Deep Reinforcement Learning based Continuous Control for Multicopter Systems

Anush Manukyan
University of Luxembourg, Luxembourg
Luxembourg, Luxembourg
anush.manukyan@uni.lu

Miguel A. Olivares-Mendez
University of Luxembourg, Luxembourg
Luxembourg, Luxembourg
miguel.olivaresmendez@uni.lu

Matthieu Geist
Université de Lorraine & CNRS, LIEC
Metz, F-57070 France
matthieu.geist@univ-lorraine.fr

Holger Voos
University of Luxembourg, Luxembourg
Luxembourg, Luxembourg
holger.voos@uni.lu

***Abstract*—In this paper we apply deep reinforcement learning techniques on a multicopter for learning a stable hovering task in a continuous action state environment. We present a framework based on OpenAI GYM, Gazebo and RotorS MAV simulator, utilized for successfully training different agents to perform various tasks. The deep reinforcement learning method used for the training is model-free, on-policy, actor-critic based algorithm called Trust Region Policy Optimization (TRPO). Two neural networks have been used as a nonlinear function approximators. Our experiments showed that such learning approach achieves successful results, and facilitates the process of controller design.**

***Index Terms*—reinforcement learning, UAV, multicopter, TRPO, neural network**

I. Introduction

Unmanned aerial vehicles (UAV) are being increasingly deployed in many civil applications owing to their high flexibility, possibility to carry a wide range of sensors, inexpensive cost and hovering abilities. They are already being used for tasks such as remote sensing and monitoring of objects, search and rescue operations and even goods delivery. However, providing an autonomous and stable navigation in an unknown environment, remains a challenging problem. At present, the control systems of the most industrial UAVs are predominantly based on Proportional, Integral Derivative (PID) controller. It has an advantage of having a simple design, easily adaptive parameters and good robustness [1]. In case of stable environments PID controller has close-to ideal performance [4]. However, it requires a complete knowledge of the environment and system dynamics. And even small changes in the environment can lead to failure. Therefore, having an intelligent control system, that can adapt to the changes of the environmental dynamics is desirable.

In recent years Reinforcement Learning (RL) based control systems became a subject undergoing intense study among robotics researchers. RL is a framework that offers to the field of robotics different goal-oriented tools, for designing hard-to-engineer behaviors [2]. The general idea behind the RL is that by autonomously interacting with its environment the agent learns an optimal policy without having a prior knowledge about its dynamics nor environment.

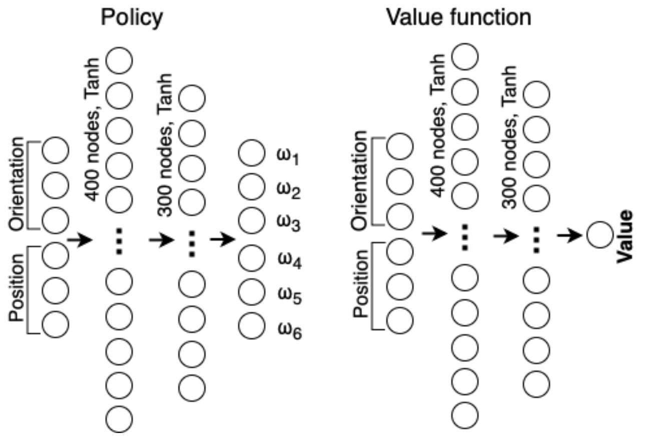

Fig. 1. The neural network architectures for policy and value function approximations used in this work.

RL algorithms have already been deployed in various robotic applications rainging from autonomous navigation to object recognition and manipulation. Our investigations in recent works showed that in many UAV applications RL techniques outperform PID controllers [4]. In this work we will focus on studies that apply RL methods on UAV applications.

For instance, Bou-Ammar et al [5] propose a nonlinear autopilot for quadrotor UAVs based on feedback linearization for precise and fast stabilization. Their approach is based on reinforcement learning, where after each generated control action the agent receives a feedback by a simple reward function. Using a value iteration method, they achieve faster convergence of the learning process. Pham et al. [6], [7] provide a RL based framework for an UAV to navigate to a target point in an unknown environment, where an exact mathematical model of the environment may not be available. The purposed framework is based on the combination of Q-learning algorithm and a PID controller. By performing experiments in the simulation and real world using a quadrotor Parrot AR.Drone 2.0, they show that the UAV successfully manages to find the least number of steps to navigate to the goal position.

A several studies address the learning problem applied on hovering tasks. For instance, Sugimoto et al. [8] use reinforcement learning for achieving stable hovering of an actual quadrotor UAV. The main idea of their work is to use a marker that the UAV should recognize and hover above it. First, an image taken by the camera of a drone is sent to the PC, and after velocity commands are calculated and send to the drone. For the learning process they use Q-learning algorithm which creates a matrix of states. The drone receives positive rewards every time when it ends up in the state where the marker is. By performing real world experiments using an AR.Drone 2.0, they show that the UAV successfully learns to hover over a given marker. Another RL based method for controlling a quadrotor UAV is presented by Hwangbo et al. [9], where they propose a novel learning algorithm. They use neural networks in order to train a quadrotor to stabilize itself in the air even from manually thrown upside-down position. The algorithm uses two different neural networks: value network and policy network. The algorithm gets the elements of the rotation matrix as an input, and outputs the rotor trust. They demonstrate the successful performance of the trained policy first in a simulation and after on a real quadrotor.

There is a considerable amount of literature that investigate the advantages of utilizing reinforcement learning for engineering intelligent controllers for the UAVs. Many studies apply on their tasks well known reinforcement learning algorithms such as Q-Learning, SARSA or Deep Q Networks. Despite the fact that these algorithms achieve successful robot training, yet they can be applied only for tasks that operate in a discrete and low-dimensional action space environments, limiting the functionalities of the robot. However, the most practical physical control tasks belong to high-dimensional and continuous action space domain. Hence, in this work we focus on reinforcement learning tasks, that observe continuous robotic control problems. First we combine OpenAI GYM, Gazebo simulator and RotorS MAV Simulator into one framework referred as RotorS Gym framework. We utilize a model-free, on-policy, actor-critic based algorithm called Trust Region Policy Optimization (TRPO) for training our agent to perform a hovering task in RotorS Gym framework. TRPO is an algorithm that aims to solve the classic reinforcement learning problem of maximizing the cumulative return. Since TRPO adopts the actor-critic architecture it uses two different neural networks: one for policy function approximation and another for value function approximation. To improve its policy, TRPO tries to maximize the expectation of Q-values, over the distribution of states and actions [12].

The remaining of the paper is organized as follows. Section II provides details about the multicopter kinematics and dynamics, and presents the basis of reinforcement learning and TRPO algorithm. Section III introduces the structure of our training framework in great detail. Secton IV gives full description of the environmental setup, reward function, hyperparameters and the network architecture, and presents the obtained results. Some of our conclusions are drawn on the final section V.

II. Background

In this section we present a brief introduction of kinematics and dynamics of a multicopter along with the learning algorithm used for training the UAV.

A. *Kinematics and Dynamics of Multicopter*

The schematic representation of the hexicopter used in this work is shown figure 2. We consider two frames: 1) the world-fixed inertial frame, F_I, and 2) the body fixed frame, F_B, which is attached to the hexicopter. The origin of the body frame is in the center of mass of the hexicopter. The hexicopter is actuated by changing the speeds of six rotors. The coordinates of F_I is denoted as X_I, Y_I, Z_I, and X_B,Y_B, Z_B are the coordinates of F_B, respectively. The position of the center of mass of the hexicopter, expressed in the inertial frame F_I is represented by $\Omega = [x, y, z]$. The attitude is described by roll, pitch and yaw, which are the ZYX Euler angles. Finally, the rotation of the body with respect to inertial frame is presneted by rotation matrix R:

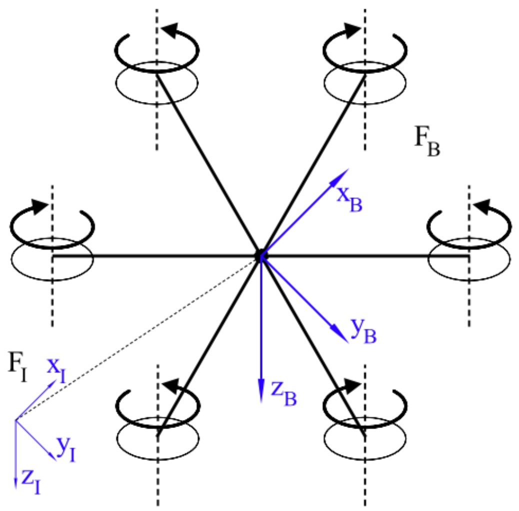

Fig. 2. The average reward obtained per policy iteration.

$$R = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\theta \\ s\psi c\theta & s\psi s\theta s\phi - c\psi c\phi & s\psi s\theta c\phi - c\psi s\theta \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}$$

where $c\psi = cos\psi$, $c\theta = cos\theta$, $c\phi = cos\phi$, $s\psi = sin\psi$, $s\theta = sin\theta$, $s\phi = sin\phi$, and ϕ, θ, ψ are the roll, pitch and yaw angels respectively [10], [11].

More detailed description of mathematical models about quadrotor dynamics are represented in [21]–[23].

B. Reinforcement learning

Using RL method an agent learns an optimal policy by trial-and-error interactions with its environment without having any prior knowledge about it. The agent receives a positive or negative reward for each action taken. Having a goal of maximizing the cumulative reward, the agent learns to take the correct sequence of actions. Such task can be described as a Markov decision process (MDP), which is defined using $(S, A, P, r, \rho_0, \gamma)$ tuple, where S is the finite set of states, A is the finite set of actions, P is the transition probability distribution being at a state $s \in S$ and taking an action $a \in A$ to transient to a new state $s' \in S$, r is the reward function, which maps the state-action-pair to the set of real numbers [5], ρ_0 is the distribution of the initial state s_0 and $\gamma \in (0,1)$ is the discount factor.

The idea behind the procedure of MDP is as follows: the agent starts at an initial state $s_0 \in S$, performs an action $a_0 \in A$ which transfers it to a new state s_1 by transition probability distribution of $P_{s_0 a_0}$, and receives a reward of $r(s_0)$. The process repeats until all states have been visited. We will denote the stochastic policy by $\pi : S \times A \to [0,1]$, and the expected discounted reward $\eta(\pi)$. The main objective of the agent is to choose a sequence of actions over time, that maximizes the expected discounted reward $\eta(\pi)$:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\textstyle\sum_{t=0}^{\infty} \gamma^t r(s_t) \right],$$

where $s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ [12]. Based on the standard definitions the value function V_π, the state-action value function Q_π and the advantage function A_π are following:

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\textstyle\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right],$$

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\textstyle\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right],$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s),$$

where $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$ for $t \geq 0$.

As was mentioned before, TRPO adopts the architecture of the actor-critic method, but it modifies the way the policy parameters of the actor are being updated.

The new policy will be denoted as $\tilde{\pi}$ and $\eta(\tilde{\pi})$ will be the expected return of the new policy $\tilde{\pi}$.

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}} \left[\textstyle\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \quad (1)$$

In (1) the advantage function is used which measurs how good the new policy is with regard to the average performance of the old policy. $\eta(\tilde{\pi})$ can be rewrite into the following form, where instead of the sum over timesteps is the sum over states [12]:

$$\begin{aligned} \eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_s P(s_t = s|\tilde{\pi}) \sum_a \tilde{\pi}(a|s) \gamma^t A_\pi(s, a) \\ &= \eta(\pi) + \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_a \tilde{\pi}(a|s) A_\pi(s, a) \\ &= \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a), \end{aligned} \quad (2)$$

where π is the old policy, $\tilde{\pi}$ is the new policy and ρ is the discounted visitation frequencies: $\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots$.

However, (2) is hard to be optimized due to the strong dependency of $\rho_{\tilde{\pi}}$ to the new policy $\tilde{\pi}$. Therefore, a local approximation to η is used:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a). \quad (3)$$

In (3) L_π uses the state visitation frequency ρ_π instead of $\rho_{\tilde{\pi}}$, assuming that the state visitation frequency of the new policy is not too different from the old policy. Kakade and Langford [12] proved that small steps that optimizes the local approximation $L_{\pi_{\theta_{old}}}$ also improves η. Hence, in order to find an optimal step size, Kakade and Langford proposed a conservative policy iteration update scheme:

$$\pi_{new}(a|s) = (1 - \alpha)\pi_{old}(a|s) + \alpha \pi'(a|s), \quad (4)$$

where π_{old} is the current policy, $\pi' = argmax_{\pi'} L_{\pi_{old}}(\pi')$ and π_{new} is the new policy.

After they obtained the following bound which constraints the policy update to be within some trust region:

$$\eta(\pi_{new}) \geq L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \quad (5)$$

where $\epsilon = \underset{s}{max} \left| \mathbb{E}_{a \sim \pi'(a|s)} \left[A_\pi(s, a) \right] \right|$.

By replacing α with a distance measure between π and $\tilde{\pi}$ the policy improvement bound in (5) can be extended to general stochastic policies.

Involving KL-divergence between the new policy and the old policy the bound above can be written as follows:

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - C D_{KL}^{max}(\pi, \tilde{\pi}), \quad (6)$$

where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$ and represents the penalty coefficient, D_{KL}^{max} denotes the maximum KL divergence of the two policies.

From (6) implies that the generated sequence of policies are monotonically improving: $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \ldots$. The right-hand-side of the equation (6) will be repalced with $M_i(\pi)$: $M_i(\pi) = L_{\pi_i}(\pi) - C D_{KL}^{max}(\pi_i, \pi)$. Then from (6) $\eta(\pi_{i+1}) \geq M_i(\pi_{i+1})$, since the KL divergence between π_i and π_i is 0, then $\eta(\pi_{i+1}) \geq M_i(\pi_{i+1})$, therefore, $\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M_i(\pi_i)$. This means that by maximizing

M_i at each iteration, the objective function η is always improving.

Finally, the optimization problem that this algorithm solves is the following:

$$\underset{\theta}{maximize}[L_{\theta_{old}}(\theta) - CD_{KL}^{max}(\theta_{old}, \theta)].$$

III. Framework Structure

Training a real robot in a real world, using RL techniques, can be expensive due to large number of explorations that the agent performs, and many of which result in crashes damaging the robot. Therefore, having a framework that provides close to real-world dynamics is necessary. There, the training of the robot can be performed safely and the optimal policy can be learned efficiently, which can later be run on the real robot, without any changes. Ideally, the agent should not be able to differentiate between a simulated environment and the real world.

In this work we designed such framework by combining a few open source toolkits, such as: OpenAI Gym, Gazebo simulator and RotorS Micro Aerial Vehicle (MAV) Simulator Framework.

$OpenAI\ Gym$ is a toolkit that provides several robotic environments for developing and comparing reinforcement learning algorithms [13]. $Gazebo$ is a free and robust physics engine for robot simulations. One of the main advantages of Gazebo is that it gives possibility to not only run the simulation using a high-quality graphical interface, but also allows to run in a headless mode, which is crucial for faster training. The combination of OpenAI Gym and Gazebo has been presented by Zamora et al. [14] where they take the baseline structure of OpenAI Gym and build a Gazebo environment on the top of it. The aim of such toolkit is to facilitate the use of the reinforcement learning algorithms for training robots in a real world environments. $RotorS$ is a modular, gazebo-based MAV simulator, which includes a position controller and a state estimator. The main motive behind the choice of using the RotorS simulator is that the structure of the simulator is analogous to the real system [15], and without applying any changes to the model, all components used in the simulated environment can be run on the real platform [15]. In addition, the MAV models, simulated by the Gazebo physics engine contain all the components that are found on the real MAVs [15].

RotorS provides several UAV multirotor models. In this work we have chosen to use AscTec Firefly as a learning agent.

By combining all these tree powerful tools together, we obtained a new RotorS Gym framework, which provides a real-world like environment for training our agents. The communication between RotorS simulator and OpenAI Gym is executed through Robot Operating System (ROS). The architecture of RotorS Gym framework is shown on figure 3.

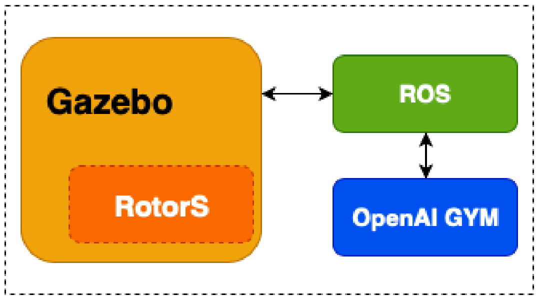

Fig. 3. RotorS Gym framework architecture.

RotorS Gym framework provides an environment consist of a continuous action space and an observation space. Similar to OpenAI Gym, the learning process in RotorS Gym is being execute in an episodic settings, where the agent learns to perform the given task through series of episodes. In each episode, the agent interacts with its environment until the environment reaches a terminal state. There are a few ways to define the terminal states: (1) by defining constraints, (2) by defining a timestep. In this work the agent can interact with its environment until it breaks down the defined constraints or if it overpasses 1000 steps, meaning that it performs 1000 actions during one episode.

IV. Experimental Validation

In this section we discuss in great details about the continuous action state, the reward function and the neural network structure. We conclude the section with presenting the obtained results.

A. Experimental Setup

1) Environmental details: This work focuses on reinforcement learning tasks, that observe continuous robotic control problems. As noted in section III, RotorS Gym framework, used for training an agent to perform a hovering task, is in continuous action space and observation space domain. It inherits two environment-specific functions of OpenAI Gym, which are $reset$ and $step$. The $reset$ function is called when an episode starts in order to reset the environment and set up the initial environmental conditions. The $step$ function is used for executing each simulation step. At each step it returns the following three important values:

- **observation** is an environment-specific object, containing information about the position and orientation of the agent, along with the rotors velocities.
- **reward** is a floating point number that the agent gets after each performed action. More details about the reward function used in this work is given below.
- **done** is a boolean value that indicates whether it is time to reset the environment. Done is $True$ when the maximum amount of steps have been accomplished or if the agent overpasses the predefined constraints.

2) Workflow of the Controller: The process of the main workflow of the controller, shown on figure 4, is as follows: it receives the position and orientation of the UAV as an input and outputs the next action which consists of six rotor velocities. Since the action space is continuous at each step six velocity values are being chosen from predefined bounds. The bounds have been chosen by experimenting with different velocity values in order to find a physically feasible constraints that lead to a realistic performance of the UAV. For AscTech Firefly we have chosen $\Omega_{min} = 535$ and $\Omega_{max} = 615$. It is worth noting that for a task such as takeoff of Firefly drone, the minimum velocity value in the RotorS simulator should be $\Omega_{min} \geq 545$.

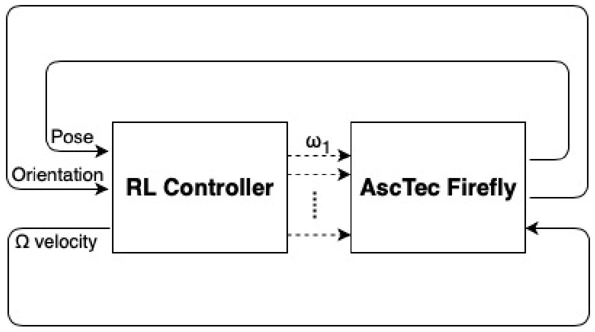

Fig. 4. The workflow of the controller developed in RotorS Gym framework.

3) Reward Function: Another challenging point in reinforcement learning is to design the reward function, since it is the only parameter that indicates the agent's performance. For instance, sparse reward functions leads to a poor performance of an agent, since it does not give sufficient information to the agent about its behaviour. In case of sparse reward the agent gets feedback only when it succeeds or fails the task. Meaning that, while the agent is exploring its environment, it does not receive any information if it is performing good or worse. And only when it stumbles to the success condition then it can learn the sequence of actions that brought it there. In this work we focus on designing shaped reward function, since it gives smooth, continuous gradual information using the UAV's position and orientation. Our experiments showed that such reward function helps the agent to converge to an optimal policy in a reasonable amount of time.

4) Hyperparameters: For the best results the choice and tuning of the optimal hyperparameters is essential. Poor hyperparameter selection can cause weaker performance of the algorithm. In this work the base values of the hyperparameters originate from the following related literature [16]–[19]. Moreover, an optimization of the baseline hyperparameters has been done to improve the performance of the presented approach. Table I represents the hypermameter values used in this work.

5) Network Architecture: Function approximators are another core component in reinforcement learning, which must be chosen and designed carefully. They have significant impact in finding the optimal policy. In this work the function approximator is represented by a neural network. As stated before TRPO employs an actor-critic architecture, where the actor model is the policy function and the critic model is the value function. Hence, two different neural networks have been constructed representing the policy and value function approximations. As shown on figure 1, first, the policy network

TABLE I
HYPERPARAMETERS

Hyperparameter	**Value**
Discount (γ)	0.995
Learning Rate	0.0005
max KL	0.01
L2 regularization	0.001
Batch Size	64
Stepsize (D_{KL})	$1e$ - 3
Random Seed	10
Timestep per episode	1000

gets the position and orientation of UAV and outputs a vector of six rotor velocities. After the neural network representing the value function returns a value which is the evaluation of the behavior of the UAV based on its position and orientation.

B. Results

Both the learning and testing phases have been performed in the RotorS Gym framework using AscTec Firefly drone. The learning phase starts with the UAV being at a given starting point, such as $z = 1.5$, with a goal of hovering around this position. Several positional constraints have been predefined in order to pinpoint a failure and reset the environment.

The UAV learns to perform the given task by training for $\sim$ 2100 policy iterations, which is approximately 2 million simulation steps and equal to $\sim$ 15 hours of learning.

Our performance metric is the average reward obtained per policy iteration, represented on figure 5. It is seen from the figure that the UAV stably increases the cumulative reward during the learning phase.

All experiments and training have been executed on a machine, equipped with $16GB$ of RAM and a $2.6Ghz$ Intel i7-6600U CPU with an $Ubuntu16.04$ operating system.

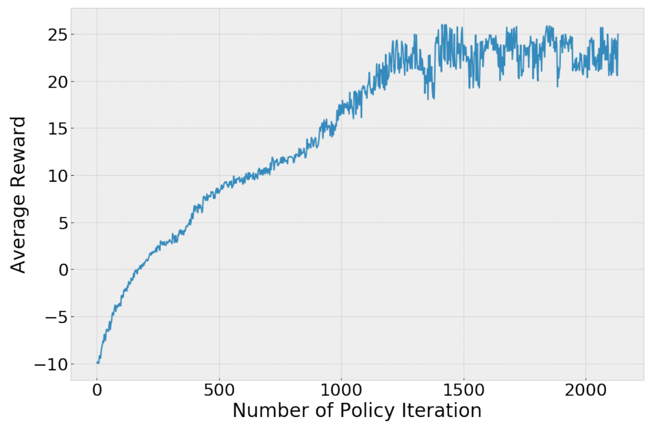

Fig. 5. The average reward obtained per policy iteration.

V. Conclusion

In this work we presented a learning approach that trains a multicopter to hover in a continuous action space environment. The algorithm used for solving such task is model-free, on-policy method, called Trust Region Policy Optimisation (TRPO). TRPO adopts the architecture of the actor-critic method, using two neural networks as nonlinear function approximators. The first neural network that is used for policy function approximation gets the position and orientation of the UAV as input, and outputs six continuous actions representing six rotor velocities of the multicopter. After, the second neural network used for value function approximation evaluates the learnt policy. Such learning process is performed in the RotorS Gym framework, which is a combination of OpenAI Gym, Gazebo and RotorS MAV simulator. For the training an AscTec Firefly drone is used in the RotorS Gym Framework.

The whole learning procedure is performed in about 2100 policy iterations which is about 2 million simulation step. Our results showed that using such learning technique it is possible to successfully train an agent to perform a stable hovering task in a continuous action space domain.

As a future work the hovering task can be extended to navigation or even more complex task. Another learning approach, such as Proximal Policy Optimization [20] can be used which may significantly increases the speed and stability of convergence on various continuous control tasks.

REFERENCES

[1] Zulu, Andrew, and Samuel John. “A review of control algorithms for autonomous quadrotors.” arXiv preprint arXiv:1602.02622(2016).

[2] Kober, Jens, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey.” The International Journal of Robotics Research 32.11 (2013): 1238-1274.

[3] Polvara, Riccardo, et al. “Autonomous Quadrotor Landing using Deep Reinforcement Learning.” arXiv preprint arXiv:1709.03339 (2017).

[4] Koch, William, et al. “Reinforcement Learning for UAV Attitude Control.” arXiv preprint arXiv:1804.04154 (2018).

[5] Bou-Ammar, Haitham, Holger Voos, and Wolfgang Ertel. “Controller design for quadrotor uavs using reinforcement learning.” Control Applications (CCA), 2010 IEEE International Conference on. IEEE, 2010.

[6] Pham, Huy Xuan, et al. “Reinforcement Learning for Autonomous UAV Navigation Using Function Approximation.” 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE, 2018.

[7] Pham, Huy X., et al. “Autonomous uav navigation using reinforcement learning.” arXiv preprint arXiv:1801.05086 (2018).

[8] Sugimoto, Takuya, and Manabu Gouko. “Acquisition of hovering by actual UAV using reinforcement learning.” Information Science and Control Engineering (ICISCE), 2016 3rd International Conference on. IEEE, 2016.

[9] Hwangbo, Jemin, et al. “Control of a quadrotor with reinforcement learning.” IEEE Robotics and Automation Letters 2.4 (2017): 2096-2103.

[10] Balasubramanian, E., and R. Vasantharaj. “Dynamic Modeling and Control of Quad Rotor.” International Journal of Engineering and Technology (IJET) 5 (2013): 63-69.

[11] Imanberdiyev, Nursultan, et al. “Autonomous navigation of UAV by using real-time model-based reinforcement learning.” Control, Automation, Robotics and Vision (ICARCV), 2016 14th International Conference on. IEEE, 2016.

[12] Schulman, John, et al. ”Trust region policy optimization.” International Conference on Machine Learning. 2015.

[13] Brockman, Greg, et al. “Openai gym.” arXiv preprint arXiv:1606.01540 (2016).

[14] Zamora, Iker, et al. “Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo.” arXiv preprint arXiv:1608.05742 (2016).

[15] Furrer, Fadri, et al. “RotorsA modular gazebo mav simulator framework.” Robot Operating System (ROS). Springer, Cham, 2016. 595-625.

[16] Henderson, Peter, et al. ”Deep reinforcement learning that matters.” Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[17] Duan, Yan, et al. ”Benchmarking deep reinforcement learning for continuous control.” International Conference on Machine Learning. 2016.

[18] Nachum, Ofir, et al. ”Trust-pcl: An off-policy trust region method for continuous control.” arXiv preprint arXiv:1707.01891 (2017).

[19] Islam, Riashat, et al. ”Reproducibility of benchmarked deep reinforcement learning tasks for continuous control.” arXiv preprint arXiv:1708.04133 (2017).

[20] Schulman, John, et al. ”Proximal policy optimization algorithms.” arXiv preprint arXiv:1707.06347 (2017). APA

[21] Omidshafiei, Shayegan. ”Reinforcement learning-based quadcopter control.” (2013).

[22] Alaimo, A., et al. ”Mathematical modeling and control of a hexacopter.” Unmanned Aircraft Systems (ICUAS), 2013 International Conference on. IEEE, 2013.

[23] Lippiello, Vincenzo, and Fabio Ruggiero. ”Exploiting redundancy in Cartesian impedance control of UAVs equipped with a robotic arm.” Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. IEEE, 2012.