Collision-free indoor flight in a simulated environment using vision-based deep reinforcement learning.

Luís Filipe Pinto Bernardo
luis.bernardo@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2018

Abstract

Considering UAV popularity increase in recent years and their extensive range of possible applications, it makes perfect sense to investigate new technologies for autonomous flight. Indoor navigation is a challenging problem in robotics, where the premise is different from outdoors (GPS signal quality does not generally allow precision flight indoors). Artificial intelligence is one of the many tools used to create and optimize solutions and, in this thesis, a deep reinforcement learning based solution is presented for collision-free indoor flight. Training a model in a real-life scenario can be dangerous since it is a trial-and-error approach and would definitely lead the drone to crash. To overcome this situation, we explored a deep reinforcement learning method in a simulated environment. Keeping in mind that for a real application the model must be transferred to a real drone, we developed a simple system, which requires only a depth and an RGB camera, and that could easily be deployed in the real world. In our approach we used the deep Q-learning algorithm to train a set of models with different types of deep convolutional neural networks, studying the performance on different parameters. The training was performed in a series of virtual environments that got gradually more complex as randomization on texture and structures is introduced into the scene, with the objective of training a generalizable model. We evaluated all trained models in an unknown environment, showing how model generalization plays a crucial role in improving the model behavior.

Keywords: UAV, Indoor Navigation, Reinforcement Learning, Deep Q-learning, Simulation

1. Introduction

Unmanned Aerial Vehicles (UAV), more commonly referred to as drones, are aerial vehicles that can be either directly controlled by a human pilot, through a ground control system, or fly autonomously. Small UAV and Micro Aerial Vehicles (MAV) have a low maintenance cost, are easily deployed and have high mobility, which might explain why we have seen an increase in their popularity. They are convenient and versatile, making the amount of possible applications high, ranging from military to commercial and civilian uses. These drones can perform real-time monitoring of forest fires, road traffic, air quality, and more. They can be used for surveillance, rescue missions, terrain mapping and recognition, deliveries, infrastructure inspections, filming, and entertainment among other ends.

Many of the mentioned applications rely on the UAV ability to fly autonomously. An autonomous system does not get tired, it is more precise and it is capable of performing more complex jobs. However, this ability implies that the drone must be able to navigate different environments by itself. This happens both indoor and outdoor, while the drone avoids obstacles and follows objective oriented guidelines. Creating a reliable autonomous UAV system, capable of safely navigating an environment, is still a challenging problem in robotics.

Traditional methods consider the navigation problem to be divided into two main stages: localization and control [1]. Localization is related to knowledge of the position of the drone and of the mapping of the environment. Control concerns the generation of commands that fly the vehicle, based on the environment’s map.

1.1. Motivation

For autonomous outdoor flight, many techniques that use GPS or other sensors have been developed [2]. These approaches, however, cannot be deployed indoors, where most of the time the access to GPS signal is denied or too weak. To overcome this problem, indoor flight methods have been studied and applied over the years. The control of these systems is important and deep reinforcement learning proves a promising technique to achieve autonomously complex control policies for drone
Applying RL methods to real world problems is, nevertheless, a complicated challenge. For autonomous flight it is required that the system is both reliable and safe. One of the main obstacles concerns training. RL training follows a trial-and-error format, meaning that using real drones during training may not only be unsafe but also expensive (since the drone will definitely crash). Although some workarounds exist, such as making the drone resistant to collision, one would immediately face another problem regarding the amount of human effort needed. Constant supervision would be required to ensure that training happened smoothly over time. Also, deep learning techniques require great amounts of data which also has a big impact in human effort.

Some solutions have been proposed to solve the mentioned problems regarding deep learning techniques. Researchers have managed to make a drone learn to fly through supervised learning by using a manned wheeled vehicle dataset, skipping the need to build their own data [1]. Another existing solution found in the literature is to perform the training in simulated environments [3],[4], which avoids the dangers of training a model in a real life scenario.

1.2. State of the Art
During a drone flight, localization can be achieved through perception and estimation of the underlying map. A reliable approach that does this is the SLAM framework, which performs simultaneous localization and mapping of the environment. The use of a decentralized and visual SLAM [5] is the current state-of-the-art regarding indoor navigation. Nonetheless, SLAM approaches still have some downsides, such as the intensive processing power they might need [2], and errors introduced in the perception system because of visual effects (visual aliasing, dynamic scenes or reflections) [1].

In the work proposed by Loquercio et al. [1], a supervised learning approach was adopted to train a neural network, denominated DroNet. The network is capable of guiding a drone safely in the streets of a city through its outputs: steering angle and probability of collision. The outputs are trained with two different databases. The steering angle prediction is learned from a database that comprises hours of videos captured from manned vehicles being driven in the streets, such as cars and motorbikes. For collision probability the authors built a database of videos captured from a bicycle approaching different obstacles. Even though the main objective of DroNet is outdoor navigation, the network was able to generalize for indoor scenarios to a certain degree.

Sadeghi and Levine[4] published a work that tackles indoor flight of a UAV with a deep reinforcement learning approach. The drone’s network predicts motor commands from monocular images and it was trained in a simulated environment, to avoid some of the mentioned problems inherent to reinforcement learning approaches. This project also focused on the problem of domain adaptation from a simulated environment to the real world. The solution found was to train a generalized model. The simulation environment is diverse in map structure and textures, furniture locations and lighting are all randomized. Even though the simulation is not realistic in terms of graphics, it proved capable and the model learned was able to generalize and guide a drone in a real world scenario. The network is based on a VGG16 structure and its output predicts the Q-function associated with the RL problem. With the Q-function learned, the drone only needs to choose the action that outputs the highest Q value for the current observation.

1.3. Objectives
Taking inspiration from Sadeghi and Levine’s work [4], described in the state-of-the-art, we purpose to apply a deep reinforcement. The main focus is collision-free flight, meaning there is no end goal objective such as finding an object or navigating the entirety of the environment. As a difference from Sadeghi and Levine’s work, the proposed system has a simpler reward function that uses only on-board sensors information. This way a model can be easily fine-tuned using real-world data. We also intend to explore how certain hyperparameters influence training.

Another focus of the project is to understand how different types of randomization improve models’ generalization capabilities in an unseen environment.

Contributions
- A network that can learn to fly a drone in unknown simulated environments.
- An analysis on how different levels of randomization on two domains (textures and map structure) affect the generalization capabilities of the trained models.
- A simple reward system that can be deployed to a real life scenario with minimal equipment (RGB and Depth camera).

2. Background
Reinforcement learning (RL) is a type of machine learning were an agent learns by interacting with an environment. As the agent performs actions that change the environment, it receives rewards ($R$). RL methods try to understand which actions lead
to higher rewards and to maximize the total amount of reward (return) the agent receives. This can be done, for example, by approximating the action-value function (or Q-function) of the RL problem, which predicts the total return from a state $S$, when the agent performs a certain action $A$.

2.1. Deep Q-Learning

This algorithm was introduced by DeepMind [6] and is a variant of Q-learning, which combines RL with deep neural networks. It’s "the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning" [6].

Reinforcement learning methods that use nonlinear function approximators to represent the action value function $Q$ are known to be unstable. Deep Q-learning uses a deep convolution network to approximate the Q function (Q-network) and, to address the instability issues, it introduces two changes to its predecessor, Q-learning. Firstly, an experience replay that randomizes the data collected during training is used. At each time step $t$ the algorithm stores the agent experience $e_t = (s_t, a_t, r_t, s_{t+1})$ in the memory ($s_t$, $a_t$, $r_t$, $s_{t+1}$ are the values assumed for $S$, $A$, and $R$, at a certain time step $t$ or $t+1$). During training, a batch of experiences is sampled uniformly from the memory and used to update the Q values [6], reducing the correlation between training samples. Besides the replay memory, a change related to how the update of the Q-network is done can be introduced [7]. The algorithm combines the update of two Q-networks. A network $Q$, parameterized by a group of weights $\theta$, is updated at every iteration $i$ using the loss function

$$L_i(\theta_i) = \mathbb{E}[R_t + \gamma \max_{A_t} Q'(s_{t+1}, a_{t+1} : \theta^+) - Q(s_t, a_t, \theta_i)]^2, \quad (1)$$

where $L_i$ is the loss at step $i$, $\gamma$ the discount factor, $R_t + \gamma Q'(s_{t+1}, a_{t+1} : \theta^+)$ is the update target and $Q'$ is the target network parameterized by the weights $\theta^-$. The target network is updated periodically with network Q weights, $\theta$. This reduces the correlation between Q values and target Q values.

3. Implementation

A simplistic diagram of the main system implemented is presented in figure 1. The drone, which is a RL agent, flies in a simulated environment. During its flight, the drone receives information of the environment through its frontal camera (grayscale images). The image or group of images seen by the drone constitute the observation of the state of the environment and are used as input of the drone’s neural network. The network outputs the Q values of each possible action and the drone performs the action with the highest Q value. The drone position changes and the system cycle continues.

The drone is capable of performing three actions: move forward 50 centimeters, turn left 45 degrees, and turn right 45 degrees.

Figure 1: Diagram of the implemented system. 1 - The drone captures a picture of the environment which represents the current state. 2 - The picture is sent as input to the model’s neural network. 3 - The neural network outputs the Q values of each action and the action with highest value is chosen.

Our approach to the experimental activities was to perform tests step by step in different environments with increasing difficulty. This was made through small changes in the system (such as the level configuration and the wall and floor textures) and in the drone model (the neural net, its inputs, and training parameters). The purpose was to move slowly and steadily in the direction of the final objective, while validating the tools and algorithms employed.

3.1. Algorithm

During training, the deep Q-learning algorithm is applied. The loss function used (equation 1) is the mean square error between the network’s prediction of the action value of the chosen action and a target value (the target value is the prediction of the next action value, made by the target network, plus the received reward from the environment). The optimizer selected for the weight update is Adam [8]. The policy learned is $\varepsilon$-greedy, which means that at each step there is a certain probability $\varepsilon$ of the action chosen being random. Algorithm 1 is a simplified version of the DQN algorithm [7]. It is adapted to the RL problem of the project and presented in a pseudo-code format.

3.2. Neural network architectures.

Four neural network architectures were considered for the experiment:

- **Alpha** - Two-layered neural network with fully connected layers. It receives as input two values (two positions) and has sixteen neurons on the hidden layer. The output is the Q value of all the possible actions. This network was only used for validation purposes.
Algorithm 1: Deep Q-learning applied to the thesis problem.

Initialize replay memory and other variables
Initialize network $Q$ with random weights $\theta$
Initialize target network $Q'$ with weights $\theta' = \theta$

for episode $= 1, M$ do
    for $t = 1, T$ or until drone collision (T $\rightarrow$ episode length) do
        Get frame and build state $s_t$ (with 1 or 3 frames)
        Get action $a_t$ from network $Q$ or randomly,
        with probability $\varepsilon$
        Perform action $a_t$.
        Receive reward $r_t$ and get new frame to build state $s_{t+1}$.
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in the replay memory
        if Number of episode steps is higher than threshold ("anneal start") then:
            Sample random batch of transitions from the replay memory
            Update network $Q$ weights wit a learning step
            Every C steps update target network $Q'$ with network $Q$ weights
            Decrease $\varepsilon$ (when it reaches a final value it stops).
        end if
    end for
end for

- **Bravo** - Architecture based on the one used in Nature’s paper [7] (diagram in figure 2). The input is a grayscale image (1 channel) of the view of the drone with 120 pixels of width and height, giving it a final shape of 120x120x1. The network is composed of three pairs of convolutional plus ReLU layers. The first convolution has 32 filters with size of 8 and stride 4, the second 64 filters with size 4 and stride 2 and the third 64 filters with size 3 and stride 1. The layers are then followed by a fully-connected hidden layer with 512 neurons, another ReLU layer, and a final fully-connected layer. This final layer is the output layer and, similarly to network Alpha, it delivers the Q values of each possible action and will have one neuron for each.

- **Charlie** - CNN with architecture identical to architecture Bravo except for the input. To take into account past actions of the drone, the input is the aggregation of the current view of the drone and the two previous views (three frames in total). Each frame by itself is a channel and, therefore, the shape of the input is 120x120x3.

- **Delta** - The Delta network is an adaptation of a VGG16 network [9]. The VGG16 is a deeper convolutional network, used for large scale image classification. Its structure is depicted in figure 3. To build network Delta the input layer is changed to receive three frames of 120x120. The last fully connected layers are replaced with two fully connected layers: a 512 neurons hidden layer and an output layer (that delivers the Q value for each action of the drone, as in the previous architectures). The weights on the VGG base are pre-trained on ImageNet database [10], and all of them except the ones on the last convolution layer are frozen.

The images used as input for neural networks Bravo, Charlie, and Delta are obtained after processing the images captured by the drone camera. The simpler and smaller the image the simpler the neural network and less the number of trainable weights. By gray-scaling the images, the number of channels is reduced from 3 to 1. Together with the decrease in resolution, these image processes allow to decrease the number of weights in the neural network while maintaining critical shapes in the

---

**Table 1: Neural network architectures considered.**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Type</th>
<th>Input</th>
<th>Number of parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>FC</td>
<td>2</td>
<td>99</td>
</tr>
<tr>
<td>Bravo</td>
<td>CNN</td>
<td>120x120x1</td>
<td>7.450.787</td>
</tr>
<tr>
<td>Charlie</td>
<td>CNN</td>
<td>120x120x3</td>
<td>7.450.787</td>
</tr>
<tr>
<td>Delta</td>
<td>CNN</td>
<td>120x120x3</td>
<td>17.076.035</td>
</tr>
</tbody>
</table>

---

Figure 2: Diagram of architecture Bravo.

Figure 3: VGG16 diagram where only the layer organization is represented. The difference to architecture Delta is that the softmax operation is not applied and the fully connected layers are different and only 2.
image, that the neural network must recognize to guide the drone.

3.3. Simulation worlds
All the simulation worlds were constructed using Unreal Engine 4 [11]. They were built successively more complex as they integrate texture and structure randomization:

- **World 1** - Simplistic scenario where a drone has to center itself with a randomly spawned ball while only capable of going left, right and hover (three actions). The main purpose of this world is to deliver a simple and easy exercise to verify the correct functioning of all the tools and plugins. Figure 4(a).

- **World 2** - World with no obstacles and fixed textures. The configuration is a squared corridor (figure 5(a)). The objective is to have the drone fly for as long as possible. The drone spawns randomly in predefined locations, chosen to represent all possible and reasonable spawning points, with a random orientation. Figure 4(b).

- **World 2T** - World equal to the previous one in all aspects, with the exception that now the textures of the walls and floor are randomly generated at each episode. The training focus is on the generalization of the model towards different textures. Figure 4(c).

- **World 3** - This world has 4 maps with different configurations (figure 5(b)) that include curved and narrow corridors, dead-ends, open spaces, crossings, and more. The world purpose is to have the model generalize not on the textures (like the previous world) but on the shape of the environment. Figure 4(d).

- **World 3T** - This world adds the random textures to the complex environment of World 3. The objective is to learn a model that generalizes both on the shape and on the textures of the environment. Figure 4(e).

- **World 4** - World 4 is a world for testing. The best models are compared to each other and to chosen baselines. The configuration takes into account the different type of situations that a drone may face indoors (figure 5(c)) and the textures implemented were never seen by the drone during model training. Figure 4(f).

3.4. Software
The experiments were performed in Windows operating system (OS) for compatibility reasons. The coding language chosen was Python. Besides generic software and tools, that do not influence the experimentation and results, the main tools are now presented.

3.4.1. Unreal Engine 4
For the implemented methodology, the physics/graphics engine of choice was Unreal Engine 4 (UE4) [11]. The simulation environments were built inside the engine using specific tools and were populated with items, such as actors and components (walls, objects, the drone, etc.). To use UE as a research platform for deep reinforcement learning the following plugins were installed:

- **AirSim** [12] (v1.1.10) - An open-source plugin that offers a physically and visually realistic simulator for drones, cars and more.

- **UnrealEnginePython** [13] (v20180507) - This plugin allows the embedding of Python directly into UE4.

- **tensorflow-ue4** [14] (v0.7.0) - A plugin that enables the usage of TensorFlow in UE4.

3.4.2. AirSim
AirSim offers an API and a set of tools dedicated to AI research in the field of Reinforcement Learning. The vehicle chosen was the Multirotor (drone), for which two main flight modes exist:

- **Multirotor mode** - The drone behaves like a realistic drone with dynamic movement.
Computer Vision mode - The drone moves instantaneously without any type of dynamic considerations.

All the experiments were made using the Computer Vision mode. This choice is justified by the considerable decrease in training time and by the fact that the Multirotor mode was unreliable (the API did not perform as supposed for some specific movements, such as moving forward). For consistency purposes, each time the drone moves, it does so in steps of 50 centimeters in the intended direction. Every time it turns over its yaw axis, it does so in steps of 45° degrees.

Besides the flight mode, the plugin offers a set of cameras that are crucial for the reinforcement learning problem of this thesis. Only the front camera is utilized during the experiments but two vision modes are considered:

- **RGB** - The camera captures an RGB image of the environment similar to what happens in most drones that carry a monocular camera. It was used to capture the environment state for training.

- **Depth** - Each pixel of the depth image carries a number that corresponds to the distance of that point in space to the camera. The value is presented in tenths of meters. This camera was used by the reward functions to help calculate the reward signals.

4. Experiments and Results

All trained models were evaluated on their objective of flying the drone without colliding. While analyzing results we present evaluation metrics, which measure the drone success at accomplishing its goal. These evaluation metrics are distance covered (in meters), return, and number of episode steps until collision. For each model 100 test runs were performed, where each run is an episode of 50 steps. The metrics values were averaged over the 100 episodes.

4.1. Training parameters and reward functions

Some training parameters were fine-tuned in order to get models with better performances. The most important to mention are:

- **Discount factor** $\gamma$ - This parameter shifts the importance that is given to future and present rewards. This parameter is commonly left fix at a value over 0.9 (e.g. 0.99) but we change the value since this work gives more emphasis to the present.

- **Learning rate (LR)** - The learning rate is a parameter of the optimizer used for training. It influences how big the changes on the network weights are (per training step).

- **Reward function** - On a reinforcement learning problem the reward function is one of the main elements and greatly influences what the drone learns.

- **Anneal start** - Number of steps until $\varepsilon$ starts to be linearly annealed. Also, the training only starts after this number of steps.

Designing the reward function was an empirical and iterative process. The two final reward functions that were tested are:

- **Reward R1** - This first function rewards the drone for moving forward (+1) and penalizes it when it collides (-1). If the drone gets too close to a wall while moving forward the drone is penalized with a value of -0.5. Whenever the drone turns it receives a negative reward of -0.1.

- **Reward R2** - The second function is an evolution of reward function R1. It is similar in every way except for one addition: if the drone turns it is penalized with the value -0.1 unless the turn clears the path in front of it. In this case the UAV is rewarded with a value of +0.5.

While some rewards depend only on the action or on a specific event (collision), other rewards require more information from the world. The drone knows the path is clear in front of it and that it is too close to a wall through the depth camera.

4.2. World 1

World 1 served the purpose of validating the different models and tools adopted for the experiments. The results were successful and the trained models learned to center the drone with the ball. Figure 6 presents the policy and Q-function learned by a model with an Alpha network. Figure 7 shows frames captured by the drone during a successful run.

![Figure 6: Policy and Q value for each action according to the ball and drone position, at left and right respectively. The position axis of the drone and the ball goes from right to left.](image-url)
4.3. World 2

Table 2 summarizes the models that were trained in World 2.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training World</th>
<th>Architecture</th>
<th>( \gamma )</th>
<th>Reward Function</th>
<th>Amen start</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2A1</td>
<td>World 2</td>
<td>Bravo</td>
<td>0.99</td>
<td>R1</td>
<td>1000</td>
</tr>
<tr>
<td>M2B2</td>
<td>World 2</td>
<td>Bravo</td>
<td>0.8</td>
<td>( 1e^{-3} )</td>
<td>1000</td>
</tr>
<tr>
<td>M2C5</td>
<td>World 2</td>
<td>Charlie</td>
<td>0.99</td>
<td>R1</td>
<td>1000</td>
</tr>
<tr>
<td>M2C6</td>
<td>World 2</td>
<td>Charlie</td>
<td>0.8</td>
<td>( 1e^{-3} )</td>
<td>R1</td>
</tr>
<tr>
<td>M2C7</td>
<td>World 2</td>
<td>Charlie</td>
<td>0.8</td>
<td>( 1e^{-5} )</td>
<td>R2</td>
</tr>
<tr>
<td>M2C3</td>
<td>World 2</td>
<td>Delta</td>
<td>0.8</td>
<td>( 1e^{-3} )</td>
<td>R2</td>
</tr>
</tbody>
</table>

Table 2: World 2 trained models.

4.3.1. Quantitative results

Table 3 holds the evaluation results for the models trained in World 2, evaluated in World 2. The results for the first 4 models give good insights on how the network input and the discount factor \( \gamma \) affect the performance of the models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean number of steps until collision</th>
<th>Mean distance covered (m)</th>
<th>Mean return</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2A1</td>
<td>49.31</td>
<td>8.803</td>
<td>10.106</td>
</tr>
<tr>
<td>M2B2</td>
<td>43.81</td>
<td>13.049</td>
<td>22.325</td>
</tr>
<tr>
<td>M2C5</td>
<td>49.16</td>
<td>11.095</td>
<td>19.370</td>
</tr>
<tr>
<td>M2C6</td>
<td>43.57</td>
<td>16.195</td>
<td>30.017</td>
</tr>
<tr>
<td>M2C7</td>
<td>47.21</td>
<td>19.415</td>
<td>37.456</td>
</tr>
<tr>
<td>M2C3</td>
<td>47.30</td>
<td>19.200</td>
<td>37.236</td>
</tr>
</tbody>
</table>

Table 3: World 2 models’ evaluation results.

The first thing to notice is that the models with a discount factor of 0.8 (\( M2B2 \) and \( M2C2 \)) achieve far greater results than the models with the same input but higher discount factor (\( M2B1 \) and \( M2C1 \)). This might be explained by the fact that the network does not need to worry about the far future. The network focus on present rewards and shifts the weights more efficiently. During the episodes, the drone does not have a specific end goal. The goal is to move forward as much as possible and that only depends in the near future.

The change of the network’s architecture from \( M2B1 \) to \( M2C1 \) and from \( M2B2 \) to \( M2C2 \) explains the increase in return, which is also related to the increase in covered distance. Architecture Charlie has memory, since it receives the frames from the last two movements. Models with network Bravo get easily stuck when facing a wall or a corner, oscillating between turning left and right, while models with a Charlie network learned to overcome this situation easily. To support the previous statement, figure 8 on the qualitative analysis section demonstrates how models with different inputs behaved on a sample test run.

Another point worth mentioning for the first set of models is that the models with higher returns also have a lower mean number of episode steps until collision. From models \( M2B1 \) and \( M2C1 \) to \( M2B2 \) and \( M2C2 \) the models’ policy started taking more risks and ended up colliding more. On the good side, this behavior leads to higher returns. If a model is too cautious, the drone tends to get stuck and turn more, avoiding to go forward even in situations where it has room for it. In the remaining models, however, this point is not as evident.

To finalize, \( M2C \) and \( M2D \) achieved outstanding return values when compared to the first 4 models. The discount factor chosen came from the tests in the previous models while the other parameter values were tuned empirically. Decreasing the discount factor to 0.8 definitely improved results. Smaller learning rates may allow to reach a better minimum of the loss function, so decreasing its value from \( 1e^{-3} \) to \( 1e^{-5} \) might also have helped. The anneal start value influences how long the exploration of the environment is totally random (\( \varepsilon = 1 \)). By increasing this value we fill the memory replay with a larger and more diverse quantity of examples. The different reward function could also influence heavily the results. Unfortunately, without training other combinations of parameters, it is hard to tell for certain which factors influenced the increase in quality of the models.

4.3.2. Qualitative results

Regarding the qualitative analyses, the first batch of models is clumsy and even the best one tends to get stuck or do more turns than necessary. Figure 8 shows the trajectories of three models during a sample run. Model \( M2B1 \) controls the drone straight along the corridor but it stopped at the end, where it remained oscillating. Stopping the movement and getting stuck oscillating, left and right, is common for models \( M2B1 \) and \( M2C1 \). In the figure, model \( M2C1 \) has a better performance than model \( M2B1 \), although it does some unnecessary turns. The last models, \( M2C \) and \( M2D \), can drive the drone through the entire extent of the corridor and almost never got stuck. In the figure, we see that model \( M2C \) only got a bit confused when performing the turn to the right.

4.4. Models performance tests

After analysing the results of models on World 2, the remaining models were trained on the other worlds, using the training parameters and reward functions that generated the best model on world 2. The list of models considered for evaluation is on table 4.

The trained models and some baselines are de-
Figure 8: Representation of three model trajectories on World 2. All models departed from the green dot.

Table 4: Main models trained.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training World</th>
<th>Architecture</th>
<th>γ</th>
<th>LR</th>
<th>Reward Function</th>
<th>Anneal start</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2C</td>
<td>World 2</td>
<td>Charlie</td>
<td>0.8</td>
<td>1e−5</td>
<td>R2</td>
<td>5000</td>
</tr>
<tr>
<td>M2D</td>
<td>World 2</td>
<td>Delta</td>
<td>0.8</td>
<td>1e−5</td>
<td>R2</td>
<td>5000</td>
</tr>
<tr>
<td>M2T</td>
<td>World 2T</td>
<td>Delta</td>
<td>0.8</td>
<td>1e−5</td>
<td>R2</td>
<td>5000</td>
</tr>
<tr>
<td>M3</td>
<td>World 3</td>
<td>Delta</td>
<td>0.8</td>
<td>1e−5</td>
<td>R2</td>
<td>5000</td>
</tr>
<tr>
<td>M3T</td>
<td>World 3T</td>
<td>Delta</td>
<td>0.8</td>
<td>1e−5</td>
<td>R2</td>
<td>5000</td>
</tr>
</tbody>
</table>

7 steps forward. The episode lengths are also short and the mean return is worse for both than for the Random baseline. In fact the models tended to guide the drone straight until collision, turning here and there without a specific purpose (as described on the qualitative analysis).

Looking at the results for models M2T and M3, it is possible to see a vast improvement and to notice that they are quite similar. The episode length and the distance covered is almost the triple (in some cases it does in fact triplicate). The worst mean return of M2T and M3 (17.823) is five times higher than the best mean return of the first two models (3.552). Comparing to the evaluation results on their own training worlds, however, World 4 mean returns barely go over half of the training world mean return (around 30). This is an expected result since the evaluation on the training world is essentially testing the model on its own training data.

Model M3T trained on the more complex environment and in theory would be the model with the best results among all that were trained. In fact, the model performance is on par with the previous two models, M2T and M3. When comparing with the baselines, all models performed better than the Random and the DroNet baselines (with the exception of models M2C and M2D). The DroNet baseline seems to perform better than the Random baseline but considering how close together they are it is hard to say that DroNet is in fact better. Two major factors contribute to this result: the fact that DroNet is a network trained to fly a UAV in an outdoor scenario, and that it is trained on real life footage and not on simulated images. With this in mind the results are indeed understandable. The last baseline is the Human. The human piloted the UAV without ever colliding and got the best mean return (42.37) and mean distance covered (22.25 m). The task is quite simple for humans, specially considering its discrete dimension. The trained models with best results can only barely reach half the return and half of the distance covered. This proves that there is still a lot to be improved regarding the training specifications (algorithms, parameters and reward function) and regarding the models’ architectures.
Qualitative analyses

Figures 9 and 10 represent three trajectories performed by each model trained and each baseline, respectively, on World 4. The drones departed from three different positions with random orientation.

Figure 9: Trained models qualitative results. The green dots are starting points of the trajectories.

Figure 10: Baselines qualitative results. The green dots are starting points of the trajectories.

Qualitatively, models $M^2T$, $M$, and $M^3T$ were able to navigate the environment to some point, all of them similarly. The trajectories from figure 9 demonstrate how the models control the drone, sending it straightforward and only making it turn when necessary (at least until the network fails to recognize a wall and a collision happens). Model $M^2D$, on the other hand, kept sending the drone forward as it is possible to see from the blue trajectories on figure 9. The network clearly does not generalize to World 4 and behaves as if it was perceiving the environment as an open space.

From figure 10, it can be concluded that the Human baseline is indeed the best one. The drone never collided and traveled the environment smoothly, only turning when it had to. DroNet seems to be able to control the drone in some situations (the two bottom green trajectories on figure 10), but in others its quick to collide (top trajectory on figure 10). Finally, the Random baseline, as expected, takes random trajectories without a specific objective, colliding always after some steps.

5. Conclusions

In this thesis project we implemented a deep reinforcement learning (RL) method to teach a drone how to fly indoors. The training was done with a DQN algorithm in distinct simulated environments, with the purpose of testing different levels of generalization. All simulated worlds were developed in Unreal Engine, and models were trained in each of them. They were all tested together on a test world and the results were analyzed and compared to each other.

As the complexity of the world increased the harder it was for models to learn a good policy. The models trained in the last worlds made more mistakes and ended up colliding sooner than in previous worlds. In the test world, on the contrary, the generalized models $M^2T$, $M$, and $M^3T$ achieved far better results. It is also interesting to notice that the model trained with generalization for both map structure and textures ($M^3T$), performed slightly better than the models that were trained with only one type of randomization ($M^2T$ and $M$). They performed better than almost all baselines. The human baseline had the best results of all models and other baselines tested, and it clearly indicates that the models we trained have a lot of room for improvement.

5.1. Achievements

In this work we trained different models for indoor collision-free flight using DQN in a simulated environment. A simple reward function that can be deployed for training or fine-tuning in a real world scenario was developed. The drone only needs to carry an RGB and a Depth camera as sensors for that purpose. This is a contribution over previous works such as CAD2RL [15], which requires extra information from the simulator to perform the training.

The discount factor proved to be determinant during training. For the first set of models in World 2 the results more than doubled when the discount factor was decreased from 0.99 to 0.8. From that point on all models were trained with a 0.8 value. Since the RL problem focus only the immediate behavior of the drone, reducing the importance of future rewards is what might have helped the neural network approximate the real Q-function.

The study on generalization also brought good insights. Models with a generalized policy perform better in a new environment with never before seen textures than models trained without any type of generalization. This notion might be the key to adapt models from simulation to the real world.
5.2. Future Work

There are many interesting aspects that can be added and looked into in future approaches to the problem. One of the most obvious is to perform the transition to a real drone. A logical road map might be:

- **Train continuous models** - Train a drone with dynamic movement instead of discrete. This change will bring a more realistic setting to the simulated environment and might prove fundamental to adapt the model to the real world.

- **Perform the training in more diverse and/or realistic scenarios** - The worlds in this work could be improved with additional items such as common objects (chairs, tables, closets) and structures (stairs, doors, windows). The lighting, for example, can also be varied for the same environments.

- **Transfer the model to a real drone** - Demonstrate the learned models on a real drone and further improve them through more training.

A different possibility to further work on the problem can be to explore the possibility of adding an end goal to the problem. Besides the collision-free flight, the drone could also learn the map of the environment or search for a specific target. Adapting the existent networks or developing a new one with this in mind can be a continuing point for this work.

**References**


