Deep-RL-based safety landing using RGB camera on rough terrains

Nicola Loi
Department of Mechanical and Process Engineering
ETH Zurich
nicloi@student.ethz.ch

Valentin Ibars
Department of Electrical Engineering and Computer Sciences
ETH Zurich
vibars@student.ethz.ch

Abstract:
This project report aims to present the feasibility of using reinforcement learning for the task of landing autonomous MAVs in challenging environments. In this work, a framework is presented for training and testing reinforcement learning agents in photo-realistic simulation environments built on Unreal Engine. The effects of model shaping on the learning outcome of the agent are explored, followed by an evaluation and discussion of the respective model configurations.

Keywords: CORL, Reinforcement Learning, PPO, Drones, Emergency landing

1 Introduction

The reinforcement learning field is getting more and more attention in the last decade, with multiple implementations being developed, such as Deep Q-Network (DQN) [1], Asynchronous Advantage Actor Critic (A3C) [2], Trust Region Policy Optimization (TRPO) [3], and Proximal Policy Optimization (PPO) [4]. Reinforcement learning is studied and researched on various applications, mostly as an alternative to supervised learning, to get better performance and overcome its limits.

In this paper is presented a reinforcement learning framework, to experiment reinforcement learning performance on MAV autonomous landing, employing Proximal Policy Optimization. Most of today's drone applications require to have the ability to land safely at the end of the task, to not damage or destroy the drone. The specific environment of our experiments will be a glacier, in the context of the many glacier monitoring projects carried out by the Autonomous System Lab of ETH. A glacier is a challenging environment to land in, due to the roughness of the terrain; landable spots are difficult to recognize, due to the slope, the reflection, and the homogeneous grayish texture of the ground. The application of reinforcement learning to such a problem benefit from the global understanding of the environment and is today an active area of research [5], [6], [7].

The advantage of reinforcement learning over supervised learning on this particular task of safe landing is given by the ability to learn how to explore the environment and how to behave in particular states, where the learned experience plays a decisive role in the best action to perform.

2 Related Work

Autonomous landing system involving reinforcement learning is an active area of research. In this section some research and related works are presented to give the reader a sense of what approaches have been investigated thus far.

The first approach by Rodriguez-Ramos et al. relies on a deep-RL based strategy for UAVs autonomous landing on a moving platform [5]. In this paper, they use Deep Deterministic Policy Gra-

5th Conference on Robot Learning (CoRL 2021), London, UK.

dient (DDPG) [8] from OpenAI as the reinforcement learning framework to provide commands for UAV to land on a moving platform. According to the paper, many experiments have been conducted in different conditions, and they demonstrated there model capability to generalize on multiple scenarios. Hence paper's authors present a powerful work flow for robotics, where robots can learn in simulation and perform properly in real operation environments.

Another approach from Lee et al. is a vision-based method for autonomous landing of UAVs using Actor-Critic reinforcement learning framework [6], which is used to provide to the agent the commands to move it. The paper's author claim that their reinforcement learning framework provide comparable result to human's made guidance method.

The final method we would like to present in this section is a segmentation based approach for fixed wing UAVs presented by Hinzmann et al. [7]. In their approach, they propose a perception system using RGB image produced by an on-board, downward facing camera to detect landing sites based on their texture and geometric shape without any prior knowledge about the environment. The image segments are filtered down to the largest, connected areas and each is then classified as either potentially landable or not. Random forest and Extended Kalman Filters are respectively used to classify landable zone and estimate UAV pose considering the local wind. Throughout the landing approach, the regions are tracked and a decision layer produces a mask designating suitable landing sites, taking into account the terrain roughness and slope.

3 The Framework

In this section we will introduce our approach by presenting our general framework as well as the building blocks.

3.1 Reinforcement learning Framework

Reinforcement learning is described by [9] as a paradigm of machine learning with two principal components: an agent and an environment. The agent has to learn how to act in an environment in order to maximise its reward function. The famous dilemma in reinforcement learning is the challenge of exploitation vs exploration. The agent should perform the actions that lead to high rewards, but on the other hand, it needs to also try different, and probably wrong, actions to discover which ones lead to higher rewards. Many possible methods exist to implement this concept to the problem of autonomous landing on a rough terrain. We choose to use Proximal Policy Optimization from OpenAI, specifically its second version (PPO2) optimized for GPUs, introduced by Schulman et al. in [4] for a discrete action space. This algorithm combines ideas from A3C[2] by having multiple workers and TRPO[3] as it uses a trust region to improve the actor. PPO approach was used in this project as it is stable, reliable, simpler to tune and to initialize, it allows some flexibility with the choice of the architecture and has good overall performance.

PPO is an on-policy algorithm with two main features: it exploits first order derivatives such as gradient descent, but approximates the second order derivatives through soft constraints to reduce the complexity, and it keeps the new calculated policy close to the old one, thus the name Proximal Policy Optimization, dissuading high policy updates if they are outside the trust zone.

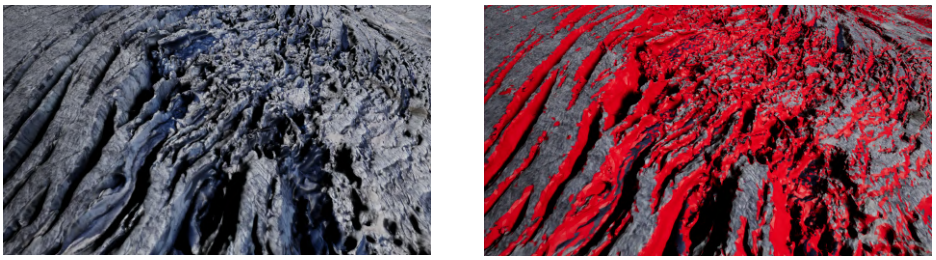
From the PPO's implementation of the `stable-baselines3` library [10], two policy networks are tested: the multi-layer perceptron policy (Mlp) and the convolutional neural network policy (Cnn). The Mlp is a fully connected layer with two hidden layers of 64 neurons each, while the Cnn is composed as following:

- layer 1: channels=32, kernel size=8, stride=4;
- layer 2: channels=64, kernel size=4, stride=2;
- layer 3: channels=64, kernel size=3, stride=1;
- layer 4: fully connected size=512.

3.2 Simulation environment and setting

The training and the testing of the reinforcement learning algorithm are effectuated in a simulated environment of the Swiss Gorner glacier. The model and the texture of the glacier are used to create

a photo-realistic simulation on Unreal Engine 4.16. The dimension of the mesh is 1 $\times$ 0.5 km, and it contains about 1 million triangles. For the rewards of the reinforcement learning, it is needed a ground truth label for each triangle, to classify it as a safe landable spot or not. The data about the slope, the local roughness and the dimension of each triangle are extracted; the roughness is calculated using the arithmetic mean deviation of the normals (Ra). From these data, it is generated a ground truth label: if a triangle has a slope of less than 30°and with a local roughness of less than 0.5 or with at least 0.7 m of space available, the triangle is classified as landable, otherwise it is classified as not landable. With the chosen thresholds, only 23% of the triangles are labelled as landable, but considering the area as seen from above by a drone, the actual landable area is 52%. This difference is due to the fact that the regions of the mesh with high roughness have an high density of smaller triangles, since many more triangles and with smaller size are needed to represent a rough area with respect to a flat piece of land. Figure 1a shows a part of the mesh and texture of the simulated glacier environment, while Figure 1b shows the same mesh, but the triangles with the non landable label are colored in red, for visualization purpose only.



(a) Environment with original texture

(b) Environment with non landable spots in red

Figure 1: Snapshot of the mesh representing the glacier used as environment.

A camera is added to the environment, representing the camera mounted under the drone looking downwards towards the ground with a pitch of -90°. To be able to move the camera (the drone) within the simulation and to receive images from it, it is used `UnrealCV` [11], an open source Unreal Engine plugin. It will start a TCP server that can be contacted by a client embedded inside the reinforcement learning code. From this simulation, it is then needed to establish an environment specifically constructed for reinforcement learning. For this task the `gym` module from OpenAI [12] and `gym-unrealcv` [13] are exploited. A representation of the framework is illustrated in Figure 2.

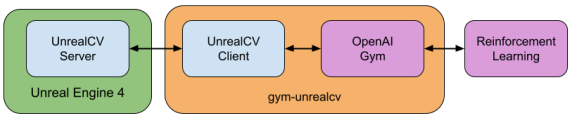


Figure 2: General framework of the project

3.3 State and action spaces

The state space of the agent is constructed from the pixels of the RGB image taken by the drone camera. For the Mlp policy, the image is first scaled to the size 256×256, and then fed to the pretrained mobilenetv2 feature extractor architecture [14] provided by PyTorch, after standardize it with mean = [0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225] to match the statistics of the pretrained network; the features are then fed to the fully connected layer. The Cnn policy on the other hand uses the normalized RGB image at its original size as input.

The displacement of the drone within the environment is described by a discrete action space. Two variants of the action space are tested: one with three-dimensional and one with two-dimensional movements. The three-dimensional variant is composed of 6 actions: up, down, left, right, forward, and backward; it is not present a specific action to land: when the agent is at 2 meters from the ground, it is automatically considered landed. The two-dimensional variant has 5 actions and is constructed to move the agent in a two-dimensional plane, without vertical movements. This action space is tested to see if a simplification of the problem will lead to an increase in the performance.

Instead of moving both horizontally and vertically, the agent is restricted to move only horizontally during the searching of a safe spot to land in. Since there is no more vertical displacement towards the ground, the landing is implemented with a specific action. The complete actions are: left, right, forward, backward, and land. At the beginning of each training or testing episode the agent spawns at a closer height above ground with respect to the three-dimensional case.

3.4 Reward shaping

Reward shaping describes the process of defining a reward function to teach the agent which action to take depending on its state, to maximize the cumulative reward for the entire episode. For the problem of autonomous landing, we introduce different reward terms to achieve the following goals:

- land in a safe spot;
- take a short path to a safe spot.

To fulfill these goals the reward function is composed of multiple parts. Here the multiple components that are used in creating the reward function are presented. Some are functions themselves, depending on information from the environment, such as ground distance or its slope, some depends on the action performed, while others are simply constant values.

- height penalty, to penalize high distance from the ground: $r_{height} = -s_{height_1} tanh(\frac{dist}{s_{height_2}})$;
- elapsed time penalty, to encourage the agent to take the shortest path: $r_{time} = k_{time}$
- action penalty, to penalize up action and to a lesser degree horizontal actions while rewarding down action; if the chosen action is up: $r_{act} = -\ 1.1\frac{stepscale}{s_{act}} dist$, if the action is down: $r_{act} = 0.9\frac{stepscale}{s_{act}} dist$, while for the horizontal movements: $r_{act} = -\ 0.1\frac{stepscale}{s_{act}} dist$;
- slope/roughness penalty, to penalize the agent when flying above non landable spots, based on the slope and roughness of the ground below; when slope$\geq$30: $r_{slope/rough} = -\frac{slope}{s_{slope}}$, and when there is also roughness ≥ 0.5, the penalty has the additional term $-\frac{roughness}{s_{rough}}$;
- landable reward, to reward the agent when landing on a landable spot: $r_{succ} = k_{succ}$
- non landable penalty, to penalize the agent when it lands on a non landable spot: $r_{fail} = k_{fail}$
- other termination penalties, to penalize the agent when it collides with the environment: $r_{coll} = k_{coll}$ or when it exceeds the maximum number of steps: $r_{max} = k_{max}$.

dist is the distance of the agent from the ground in meters, $stepscale$ is the displacement of the discrete action in meters, s_{height_1}, s_{height_2}, s_{act}, s_{act} ,s_{slope}, and s_{rough} are positive scale factors to individually scale each segment of the reward function, and k_{time}, k_{succ}, k_{fail}, k_{coll}, and k_{max} are constant values representing single components of the total reward function. While r_{height}, r_{time}, and r_{act} are present at each time step, $r_{slope/rough}$, r_{succ}, r_{fail}, r_{coll}, and r_{max} are applied only if their activation conditions are satisfied.

4 Experimental Results

In this section the evaluations of our experiments and their results are presented, showing the evolution of our work. We started evaluating the model employing the 3D action space and then moved to the 2D one for simplification and generalization purposes. To asses the performance of our different tests we used a simple metric: the percentage of success for 200 test episodes, i.e. the percentage of the test episodes where the agent was able to land on a safe spot. At the beginning of each training episode of the training and of each test episode of the testing, the agent randomly spawns on the environment, with a restriction in the z coordinate, so that the height from the ground is between 20 and 100 meters in the 3D case, while in the 2D variant it spawns at a fixed height of 20 meters. The yaw of the agent is also randomly chosen, but the roll and pitch are fixed at 0°, so that the down-facing camera is always looking straight down.

4.1 3D action space

At the beginning, some tuning in the configuration and in the reward function is needed to guide the agent towards the ground, or it could keep wondering in the air and reaching the maximum number of steps. This is what happened in our first tests, where the agent was exploring the environment too much and most importantly in a redundant way, as it can been seen in Figure 3. The reward of one of our initial tests is $R = r_{height} + r_{coll} + r_{max} + r_{succ} + r_{fail}$, with $s_{height_1} = 0.01$, $s_{height_2} = 100$, $k_{coll} = -2$, $k_{max} = -2$, $k_{succ} = 1.2$, and $k_{fail} = -0.4$, and with a $stepsize$ of 0.5 meters. The maximum number of steps is 400, and in total the agent is trained for 50,000 training steps.

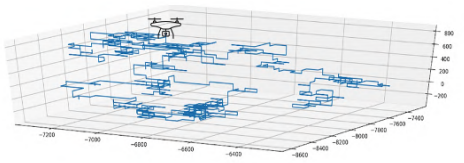
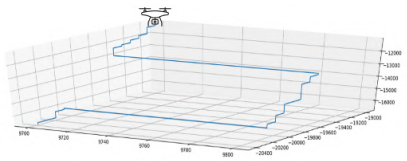


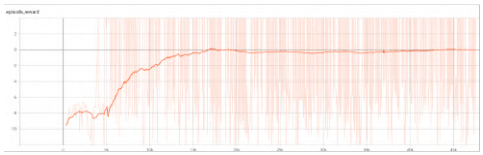
Figure 3: Trajectory of one testing episode reaching the maximum number of steps

The agent is always reaching the maximum number of steps, without really learning how to land (approaching the ground) and how to distinguish safe or not safe spots. Hence, in the following tests we changed how the reward function is constructed, adding more components, and changing the various s and k values. It is added the r_{time} penalty to hurry the drone to land, together with the action penalty r_{act} penalizing up movements and in a lesser degree horizontal movements, to moderate the environment exploration of the agent avoiding redundant displacements. The $r_{slope/rough}$ penalty is also exploited, to try to teach the agent to better infer landable/non landable spots during the flight. After trying some configurations and reward values, our best result was achieved with $s_{height_1} = 0.1$, $s_{height_2} = 25$, $k_{time} = -0.005$, $s_{act} = 50$, $s_{slope} = 0.05$, $s_{rough} = 5$, $k_{coll} = -2$, $k_{max} = -4$, $k_{succ} = 2$, and $k_{fail} = -1$, with a $stepsize$ of 1 meter. The maximum number of steps is 100, and in total the agent is trained for 50,000 training steps.

In Figure 4a is represented a typical successful agent trajectory. The agent has learned to go down as well as reasonably exploring the environment. Additionally, Figure 4b shows the episode reward function keeping track of all the training samples; the moving average of the total reward of the episodes is increasing during the training, stabilizing after 20,000 training steps. However, even if the agent is landing, the percentage of success is only 54%. Thus it is decided to simplify the model, moving to a two-dimensional action space, to focus the agent into learning how to interpret the terrain at a fixed height and how to horizontally explore the environment, without worrying about vertical exploration or all the different states given by the various points of view of the ground at different heights.



(a) Trajectory of one testing episode



(b) Moving average of the total reward of the training episodes

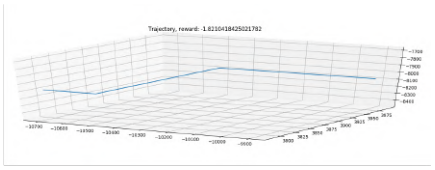
Figure 4: Best test with the 3D action space

4.2 2D action space

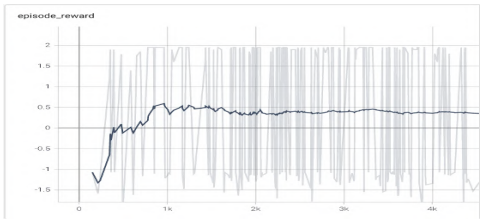
In order to simplify the model, the action space is reduced to the four horizontal actions and one action land. After many simulation tests using PPO2, we empirically deduced CNN policy was providing better results on the simplified model, using an image with size [320,240].

Our most promising implementation is presented in Figure 5; in the reward function, r_{height} and r_{act} are deleted, since the drone is starting at a fixed height and it can move only in horizontal directions. $k_{time} = -0.05$, $s_{slope} = 0.15$, $s_{rough} = 15$, $k_{coll} = -2$, $k_{max} = -4$, $k_{succ} = 2$, and $k_{fail} = -1$. Since the

goal of the agent is to find a safe spot moving in a 2D plane only, without moving for tens of meters vertically to reach it, the maximum number of steps is reduced to 25, with a *stepsize* of 2 meters. In total the agent is trained for 30,000 training steps. The best result is achieved when the agent is spawning at 20 meters from the ground with a reduced field of view of 70°, instead of the default 90°. In Figure 5a is presented a typical trajectory where the agent is horizontally exploring the environment, while Figure 5b shows the moving average of the total reward of the episodes during the training.



(a) Trajectory of one testing episode



(b) Moving average of the total reward of the training episodes

Figure 5: Best test with the 2D action space

During the testing, the success rate is 70%, a noticeable increase from the previous model, which was exploiting the 3D action space. With this simplified model, the reward is stabilizing after 2,000 training steps, fewer than the 20,000 needed for the 3D action space. The simplification has brought an enhancement of the performance, but more tests should be done in the future to try to increase even more this percentage.

5 Conclusion

The objective of this project is to contribute to general frameworks used for developing reinforcement learning agents, specifically for the task of autonomous safe landing. Through this report we have confirmed the feasibility of using Deep-RL based methods for the problem of autonomous landing on rough terrains using only RGB images. However, it is important to note that reinforcement learning framework are an active area of research but stay at development stage. Thus these machine learning methods have not yet uncovered there full potential but it appears as additional motivation for us.

Automate safe landing spot discretization is a very important feature for the whole UAV industry and having performance data on how well this kind of model could work on challenging environment are essential. Even if our final proof of concept model was constructed over a simplified problem, the environment used was a photo-realistic simulation of a rough terrain with a limited palette of colours, demonstrating that the agent is able to infer information from textures that are mostly homogeneous and act correspondingly to pursue its goal.

The achieved results should be a point of departure for future testing, with a particular emphasis on the model with three-dimensional movements, since it represents the actual implementation that should be applied in a real world scenario. Future research could focus not only on a new reward function but also on other reinforcement learning implementations different from PPO. Moreover, and this is for us the main road to follow, an expansion of the state space of the agent (adding states uncorrelated with the RGB data) should be studied. For example, instead of using only RGB images, the state could also be composed of depth images and features, of the current ground distance or the distance from the nearest landable spot.

Acknowledgments

We would like to thank our advisors from the ETH Autonomous System Lab, Olov Andersson, Tim Kazik, and Michael Pantic for their guidance during the different stages of the project. Another thanks goes to Nasib Naimi, whose work in his Semester Thesis [15] was the starting point of our project.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

[3] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[5] H. B. P. d. l. P. . P. C. Alejandro Rodriguez-Ramos, Carlos Sampedro. A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *Journal of Intelligent Robotic Systems*, 2018.

[6] 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. pages 108–114, 2018. doi:10.1109/ICUAS.2018.8453315.

[7] C. C. R. S. I. G. Timo Hinzmann, Thomas Stastny. Free lsd: Prior-free visual landing site detection for autonomous planes. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[8] A. P. N. H. T. E. Y. T. D. S. D. W. Timothy P. Lillicrap, Jonathan J. Hunt. Continuous control with deep reinforcement learning. *Google Deep Mind*.

[9] R. S. Sutton and A. G. Barto. Reinforcement learning: an introduction, second edition. *The MIT Press)*.

[10] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[11] Y. Z. S. Q. Z. X. T. S. K. Y. W. A. Y. Weichao Qiu, Fangwei Zhong. Unrealcv: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, 2017. URL https://unrealcv.org/.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016. URL https://gym.openai.com/.

[13] T. Y. A. Y. Y. W. Fangwei Zhong, Weichao Qiu. Gym-unrealcv: Realistic virtual worlds for visual reinforcement learning. Web Page, 2017. URL https://github.com/unrealcv/gym-unrealcv.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. doi:10.1109/CVPR.2018.00474.

[15] N. A. Naimi. Reinforcement learning for implicit landing site detection and autonomous landing of mavs. *ETH Semester Thesis*, Autumn Term 2020.