



Utrecht University

Project Proposal

Drone Follow-Me Using Reinforcement Learning

Author: Amir PLIEV

E-Mail: amirpliev@gmail.com

Student Nummer: 4123212

1st Supervisor: dr. ir. R.W. POPPE

2nd Supervisor: dr. A.A. SALAH

Degree: ARTIFICIAL INTELLIGENCE

Faculty: GRADUATE SCHOOL OF NATURAL SCIENCES

November 19, 2020

Contents

1	Introduction	2
2	Literature Review	3
2.1	Object Detection	3
2.1.1	Human Detection	4
2.2	Reinforcement Learning	4
2.2.1	Q-Learning	5
2.2.2	Deep Q-Network	5
2.2.3	Reward function	5
2.2.4	Hindsight Experience Replay	6
2.3	Obstacle Detection Technologies	6
2.3.1	Additional hardware	6
2.3.2	Depth Perception	7
2.4	Embedded Device	7
3	Research Questions	8
3.1	Training a Reinforcement Learning algorithm with Hind-sight Experience Replay	8
3.2	Using depth-images as a state-space	8
3.3	Deployment in online phase	8
4	Methods	9
4.1	Goal	9
4.2	Pipeline	9
4.2.1	Object Detection	9
4.2.2	Depth-Perception	10
4.2.3	Reinforcement Learning Agent	10
4.3	Offline Phase	10
4.3.1	Simulation Program	10
4.3.2	Environments	10
4.3.3	HER and State-space	11
4.4	Online Phase	11
4.5	Hardware	11
5	References	12

1 Introduction

The use of drones as a means of automating tasks that are currently being performed manually are becoming ubiquitous [1, 2, 3, 4]. These applications can vary from using drones as a means of package delivery, object tracking and entertainment purposes. Many of these applications require behavior from the device that includes autonomy, which in itself is a very complex defined set of behaviors. Many applications for the drone still require manually controlling of the device in order to perform intelligent behavior. The rise of new implementations of autonomous drones therefore provides the challenge to find new practices in which automating these tasks is possible.

Many of these applications are based on the behavior of tracking an object. The task of keeping a certain object in view of the drone, is a type of behavior that can be used in a variety of practices. The most effective solution would be to have the drone keep an object in its field of vision using a camera. The tracking of an object has recently been applied to follow a powerline in order to perform maintenance [4]. This implementation raises the question whether this would also be possible for the tracking of persons in a variety of environments. Already established practices have been the tracking of athletes while filming [5] and guiding blind people in the streets [3]. In these environments, the drones are able to accurately track and maintain their distance from an object, while simultaneously keeping the object in sight.

However, some applications require smaller devices and therefore, lower computational models in order to achieve the same type of behavior. As an example, the purpose of a drone camera that can automatically take pictures and/or record an individual in indoor environments. Smaller devices have computational limitations as well as physical limitations that need to be taken into account. These devices, referred to as Embedded Devices (ED), can be defined as having been produced with a specific function in mind [6]. Per definition,

these devices can be categorized as everything between a coffeemaker and mobile phones. In most cases, these EDs lack GPU power, which limits them from performing heavy computations. Therefore, when developing models for these devices, it is necessary to sacrifice model accuracy for overall speed of inference. At the same time, certain technologies used in obstacle detection become problematic since these could outweigh the carrying capacity of the drone. Therefore, many of the techniques used in developing the follow-me drones, can not be used in indoor applications. The need for the application of these technologies on EDs is therefore still relevant.

Recently, the rise in using Deep Neural Networks applied to robotics could show promising possibilities for these problems. These neural networks are used in the context of Machine Learning [7]. A specific way in which these can be applied, is in a Reinforcement Learning (RL) application. This method is most useful in applications where there is a necessity to map actions to certain environmental states and the annotation of these states is too large to perform manually. These RL agents thrive in decision making problems where the agent is able to explore the environment itself and correct itself according to a pre-determined reward function. Applying this to the use of indoor follow-me drones would be an interesting challenge. Furthermore, a pitfall is that training in real-time is an unreasonable endeavour since RL requires a vast amount of training instances for the learning process. Therefore, simulation environments will be considered as a possibility for the algorithm to learn in an offline environment before being deployed in the real-world. Optimizing the process of training and deployment of an agent to perform follow-me behavior on a drone is therefore an active field of research.

2 Literature Review

This section will investigate the current state of the scientific knowledge about the relevant topics of this thesis. The first section will explain what the current state of object detection on embedded devices is. This is then followed by discussing how the problem of controlling the drone using RL has been solved in earlier studies and how this thesis will aim at optimizing this process. Finally, as has been mentioned before, the RL algorithm will be trained in a simulation environment. Deploying such an agent to the real-world brings with it a set of problems that will be addressed.

2.1 Object Detection

The primary problem to solve is to make sure that the drone is able to detect the required object to follow. This problem is most effectively solved by the use of neural network models that have been specifically trained to detect a vast variety of objects. The attempts at solving the object detection problem using neural networks has already been a topic of interest for more than a decade [8]. The first largest breakthrough in this area has been the application of a combination of networks [8]. The first part of this combination is a Convolutional Neural Network (CNN), which is a model that performs object classification by extracting low-level spatial features from the images and combining these into high-level features before classifying the image into a set of classes. The second part is a region proposal network which propose regions where objects could be present. By having the first part of the network perform region proposals, and supplying these to a CNN, the R-CNN network was significantly able to increase speed and accuracy in object detection models. Nonetheless, this network was soon improved by performing the input to the CNN in one forward sweep for all the different proposed regions in the previous step [9]. Finally, combining the whole process, provided the final step in accurate and reliable object detection [10].

However, these improvements, although

vastly progressing the speed and accuracy of the models, have still left room for networks that might sacrifice a proportion of accuracy for the sake of speed. The most notable model that has been able to do this, is called You Only Look Once (YOLO) [11]. The idea is to only perform a single pass through a network per image, and get the required predictions. This is done by first dividing the image into an $S \times S$ grid. For each of these grid cells, the model predicts N amount of bounding boxes and corresponding confidence scores. These bounding boxes are possible rectangles that enclose an object. Then, for each of these bounding boxes, a probability distribution over the possible classes is performed. From this set of bounding boxes with corresponding class probabilities, the bounding boxes with the highest confidences are preserved, leaving a set of bounding boxes and their corresponding class predictions for each object behind. Performing object detection using this technique vastly improves the speed while only minimally sacrificing accuracy. Additionally, through the use of neural network optimization techniques, this model has seen drastic improvements in later years as new versions of it have been introduced [12, 13, 14]. These YOLO algorithms deliver much higher speeds while minimally sacrificing accuracies. Situations such as embedded devices could use these aspects of these object detectors.

However, alongside with the increase in research into these YOLO algorithms, there has also been research into developing these models for low computational devices [15, 16, 17]. The method that has been used in these to reduce the computational load have been either pruning methods, where parameters least crucial to the final output are discarded, or by creating a similar network architecture with less parameters and performing the training process anew. Nonetheless, they still sacrifice a significant proportion of accuracy for the sake of speed on EDs.

2.1.1 Human Detection

As a subset of Object Detection, human detection has been a field that has benefited greatly from the advances in Deep Learning (DL). From the first large breakthroughs of the field [18, 19] the contribution of DL is still not outperformed. Problems such as trouble finding humans in different poses and unreliable bounding boxes make the alternative much more attractive. Using DL networks for the object detection problem has been able to perform the task without these problems. This leaves the use of the previously described object detectors, as the best option in order to perform overall human detection in most cases where the previously discussed issues are vital.

2.2 Reinforcement Learning

The method of performing obstacle avoidance and following in this thesis is to use Reinforcement Learning (RL). RL is a type of Machine Learning that focuses on mapping situations to actions according to the maximization of a numerical reward function [20]. The main process of learning here is to adapt a decision maker, which we will call an agent, according to experiences [21]. Here an attempt is made to perform the action, after which a reward is given in the form of a scalar value by the environment, which are all the externalities that determine what state the agent is in. This reward is then used as a cue to alter the mapping from states to actions accordingly. This process allows the agent to learn from experience on how to act in order to achieve the desired behavior. The foundation of RL is based on the collaboration between the agent and the environment. For each time step, the agent finds itself in a certain state. During this state, the agent has a certain set of actions that it can choose from. For each of these actions, there is an effect on the environment. Therefore, the agent finds itself in a new state, repeating the process of action-selection. This process is illustrated in Figure 1.

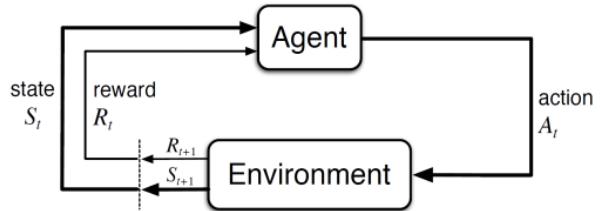


Figure 1: Agent-Environment interaction in RL

The essence of this algorithm is based on the environment's response to each action that the agent chooses. The environment, during training time, gives the agent either a reward (possibly in the form of punishment) that the agent uses to gather experiences. This reward is based on a function which determines what the agent will try to optimize and therefore by definition, what the agent will try to learn.

Formally, this process takes an arbitrarily defined timestep t , and provides the agent with a state S_t . Given this state S_t , the agent decides on an action a out of the possible available actions A using a certain policy π . Performing this action changes the environment and provides the agent with a new state: S_{t+1} . Together with this new state, the environment also provides the agent with a reward r_{t+1} . The eventual learning goal of the agent, is to find a policy $a = \pi(s)$ that maximizes the overall reward that he will receive. This requires taking into account future rewards as well, which is recorded in the return R_t of a certain action in a specific state. However, what actions will be chosen by the agent is dependent on the policy π that the agent employs. The policy is defined as a probability distribution over the possible actions for each state. The goal is therefore to maximize its return by defining a certain policy.

An important concept here is the balance between exploration and exploitation. When the policy is completely directed towards exploitation, the agent will not explore many of the other possible actions that can be taken that could potentially lead to higher returns. However, if an agent is fully committed to exploration, it will never maximize the returns and will keep exploring the space without settling on any maxima. The policy which perform the best are therefore, the ones that are

able to balance both of these strategies in order to maximize the overall returns.

2.2.1 Q-Learning

There are multiple ways in which to approach the learning problem in RL. One of these is to focus on the determination of the Q -values. These values connect the action with the states and ascribe to them a value. This value represents the expected return for choosing an action a in state s , expressed as $Q(s, a) = \mathbb{E}_\pi [R_t | s_t = s, a_t = a]$. Intuitively, the expected return is the total sum of expected reward that will be collected by choosing this action and continuing from there onwards. This leads to the following simplified equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \quad (1)$$

The function of this equation is, therefore, to update the Q -values throughout the learning process. Since these values represent the overall return of choosing an action, this will create a mapping from each state to what the value is of each action in a state, taking into account later states. Having this information provides the algorithm with an overview of what action to take in each situation by choosing the action with the highest Q -value for that specific state.

2.2.2 Deep Q-Network

In the domain of this thesis the actions that the RL algorithm can choose from are a discrete set that consist of directions that the drone can go towards. The states, however, are a continuous space that are represented by the bounding box of its target on the input image and an additional image, either depth- or normal-image.

Keeping a tabular representation of the mapping of each state to its possible actions would vastly inflate the amount of information that the algorithm would need to keep track of. This means that there is a necessity to represent the action space, not categorically, but as a function.

This is where the previously described neural network approach comes into play. Deep

CNNs are used in order to do the mapping of a continuous state to a discrete action space [22]. The DQN chooses its actions using a ϵ -greedy policy, which is to say the action with the highest Q -value is chosen, except for a ϵ proportion of the time where another action is chosen randomly.

For each of these timesteps the DQN stores the experience, which is a sequence of $(S_t, A_t, R_{t+1}, S_{t+1})$, to a replay memory buffer. This buffer tracks a million of the most recent experiences of the agent. Using this buffer, a random sample of experiences are taken and fed forward through the CNN, adjusting its mapping from states to actions. This has been developed for smoother learning as the agent refers back to a larger memory of experiences throughout the learning process and is not as prone to variations in parameters.

2.2.3 Reward function

It becomes apparent that the most crucial aspect of what the agent will learn is mostly dependent on the reward function that determines what reward the RL algorithm will receive throughout each time-step. The definition of a reward function therefore becomes crucial in the process of training an agent with a behavioral goal in mind. The engineering of a reward function can therefore be approached differently.

The easiest way in which to do this is to simply mark the states where the goal has been achieved and mark the remainder of the states as being failed. This formalizes what states the agent will try to achieve very clearly. However, this also brings with it some complications during the learning process as this means that the agent will be exploring a vast amount of state-space where no reward signal is given.

On the other hand, the state-space can also be given a certain type of reward function that reflects an ordering of the states in a way that represents how close an agent is to the goal state. However, the disadvantages here are that this requires domain-specific knowledge about when an agent is closer to the goal or not. This imbuing of some predetermined knowledge in

the reward function also blocks the agent from finding new ways to solve the problem. Performing this reward function engineering is also a limitation in situations where the composition of permissible behavior is not known prior to training [23].

This makes the alternative of having a delimited goal space much more attractive, even when considering the sparse rewards that it might give.

2.2.4 Hindsight Experience Replay

Domains with sparse rewards pose some challenges in the training process. Since many of the states possess rewards that are negative, there are not many instances of experiences where the agent receives a reward signal that is significant for it to relate what it would need to do to maximize this reward signal. In the domain of the drone operating in the simulation, only in the camera shots where the drone is actually including a person, is there a positive reward. It therefore becomes extremely difficult for the RL algorithm to learn in this environment, because the instances where it does receive a reward are very sparse.

A large breakthrough in training RL agents in sparse reward environments has been the concept of Hindsight Experience Replay (HER). Closely related to the experience buffer that has been mentioned, this method uses previous experiences that have not performed the task successfully, and reshapes them into positive experiences. By taking a sample from the experience replay buffer, and changing the received reward for that experience to a positive reward, as if the goal had been reached in that state, the experience can be used as though it had been a successful attempt at performing the task. This simple method, has provided many robotic applications with a faster learning curve and better performance [24].

However, the application of HER in the domain of a drone operating in a virtual environment has not been performed before. Many of the applications have included simulated environments where a robotics task is being performed. The behaviors that HER significantly

improved have been tasks where a sequential approach to solving the problem was necessary. In the case of the drone, there is a similar requirement where the drone would need to perform a sequence of tasks in order for the target person to be visible in its view. The application of the HER method could lend itself very well to the domain of the drone following task.

2.3 Obstacle Detection Technologies

Next to the target to be followed, another problem is the obstacle avoidance issue. The problem here is that the agent requires some type of information in its input to be able to deduce whether an obstacle is in its way. Therefore, the primary problem to solve is what sensory information the agent has access to, to be able to determine whether obstacles are present. Many attempts have implemented deep neural networks to perform the object detection, and included additional hardware to perform the obstacle detection [25, 26, 27]. More remarkably, there is an increasing interest in performing the obstacle avoidance task using only computer vision techniques. Both of these will be explored.

2.3.1 Additional hardware

Many solutions to the problem of obstacle avoidance have been to install a certain type of sensor technology in order to detect obstacles. Technologies such as LiDAR [28], for example, are being used in many devices that require the ability to potentially avoid objects in order to follow a certain path. A most notable example of this application has been in self-driving cars [26]. LiDAR works by emitting strobes of light at the scene and recording the return time of these strobes. This can be used to sense the distance from the objects in around the sensor. Next to this, the use of acoustic signals as a way of detecting obstacles has been used in the case of flying unmanned drones [27]. This is similar to the LiDAR technology but instead uses sonar waves to calculate the distances of the objects around the object. However, a considerable disadvantage of these types of sensors are that both of them add a physical weight to

the device which impedes the movement of the drone, posing a limitation to the use of these techniques. The preference would still go towards a method that would omit the need for extra hardware.

2.3.2 Depth Perception

At the same time, another technology to use for obstacle avoidance is to solely rely on the camera to perform the task. This allows the system to detect and avoid possible obstacles in its path without the need for extra hardware to perform this detection. When it comes to the limited resources of an ED, the preference goes to these methods.

One of these vision-based methods for obstacles detection is depth-perception. The use of depth-perception greatly reduces the necessity of the drone to recognize other objects next to the target object. This technology works by using two cameras that are setup to take an image simultaneously. By finding the pixel that corresponds to the pixel in the other image taken, the disparity between both of these pixels can be used to calculate the distance of the two cameras to that point.

Similarly to humans, this would allow the drone to recognize that something is in its way, without the necessity to recognize what it is that is obstructing its path. Using a depth perception image as an input for any decision making process could vastly simplify the problem of obstacle avoidance all together. Using depth-images as an input to a RL algorithm has been performed before [29], however, whether this improved the performance compared to direct camera input or whether this translated to real-world applications is still missing. Testing whether the depth-perception adds an easier training process and whether it allows for generalization to real-world applications is therefore still relevant and a interesting option to use as an input to a RL agent.

Many of the depth-imaging technologies face difficulties performing accurate depth perception in various circumstances. This is especially the case when the required distance becomes larger than a couple of meters [30].

However, since the application of this thesis is to let the drone perform indoor environments, this limitation becomes less relevant, especially considering that stereo vision is capable of performing depth-imaging within 2 meters with a maximum error margin of 10%.

2.4 Embedded Device

This thesis will focus on the implementation of these algorithms on an embedded devices (EDs). These devices are per definition poor in computational resources because they are developed with a specific functionality in mind [6]. Therefore, the aforementioned problems of physical and computational limitations are . Indoor drones are modest in size and consequently, also in computing power. There are certain approaches that can be used in order to circumvent or solve this problem of lower computation. These include cloud computing or a ground controller [31, 32]. However, these also pose limitations to the flying range, capabilities and add latency that only increase the list of constraints to the computationally limited drone.

The focus in this thesis is therefore to keep the necessary algorithms computationally low. This would come with additional efforts into creating smaller or computationally lighter algorithms without drastic loss in functionality. The previously mentioned pruning techniques that the object detectors have used to develop lighter and faster versions are relevant for devices such as these. The reduction in computational power that these methods require leaves more room for the DQN.

3 Research Questions

The main focus of this thesis will be to implement, optimize and evaluate a system that can control a physical drone in order to track and follow a person. The goal here is to propose a pipeline that will be implemented on an ED, containing low computational power as a constraint. The pipeline looks as follows. First, the system will contain an object detector that can specifically locate a person in real-time. Next to this, the option of having a depth-image created and fed to the RL agent, together with the coordinates of the located person, will be available. Using this information, the system then needs to decide on what action to take in relation to the position of the person and possible obstacles. This will be implemented in two phases. The system will be implemented and trained in a virtual environment in an offline phase. Afterwards it will be deployed on a physical drone to observe whether the optimized learning process translates to a new domain. In order to formalize this process, the following research questions have been defined.

3.1 Training a Reinforcement Learning algorithm with Hindsight Experience Replay

The RL agent being implemented in a domain with sparse rewards means that the need for some optimization techniques are required. Learning in these environments is harder but fortunately, this problem has seen a recent breakthrough, which has been the aforementioned Hindsight Experience Replay (HER). This algorithm allows the agent to also learn from experiences where the task was not performed successfully. The results would hopefully show that the implementation of this method during learning will speed up the learning process and improve overall performance. The goal here is to compare the learning process of an RL agent that has implemented the HER method, with one that did not.

1 – Does Hindsight Experience Replay improve the training time of

the RL agent for the task of following a person?

3.2 Using depth-images as a state-space

The desired behavior would preferably not only be to follow a person, but to also successfully identify and avoid objects that are in the way. The biggest problem is that the RL agent would require some type of input that could relay that information about possible obstacles to the agent. The use of depth-imaging has been used to in this application, however whether this improves the performance of the agent or reduces training time is unclear. This thesis will investigate whether using depth-imaging increases the average reward of the agent compared to the use of direct image input for the RL agent.

2 – Does the use of depth-images significantly improve the performance of the RL agent?

3.3 Deployment in online phase

The goal is to have this system deployed on a physical drone that can perform the same task. The effects of the optimization techniques that have been implemented in the earlier questions on the simulated environment would also be expected to increase their performance when deployed on the physical drone. Whether the optimizations techniques can be said to have translated to the physical drone is to be investigated.

3 – Does the model with the proposed optimizations perform similarly on a physical drone?

4 Methods

This section will explain the overall architectural pipeline that will be used in order to implement the required models and agents to perform the following behavior, which is illustrated in Figure 2. This pipeline will be implemented in two phases: an offline and online phase. Both of these will be discussed. Finally the hardware used in this thesis will be explained.

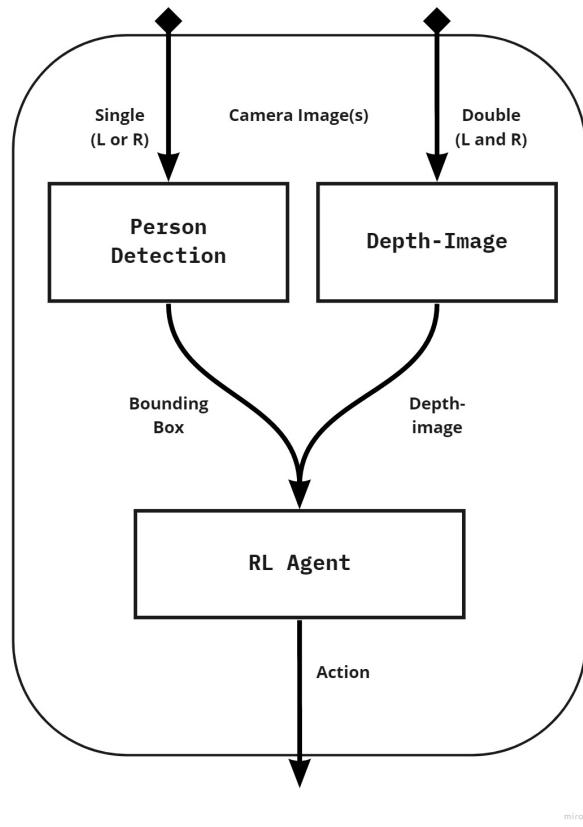


Figure 2: The pipeline to be implemented

4.1 Goal

The desired behavior for the drone is to follow the person. For this thesis, this can be interpreted in a wide sense of the word: as long as the person is in the center vision of the camera of the drone, this task is being fulfilled. Translated to the inputs that the drone will receive, this means that the drone should maintain a bounding box as much in the center of its image input as possible. This can easily be calculated by simply taking the middle of the bounding

box produced by the object detector. Bringing this point as close as possible to the center point of its image, is what the RL agent would need to learn. Next to defining the problem statement, this simultaneously removes the need to specify what distance the drone would need to fly from because the system can decide on its own what the perfect angle and distance will be to perform this task.

4.2 Pipeline

To perform this successfully, the pipeline, as illustrated in Figure 2, will be implemented. This pipeline includes an object detector, depth-perception technology and a RL agent, each of which will be discussed further.

4.2.1 Object Detection

Longduring efforts to reduce the computational weight of object detection models has resulted in a set of networks that can operate in some capacity on the RP [17, 16, 15]. However, since the rest of the pipeline also needs to be implemented on the RP, the need for the lightest of these models is required in this thesis. For this reason, YOLO-Fastest [15] will be used. This model only takes up a total of 1.3MB in size and has a 4.4ms runtime, which is the fastest of the slimmed YOLO versions. The reported mAP on the VOC dataset is 23.65% for the smallest version. The latest YOLOv4 [14] has a mAP of over 43.5% on the VOC dataset. This does not compare directly, but this does give a good indication for the possible results.

Since the models are built to distinguish a multitude of classes and this thesis only requires one class, a person, the model would need slight fine-tuning. This can be done by performing transfer learning. By freezing all the layers except for the last classification layer, and retraining this layer, the desired class detection will be achieved. This process will result in a smaller network leading to faster inference, while not losing in performance.

4.2.2 Depth-Perception

The next aspect of the pipeline will be the depth-perception. Since this thesis consists of two stages, an offline and online phase, there is a distinction in how the depth-imaging is performed. In the offline phase, which will take place in a simulated environment, the depth-image is being provided by the simulation itself. Since this simulation has full access to the ground truth, an accurate depth-view can be produced. However, during the online phase, this is not the case. Here the choice of technology has been to implement stereo vision. This means that there will be two images from two cameras, resulting in a left and right image. These two images will have to be processed into a depth-image which will be performed using the OpenCV library, a library which is specialized in performing computer vision operations. This will produce a depth-image with a slight margin of error, however. Whether the trained network in the offline phase can generalize to the real-world will be investigated in the online phase.

4.2.3 Reinforcement Learning Agent

After object detection, the next aspect is to implement an RL agent for the use of determining what actions to take in each situation. The training stage of this agent will happen using a virtual environment. The ability to command a drone in a virtual environment will be used to model the situation as it would be on the physical drone. This simulation drone will receive the same camera inputs as the physical drone, which will be an image of the camera on the drone and a corresponding depth image. The normal image will be used for performing the object detection. The depth-image will be used as an input for the RL agent, together with the bounding box.

4.3 Offline Phase

The deployment of the pipeline will happen in two stages. The first stage, the offline stage, will be training the RL agent in a virtual environment. In order to do this, the full pipeline

must be implemented in a simulation. This section will elaborate on the choice of simulation program, environments used for training and delineate the training process.

4.3.1 Simulation Program

The program used to implement and train the agent will be Microsoft's AirSim [33]. Based on the Unreal Engine [34], the simulation allows for a multitude of created environments to be used. Next to this, the program allows the user to instantiate a quadcopter directly in a virtual environment, together with simulated vision possibilities including a normal camera and depth-view, making this the preferred option. These views can be observed at the bottom of Figure 3. Additionally, the ability to command the vehicles through a Python or C++ script, makes AirSim very suitable to perform Deep Learning. What makes it even more attractive, is that AirSim allows easy connectivity with the PixHaw API, which, as will be explained later, is also used to control the physical drone. Once the agent has been trained, this makes the transfer to the physical drone much easier.

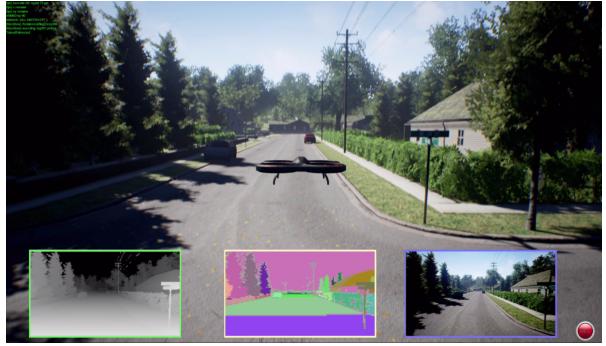


Figure 3: The AirSim program

4.3.2 Environments

As has been mentioned before, the ability to use pre-built environment for the deployment of the drone in the simulation exists. Since the application of deep learning to quadcopter command has been executed before [35], the same environments can be used in this thesis. These environments have been developed by the researchers with the goal of training the drone in

a navigation task in mind. Examples of these environments can be seen in Figure 4. They contain hallways and indoor furniture in order to make sure that the drone can train in environments that are representative for the task. This pack contains a total of eleven environments, where a set of factors have been randomized (e.g. wallpaper pattern) for generalisability purposes.

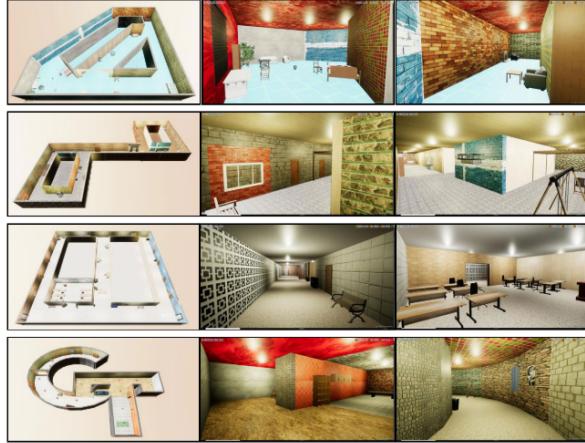


Figure 4: Examples of environments used during offline phase

4.3.3 HER and State-space

In order to measure the first research question (3.1), whether the addition of HER to the training process reduces the training time, both processes will be performed and evaluated separately. For the second research question (3.2), the comparison between normal image input and depth-image as an input for the RL is tested. This will happen by using the faster of the two options (HER or no HER) and performing training for both options will answer this question.

4.4 Online Phase

Finally, the complete trained pipeline will be implemented on the ED and on the physical drone. In this online phase, the better performing implementation from the previous processes will be tested to see whether the transfer to the physical drone translates to similar behavior.

4.5 Hardware

The devices that comprises the drone is split into a couple of subsections. The primary aspect is the actual quadcopter that keeps the device in the air and can take actions. Next to this, the device that actually carries the computational load of the desired autonomous behavior is attached separately.

The drone used in this thesis is the RDDRONE-FMUK66 model, as seen in Figure 5. This device is a lightweight version of a drone that needs to be assembled component by component making it easily modifiable. Important to note, is the fact that the drone comes with the availability of controlling the drone using a high-level API called PixHawk. This can be used to control the drone from different devices with commands such as "Up" and "Forward" without going into the details of determining how to instruct the rotors separately. This makes the output of the RL agent easily defined and transferred to the physical drone.



Figure 5: The RDDRONE-FMUK66 drone

The ED used in this thesis is the Raspberry Pi (RP) 4B with 4GB of RAM. The RP has the advantage of being physically small but without adding the problem of completely designing an embedded device for this single purpose task.

The technology performing the depth-perception that has been chosen is a form of stereo vision. For the RP, there is already a starter kit that includes all the necessary hardware and software requirement to create a depth image. This kit, the StereoPi [36], includes two cameras and additional libraries to help with easy connection to the RP. Further-

more, the StereoPi includes a module that can additionally perform some computations. This helps remove some load from the RP so that the focus can be on performing the object detection and path planning. The images that this camera will produce, will first be used to calculate a depth-image, which will then be used as a potential input to the RL agent.

5 References

- [1] A. Claesson, D. Fredman, L. Svensson, M. Ringh, J. Hollenberg, P. Nordberg, M. Rosenqvist, T. Djärv, S. Österberg, J. Lennartsson, *et al.*, “Unmanned aerial vehicles (drones) in out-of-hospital-cardiac-arrest,” *Scandinavian journal of trauma, resuscitation and emergency medicine*, vol. 24, no. 1, p. 124, 2016.
- [2] W. Krüll, R. Tobera, I. Willms, H. Essen, and N. von Wahl, “Early forest fire detection and verification using optical smoke, gas and microwave sensors,” *Procedia Engineering*, vol. 45, pp. 584–594, 2012.
- [3] M. Al Zayer, S. Tregillus, J. Bhandari, D. Feil-Seifer, and E. Folmer, “Exploring the use of a drone to guide blind runners,” in *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, pp. 263–264, 2016.
- [4] A. Muhammad, A. Shahpurwala, S. Mukhopadhyay, and A. H. El-Hag, “Autonomous drone-based powerline insulator inspection via deep learning,” in *Iberian Robotics conference*, pp. 52–62, Springer, 2019.
- [5] V. Iastrebov, C. Y. Wong, W. C. Pang, and G. Seet, “Motion tracking drone for extreme sports filming,” 2014.
- [6] “What is an embedded system? definition and faqs.” <https://www.omnisci.com/technical-glossary/embedded-systems>.
- [7] L. Hardesty, “Explained: Neural networks.” <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>, Apr 2017.
- [8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” pp. 580–587, IEEE, 2014.
- [9] R. B. Girshick, “Fast R-CNN,” *CoRR*, vol. abs/1504.08083, 2015.
- [10] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.
- [11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.
- [12] J. Redmon and A. Farhadi, “Yolo9000: better, faster, stronger,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.
- [13] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.
- [14] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” *arXiv preprint arXiv:2004.10934*, 2020.
- [15] R. Huang, J. Pedoeem, and C. Chen, “Yolo-lite: a real-time object detection algorithm optimized for non-gpu computers,” in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510, IEEE, 2018.
- [16] H. Zhao, Y. Zhou, L. Zhang, Y. Peng, X. Hu, H. Peng, and X. Cai, “Mixed yolov3-lite: A lightweight real-time object detection method,” *Sensors*, vol. 20, no. 7, p. 1861, 2020.

- [17] P. Adarsh, P. Rathi, and M. Kumar, “Yolo v3-tiny: Object detection and recognition using one stage improved model,” in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 687–694, 2020.
- [18] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1, pp. I–I, IEEE, 2001.
- [19] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1, pp. 886–893 vol. 1, 2005.
- [20] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2018.
- [21] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” *arXiv preprint arXiv:1710.02298*, 2017.
- [22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [23] A. D. Laud, “Theory and application of reward shaping in reinforcement learning,” tech. rep., 2004.
- [24] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” *Advances in neural information processing systems*, vol. 30, pp. 5048–5058, 2017.
- [25] T. T. Do and H. Ahn, “Visual-gps combined ‘follow-me’tracking for selfie drones,” *Advanced Robotics*, vol. 32, no. 19, pp. 1047–1060, 2018.
- [26] J. Hecht, “Lidar for self-driving cars,” *Optics and photonics news*, vol. 29, no. 1, p. 26, 2018.
- [27] T. T. Mac, C. Copot, R. De Keyser, and C. M. Ionescu, “The development of an autonomous navigation system with optimal control of an uav in partly unknown indoor environment,” *Mechatronics*, vol. 49, pp. 187–196, 2018.
- [28] A. Devos, E. Ebeid, and P. Manoonpong, “Development of autonomous drones for adaptive obstacle avoidance in real world environments,” in *2018 21st Euromicro Conference on Digital System Design (DSD)*, pp. 707–710, IEEE, 2018.
- [29] K. Ko, “Visual object tracking for uavs using deep reinforcement learning,” 2020.
- [30] A. t. A. R. 3D, “Comparing three prevalent 3d imaging technologies.” <https://www.revopoint3d.com/comparing-three-prevalent-3d-imaging-technologies-tof-structured-light-and-binocular-stereo-vision/>, Nov 2019.
- [31] A. Koubâa and B. Qureshi, “Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet,” *IEEE Access*, vol. 6, pp. 13810–13824, 2018.
- [32] R. Bartak and A. Vykovský, “Any object tracking and following by a flying drone,” in *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, pp. 35–41, IEEE, 2015.
- [33] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in *Field and Service Robotics*, 2017.
- [34] “Unreal engine: The most powerful real-time 3d creation platform.” <https://www.unrealengine.com/en-US/>.

- [35] A. Anwar and A. Raychowdhury, “Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning,” *IEEE Access*, vol. 8, pp. 26549–26560, 2020.
- [36] “Stereopi.” <https://stereopi.com/#home>, 2015.