare the battery consumption to classical control strategies. For all

results in this section, we test the model for 10 episodes for every saved model and record the mean reward. The best model is determined from the test curve then run for 100 test episodes to evaluate the battery and success rate.

4.1 Single Goal Navigation

The quadcopter was first trained to navigate from a predefined start and goal position with a fixed battery limit of 200 and the network and hyperparameter setting explained in Section 3.4. Starting from $[0, 0, 0]$, the quadcopter was first trained for the further Goal A at $[-10, 10, 3]$ and then to the nearby Goal B at $[5, -10, 2]$. The testing curve for Goal A for episodes 450 to 2100 is shown in Fig. 2a



(a) Test curve for Goal A: [-10,10,3]

(b) Test curve for Goal B: [5,-10,2]

Figure 2: Test curves for single goal navigation

As seen in the testing curve, there are a few models that learned to navigate successfully out of which model 1400 was chosen for testing. The chosen model is seen to give 100% success rate with a mean battery consumption of 128.957. The trajectory followed by the quadcopter is shown in Fig. 3.

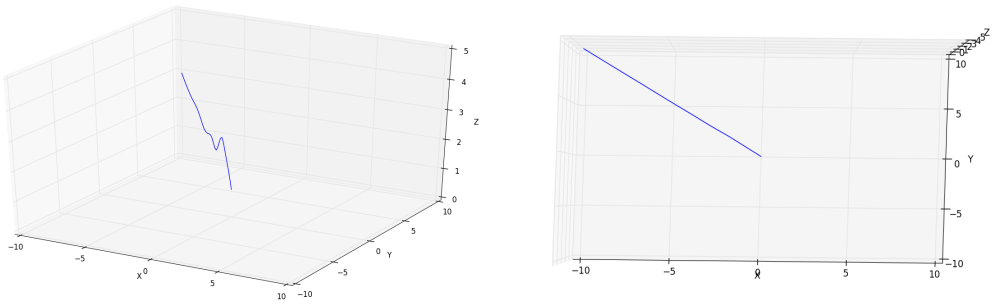


Figure 3: Trajectory followed to goal [-10,10,3] - isometric view (left) and top view (right)

The results for the nearby goal are shown in Fig. 2b and Fig. 4. Here, the agent shows 70% success rate in reaching the goal and a mean battery consumption of 126.516. This proves that the agent takes into consideration the constraint no matter where the goal is, and learns control policies that ensure judicial usage of battery. The remaining 30% of test episodes either involved the agent tipping over during take-off due to bad initial orientation or flying slightly above or below the set threshold of the goal.

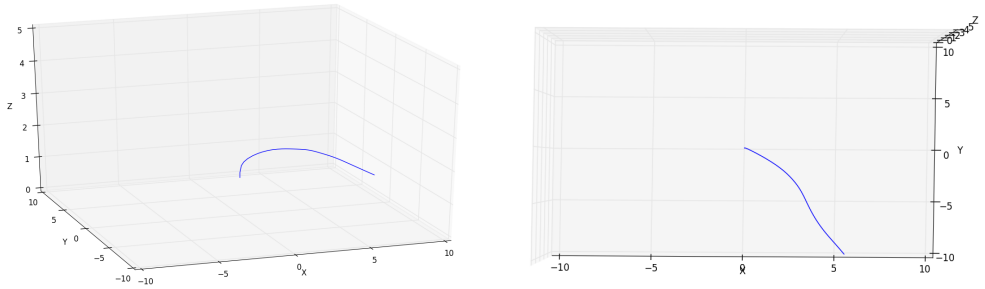


Figure 4: Trajectory followed to goal [5,-10,2] - isometric view (left) and top view (right)

4.2 Influence of battery constraint

The previous results prove that the agent learns to adapt its control velocities depending on the location of the goal, while spending a judicial amount of battery. The next major result is a comparison with another DDPG agent that does not have a battery constraint as its state or terminal condition. The testing curve showed that a few models were able to solve the navigation problem to the $[-10, 10, 3]$ goal, out of which the model with the least battery consumption was tested further. This is seen to give a 75% success rate with a mean battery consumption of 156.98. The 12 episodes where a bad initial state caused immediate failure are not included in this battery consumption calculation as the agent never really takes off. This is roughly **14% more battery consumption** than the DDPG agent trained with the battery constraint. Fig. 5 further demonstrates the impact of battery constraint on the agent. Here, the actor modified to have 3 hidden layers of 64 units while the critic network remained the same.

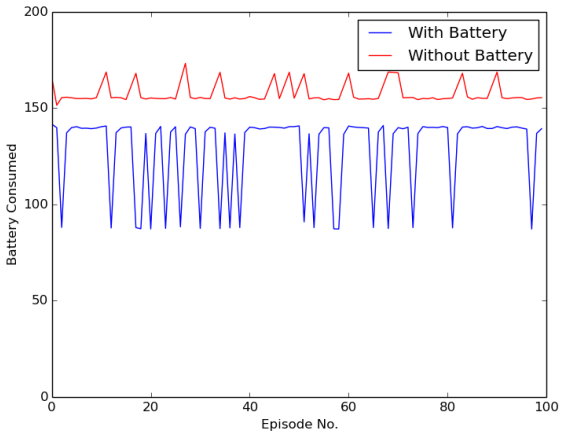


Figure 5: Comparison of battery consumption between the two DDPG agents

4.3 Comparison with Classical Control

In the previous subsection we showed that the battery constraint has helped improve the navigation behavior of the quadcopter. A true test of the battery consumption performance of the agent is to compare it with the performance of a classical control algorithm. We tuned a PID controller to navigate in a straight line from the start state to the goal state. The controller was tuned such that it can reach the goal within the same battery constraints as the DDPG agent.

To compare the battery consumption, we calculate the battery consumption by both the trained DDPG agent and the PID-controllers over 100 episodes with the same start state and goal state. Fig. 6 shows the battery consumption by the DDPG agent and two sets of tuned PID controllers.

We can clearly see that the DDPG agent consumes significantly less battery than the PID-controller while navigating between the same start and goal states. Another observation is that the PID controller is very sensitive to the initial state of the quadcopter which has some stochasticity whereas the DDPG agent is more robust to it. The PID controllers were tuned very carefully to allow them to reach the goal to the best of their ability.

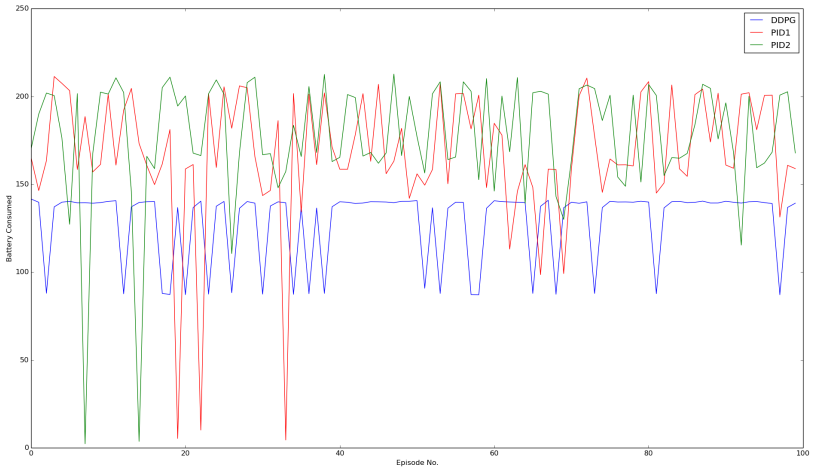


Figure 6: Comparison of battery consumption between the DDPG agent and the PID-controlled agents. PID1: $k_{p_x} = k_{p_y} = 0.36$; $k_{p_z} = 0.1$; $k_{d_x} = k_{d_y} = 0.25$; $k_{d_z} = 0.15$ PID2: $k_{p_x} = k_{p_y} = 0.36$; $k_{p_z} = 0.2$; $k_{d_x} = k_{d_y} = 0.35$; $k_{d_z} = 0.15$

4.4 Randomized Goal Navigation

Navigating to multiple goals is a highly non-trivial task for a Reinforcement Learning agent to learn. We attempted to teach the agent to fly to any goal specified rather than just the goal it was trained on. To do this, we randomly generated 1 of 4 goals for the agent to navigate to in every single episode. The distance between each goal was spaced apart to ensure that the agent cannot simply drift to one goal. Since the scope of the project also includes conserving the battery level, the input to our DDPG was the quadcopter's current position(x,y,z), its current goal location(x,y,z) and its battery level. In doing so, we hoped that the quadcopter could make an informed decision based on its inputs. Furthermore, we also drew inspiration from Hindsight Experience Replay (HER)[10] where a goal position is appended to the normal state, action, reward, next state tuple. Even though HER works best with an environment where the rewards are binary and sparse, HER also showed a performance improvement in our environment.

During training, the agent was able to arrive at the randomized goal specified with almost half of the battery to spare in several occasions. Unfortunately, the network was unable to converge and the desired behavior could not be learned. We trained the agent for approximately 7000 episodes and results were not at the level of the single goal navigation. In most cases, the episode terminates due to instability of the quadcopter control. Fig. 7 shows the test reward plot for multi-goal navigation. In the videos provided, it is evident that the quadcopter is capable of flying in the general direction of the goal but unable to execute the finer controls to bring itself within range to acquire maximum reward and terminate the episode. The agent learns the general behavior but completing the final task of getting close to the goal is a much more complex challenge.

For this setting, the actor and critic networks were modified to have all layers with 128 hidden units as more complex representations had to be learned.

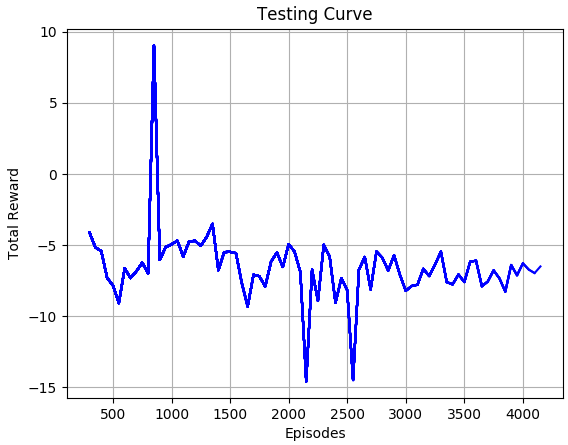


Figure 7: Test reward plot for multi-goal navigation

4.5 Project Links

Videos of our quadcopter's performance under different test conditions can be found at `https://bit.ly/2rnVSgQ`
Our code can be accessed at `https://github.com/rohit-s-murthy/quadcopter-navigation-drl.git`. The README file should help with installation and running instructions.

5 Conclusion

Throughout the project, several important lessons of reinforcement learning were observed. Firstly, reward shaping is crucial to the success of the agent's behavior. A reward function that's purely based on the distance and angle to the final goal will not be sufficient because the agent will learn a less than optimal behavior. A reward based on incremental reward is more robust as it can be written as a difference of potential functions. Furthermore, goal rewards should be well-scaled. In order for the agent to learn to conserve battery, it must be added to the state input as well as the reward function.

Secondly, we showed that the RL approach was better suited for the application where battery consumption is key. The PID controller is capable of getting to the desired goal but requires 36.5% more battery than the DDPG agents.

Thirdly, we found that the agent is able to relate the distance to goal with the battery that it has remaining. This is evident from the fact that the battery consumed when navigating to a closer goal and a further goal is nearly the same. This means that when the agent needs to go further away, it travels slower to ensure that it consumes less battery but travels faster when the goal is closer.

Lastly, generalizing a network to allow the agent to go to any point specified by the network is an extremely difficult task. Our approach was to add the goal for each episode to the input of the network as well as adding it to our experience replay buffer. We managed to get the agent to fly towards the general direction but we were unsuccessful in teaching it to get close enough to the final destination to terminate the episode. As seen in the video provided, the agent is capable of flying in the right direction most of the time, but eventually, the battery gets used up it becomes unstable, thus terminating the episode.

After this project, we have a more profound understanding of the hardships associated with teaching an agent a desired behavior. If we were to continue this project we feel that we would try different ways for the agent to generalize as well as allocate more time for training.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmille. Playing Atari with Deep Reinforcement Learning. https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[2] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, 2016, pp. 528–535. http://rll.berkeley.edu/icra2016mpcgps/ICRA16_MPCGPS

[3] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, Multi-agent quadcopter testbed control design: Integral sliding mode vs. reinforcement learning. Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on. IEEE, 2005, pp. 3712–3717. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1545025

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. CoRR, abs/1509.02971, 2015. http://arxiv.org/abs/1509.02971.

[5] Jemin Hwangbo, Inkyu Sa, Roland Siegwart and Marco Hutter. Control of a quadcopter with Reinforcement Learning. https://arxiv.org/pdf/1707.05110.pdf

[6] Yongguo Mei, Yung-Hsiang Lu, Y.C. Hu, C.S.G. Lee. Energy-efficient motion planning for mobile robots. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1302401

[7] Nirmal Kumbhare, Aakarsh Rao, Chris Gniady, Wolfgang Fink, Jerzy Rozenblit. Waypoint-to-waypoint Energy-efficient Path Planning for Multi-copters.

[8] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, Martin Riedmiller. Deterministic Policy Gradient Algorithms. http://proceedings.mlr.press/v32/silver14.pdf

[9] *hector_gazebo* package. https://github.com/tu-darmstadt-ros-pkg/hector_quadcopter

[10] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, Wojciech Zaremba. Hindsight Experience Replay. arXiv:1707.01495 https://arxiv.org/abs/1707.01495

[11] https://github.com/yanpanlau/DDPG-Keras-Torcs.git

[12] Andrew Ng, Daishi Harada and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. International Conference on Machine Learning 1999. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/NgHaradaRussell-shaping-ICML1999.pdf