UiT The Arctic University of Norway

Department of Computer Science and Computational Engineering

# Drone Swarm Simulator

Luka Jeroncic

Master's Thesis in Applied Computer Science, DTE3900, May 2021

## Summary

This thesis is about the research of available and development of drone swarm simulators. The simulator in this work, developed with the AirSim plugin for Unreal Engine 4, shows that there is a possibility of a real-time, accurate simulator with high graphical quality.

## Preface

I've chosen to write about this topic because I was interested in drone swarm systems, especially ones without external positioning systems. I was interested in different kinds of control methods and collision avoidance within the swarm.
I would like to thank my advisor and supervisor for their limitless patience and high tolerance for my unseriousness. I would also like to thank my grandma for giving me a room to work in.

# Contents

## List of Figures

# 1 Introduction

Lately, a lot of attention has been brought to unmanned aerial vehicle (UAV) technologies due to their increasing use in military and commercial purposes. For certain tasks such as tracking, surveillance, path planning, and coordination, swarms of UAVs have been shown to provide great usefulness and much better properties and operational parameters than applications running on a single UAV [1]. The cooperation of the drones within a swarm raises additional problems not present in a single UAV operation. Swarms operating as one unit need systems for collision avoidance among the drones in the swarm itself as well as avoidance of obstacles found in the environment [2]. Failure in these systems can cause material damage and increase the cost of developing such systems. Therefore, a simulator capable of visualizing the performance of the drone swarms should be a welcome addition to the industry.
An unmanned aerial vehicle (UAV) is an aircraft without a pilot, controlled from the ground or by a computer onboard [3].
The first uses of UAVs were recorded in the 19th century for military use [4]. The drones first came in the form of balloons. The first-ever practice of aerial surveillance emerged in 1898. when the U.S. military fitted a camera to a kite. A few decades and many innovations later, in mid 20th century, more efforts were put into the research of UAVs to minimize the use of human on-board pilots. Since then drones have been developed for commercial uses, the most popular one being for video recording purposes.
There are two main types of modern UAVs: fixed-wing and multi-rotor. Fixed-wing uses velocity to generate uplift, while multi-rotor uses multiple rotors to generate uplift like a helicopter. The main drawback of a multi-rotor UAV is very high power consumption, resulting in limited endurance [5].
A simulation is the imitation of the operation of a real-world process or system over time [6]. Keith Douglas Tocher created the General Simulation Program, the first general-purpose simulator for building a simulation of an industrial plant in 1958 [7]. Since then, many simulators have been developed for various purposes, greatly reducing the development cost and time.

## 1.1 Literature review

There is plenty of drone flight simulators around. Usually, drone simulators offer control of a single UAV [8, 9]. These simulators offer very accurate, true-to-life representations of drones and their controls. Accurate simulation of physics is important in these simulators to resemble reality and make the switch from a simulation onto real performance as seamless as possible. These simulators, however, do not provide any swarming mechanics.

When it comes to simulators for UAV swarms, the existing ones largely focus on mathematical models of swarming and sub-par visualization methods [10, 11]. This work is an attempt at bridging the gap between accurate physics and realistic visuals of a proper simulator and the mathematical concepts of swarming.

Some highly modular simulation software exists for development of such systems, including Gazebo [12], Modular open robotics simulation engine MORSE [13], Flightmare [14] and AirSim [15]. These frameworks provide groundwork in physics and sensor simulation and allow for further development of robotic systems.

Flocking or swarming is a phenomenon observed in natural organisms which have been a subject of many studies. Consequently, there are many research papers devoted to different aspects of flocking. One such research is [16] which describes the collective motion as a system of units that, among other things, move with nearly constant absolute velocity and are capable of changing their direction. This work is thus more applicable when a UAV in question would be of a fixed-wing type. Nevertheless, the work mentions simple interaction methods between units such as attraction and repulsion. It also emphasizes the importance of a simple model to capture the essential features of a swarm.

In [17] flocks of birds are compared to particle systems in computer graphics. While usually just a graphics programming issue, flocks as a particle system here are achieved through simulation of each particle unit. Even though it is mainly oriented towards the animation of the flock or particle system, a big part of the paper is also collision avoidance of environmental obstacles. It gives insights and ideas for modeling and avoidance of obstacles.

Extensive work on creating a quadrotor swarm that works in reality was presented in [18]. It identifies a gap between simulators and reality by introducing real-life conditions such as delays, uncertainties, and kinematic constraints. It shows good results both in simulation experiments and real-life counterparts with the minimum inter-drone distance being 5-15 meters. The work uses a repulsion mechanism and even though it mentions attraction, it actually only implements space boundaries for the agents. It provides algorithms and equations for repulsion and velocity alignment when the drones are in the mid-range distance from each other.

A good source for different concepts and methodologies can be found in [19]. It collects and categorizes many different pieces of research on the topics of robotic swarms. While not providing a lot of material that this work could directly benefit from, it is a very good pointer to other works that could be of use throughout this work.

Similarly to [18], [20] provides an algorithm that is proven to work in reality. It also accounts for the delay in communication and noise. It shows how these affect the results and swarming performances. It was tested and proven in a swarm of 9 quadrotor UAVs.

## 1.2 Contributions and scope of the thesis

The first task is to research previous works and identify gaps and shortcomings of the same. This has partly already been covered in the Literature review section above.
After the research, an attempt will be made to create a simulator or modify an existing one to allow for the swarming of quadrotor UAVs. The simulator is to be in 3D and account for all six freedoms of movement for the UAVs. The UAVs are expected to behave correctly according to laws of physics as much as possible considering the limitations of simulation software. Some exceptions apply which will be named in the limitations section.
To realize the swarm simulator it is imperative that the simulator is consistent and modifiable while maintaining the number of collisions as low as possible. Meaning that when parameters are unchanged the simulator should yield similar results. The parameters, however, should be modifiable so different combinations of parameters can be compared.
Consistency will be measured using standard deviation from average time to complete a task.
The core of the task is to successfully implement swarming algorithms. Primarily using any means available provided by the software, and following that by implementing features such as UAV to UAV communication, delay, and noise. The visual quality of the simulation is of high importance to make it appealing to a wider audience.

## 1.3 Existing simulators

**RotorS, Hector, Gazebo** RotorS and Hector use Gazebo for the simulation of quadrotor models. They sacrifice visual quality for performance. They are flexible and with some work swarming could be implemented. Gazebo uses the DART physics engine [21] by default and OpenGL for rendering. Other than DART, it also supports Bullet [22], Open Dynamics Engine [23], and Simbody [24] physics engines.
**AirSim**: Unreal Engine 4 [25] is the latest iteration in the Unreal Engine series of game engines developed by Epic Games. "Game engine" is not a well-defined term but it is widely used for software intended to develop video games. It provides tools such as a renderer and a physics engine which are needed not only for the development of video games but for anything that could benefit from those tools, such as simulators.
AirSim [15] is a plugin for Unreal Engine 4. It benefits from Unreal's class structure and renderer by using it as an environment for running and visualizing simulations. It uses its own physics engine that can operate at a high frequency for real-time hardware-in-the-loop (HITL) simulations with support for popular protocols. It is designed from the ground up to allow for the addition of new types of vehicles, platforms, and protocols.

**Flightmare**: Similar to Airsim, Flightmare uses Unity Engine for rendering high-quality visuals. It is separated into main components: a configurable rendering engine built on Unity and a flexible physics engine for dynamics simulation. It supports multiple drones, just like Airsim. However, it only provides API in Python.

## 1.4 Limitations

In order to make this work possible some limitations have to be defined in the beginning. Considering the main task of the thesis, which is swarming, some assumptions are made in order to simplify the work. Putting a lower limit on the number of drones is just guesswork but following the examples of other works, at least eight drones should be an achievable goal. There will be no external disturbances such as wind or turbulences. The world will be assumed as a flat, non-rotating earth. The UAVs will fly at a relatively low altitude with constant gravitational forces. As the work gets more complicated and sensors and communication techniques are implemented, realistic attributes of such technologies should be considered and imitated.

## 1.5 Report outline

A large part of the thesis has been covered in the above chapters. Research and analysis of previous works and available technologies are important for the first part of this thesis.
The chapters below mostly deal with the development of a swarm simulator by using the API provided by Airsim. It starts with preliminaries which are good to know for understanding the subsequent content of the thesis. It includes notation and brief descriptions of concepts used later.
After that, methods used in the development are described in order to familiarize the reader with the way that the simulator works. It defines a task of the swarm and algorithms used for swarming and collision avoidance. The scene setup which is used for gathering results is described.
After methods, the results of the simulations are shown with times to complete the task and relevant data such as the number of collisions. Results of different simulations are gathered in a table for easier comparison and analysis.
In discussion, the results are given an interpretation. The work is viewed with a critical view, the drawbacks are discussed and improvements are suggested.

# 2 Preliminaries

In this section mathematical notations which define the state of the swarm and its agents are defined which will be used later in the work. Some concepts regarding swarm control and other relevant topics are presented. A further description of Unreal Engine 4 and AirSim is also included as those will be a large part of this work.

## 2.1 Notation

A swarm of QUAVs (Quadrotor Unmanned Aerial Vehicles) is a set:

$$S = \{d_1, d_2, ..., d_n\} \tag{2.1}$$

where:

$d_i(t) = \{x, y, z\}$ is a three-dimensional position of a drone $i$ with respect to time $t$, and $n$ is the number of drones in the swarm.
Distance between a pair of drones $(d_i, d_j) \in S$ is $|d_i - d_j|$. The unit vector which points from $d_1$ to $d_2$ is $\triangleleft(d_i, d_j)$

## 2.2 Unreal Engine 4 and AirSim

Unreal Engine 4 (UE4) is used for a very large variety of applications because of its availability and modularity. It offers one of the best free-to-use renderers. Additionally, it boasts a very large marketplace of different sorts of assets that are very easily added to projects. It also allows for a very easy in-engine addition of simple geometrical objects to the project scene, such as cubes, spheres, and cylinders. UE4 also allows plugins. Plugins are described on the UE4 website as "collections of code and data that developers can enable or disable within the Editor on a per-project basis" [26]. Plugins can add runtime functionality, modify engine features and add new ones, and more. A screenshot from the UE4 editor can be seen in Fig. 1
AirSim is developed as a plugin for UE4. It uses its own physics engine and allows for the addition of AirSim vehicles from its settings file. The settings file is used to define AirSim defined objects and their attributes, such as vehicles, sensors, and their starting positions, while the environment is handled within the UE4 editor. AirSim provides APIs for controlling the drones, gathering data from sensors, and other functionalities. One example of a control API is a function that takes a velocity vector as a parameter for control of a drone.

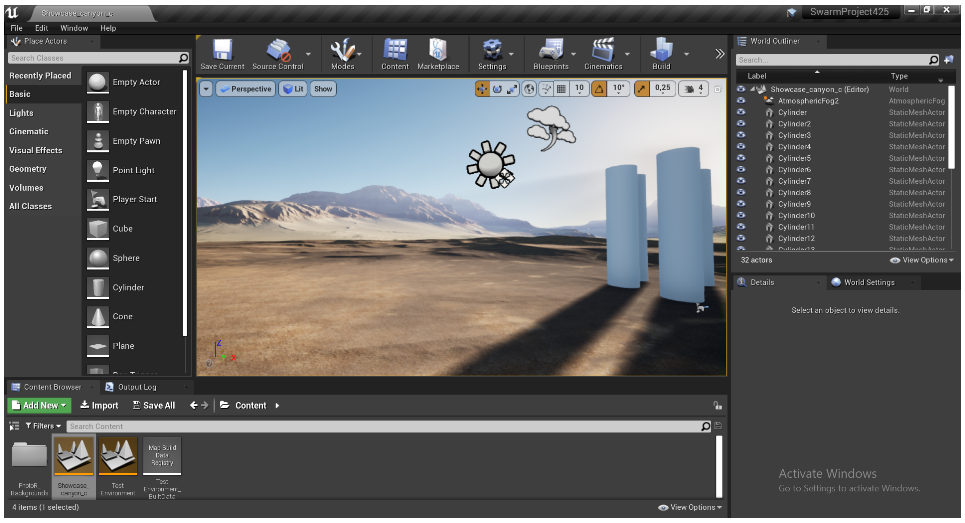


Figure 1: Unreal Engine 4 editor

## 2.3 Quadrotor UAV dynamics

Quadrotor UAV (QUAV or just quadrotor) uses four propellers to generate lift. A sketch of a QUAV is shown on Fig. 2. The quadrotor is moved by creating a disbalance in power given to each propeller. Each propeller provides lift and torque to the quadrotor. Two of the propellers rotate clockwise, and two rotate counterclockwise, as shown in the figure 2. For moving the drone up or down, power to all motors is equally added or reduced. For tilting it forward, more power is given to propeller 3 and 4, while is given to propellers 1 and 2. Tilt in other directions is realized similarly, by a combination of such disbalances. Yawing the quadrotor is achieved by creating a disbalance between torques provided by propellers with opposite rotation directions.

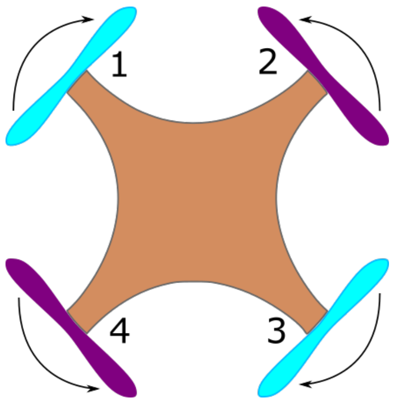


Figure 2: Quadrotor sketch, showing rotations of each propeller

## 2.4 Repulsion and attraction preliminaries

The concept of repulsion and attraction (R&A) is important for the agents of the swarm to act cohesively, controlling the swarm size and behavior. The core of R&A is the wanted inter-UAV distance $D_{int}$. Wanted inter-UAV distance is the distance at which each drone wants to stay at from every other drone. To make that happen, virtual R&A forces are introduced [27].
The virtual force between a pair of drones $(d_i, d_j) \in S$ is attractive if $|d_i - d_j| > D_{int}$, and repulsive if $|d_i - d_j| < D_{int}$.

### 2.4.1 Activation functions

In order to translate the distance between drones into the force an activation function $f(D)$ is introduced. For the drones safety, it is imperative that the repulsive force is larger than the attractive force for the same distance magnitude $|D|$. A class of R&A functions for stable swarm aggregations is described in [28]. In order to keep it simple, linear forces will be used in this work.
The activation function $f(D)$ can be divided into separate functions for R&A: $f_a(D), D > D_{int}$, and $f_r(D), 0 < D \leq D_{int}$. Since the functions depend on $D_{int}$ the function $f_{rel}(D)$ can be offset by $D_{int}$ so we define $f_r(D_r) = f(D - D_{int})$. An example of $f(D_r)$ is shown on Fig.3.

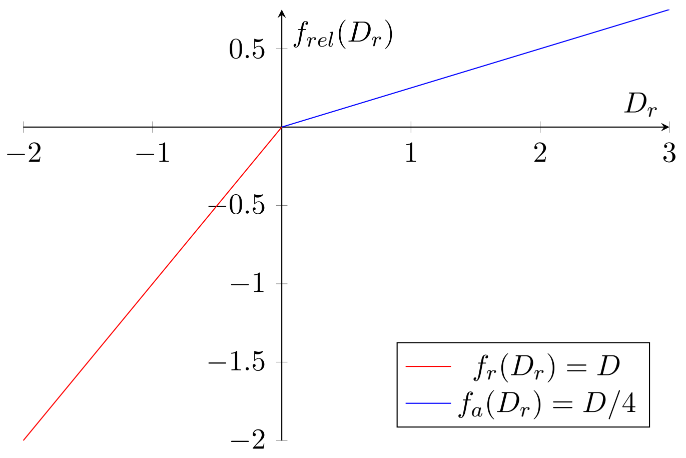


Figure 3: Activation functions example

## 2.5 Sensors preliminaries

In this work, AirSim's distance sensors are used. These sensors are attached to the center of the drones and are defined by their orientation and maximum distance $S_{max}$ that they can detect. Both parameters are configurable in

AirSim's settings file. The data from the sensor comes in the form of a floating point number $S$, distance in meters from the drone detected in the sensors orientation.
The distance sensor in this work will be used for environmental collision avoidance, with a similar function to the repulsion aspect of R&A. The sensor will thus have another activation function $f_s(S)$ with a limited domain $0 < S < S_{max}$. Increasing $S_{max}$ will increase the distance at which a drone can react to a detected obstacle. An example of the sensors activation function is given in Fig. 4, where $S_{max}$ is set to 4 meters.

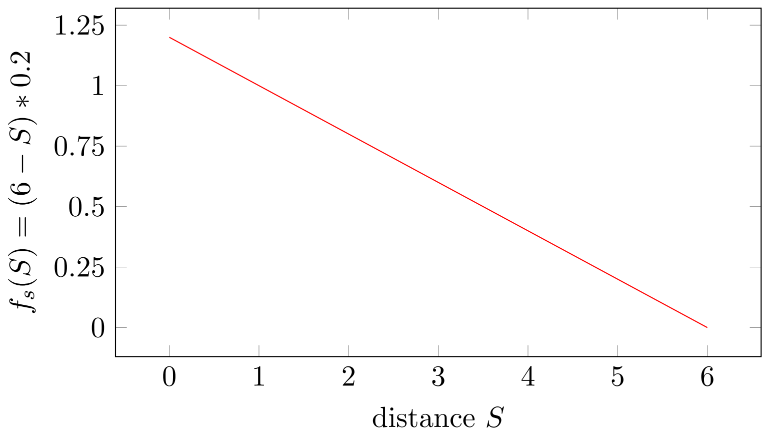


Figure 4: Sensor function example, Sensor maximum range set to 4 meters

# 3 Methods

AirSim has been chosen for implementing swarm systems simulation due to claims of multiple quadrotor support, the use of Unreal Engine 4 for rendering, and the C++ API which is preferred by the author of the thesis. Unreal Engine 4 should provide the highest quality visuals from the available selection of simulators.
A method of measurement is needed to keep track of the simulation performance. Time to complete a task and the number of collisions are the most important parameters to follow in a swarm application. The information about collisions is easily gathered by a call to an AirSim API function.

## 3.1 Task completion time

A task in this simulation is defined as moving a swarm from a starting position to an end position and completing the swarmed condition in the end. The swarm starts moving from a predetermined position towards the goal, reaches it, and swarms around it. The swarm condition is considered complete when

$$|d_i - d_j| < \varepsilon, \forall (d_i, d_j) \in S \tag{3.1}$$

where:

- $|d_i - d_j|$ is the distance between a pair of drones $d_i$ and $d_j$
- $(d_i, d_j)$ is any pair of drones $d_i$ and $d_j$
- $S$ is a swarm, or a set of drones as described in Eq. 2.1
- $\varepsilon$ is the maximum allowed distance between any two drones in order for it to be considered a swarm, or the size of the swarm.

The time to complete a task is thus defined as the time from the start of the task to the end of the task. It is measured using chrono, a part of the c++ standard library [29].

## 3.2 Repulsion-Attraction

Since a quadrotor has the ability of hovering, the algorithm for keeping a swarm of quadrotors relatively close is much simpler than that of fixed-wing type UAVs. Every effort to keep the drones swarmed includes attracting them towards each other when they are too far apart, and repulsing them away from each other when they are too close.
A number of parameters can be changed here in order to compare performances and justify the use of the simulator. Some of these are the target

distance between the drones, which also affects the size of the swarm, the functions which a drone uses to generate the response to the distance from another drone, and the number of drones itself. Finding the right desired distance is a balance between keeping the drones from spreading out too far or crashing into each other.
In this project, a system is implemented such that each drone takes into account distances to all the other drones in the swarm and reacts to them separately. The sum of these reactions results in the final result, which is the movement of the drone by a velocity vector.
The drones' safety is considered a priority, so different activation functions have been implemented for repulsive and attractive forces. Both functions are made linear, with varying gradients. The gradient of the repulsive force is larger to prevent drones from flying too close to each other and crashing. The default reaction functions are shown in Fig. 5. where $D_r$ $[m]$ is the distance between the drone to the wanted distance to the other drone, and $f_{rel}(D_r)$ $[m/s]$ is the reaction strength.

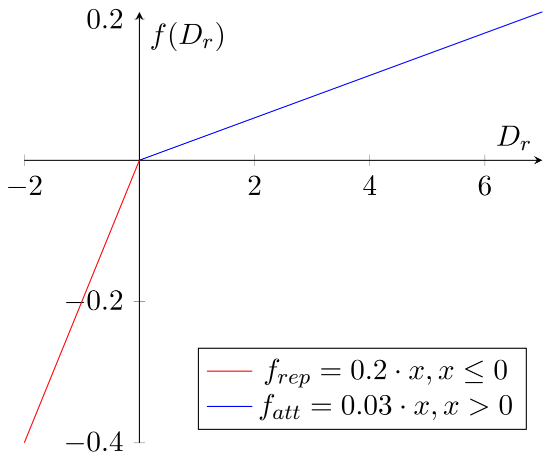


Figure 5: Activation functions

The final repulsion and attraction vector $\mathbf{C_i}$ for the control of the drone $d_i$ is:

$$\mathbf{C_i} = \sum f_{rel}(|d_i - d_j|) \cdot \sphericalangle(d_i, d_j), d_j \in S\backslash\{d_i\} \tag{3.2}$$

## 3.3 Environment collision avoidance

AirSim offers an API for distance sensors. The sensors are defined in the settings file by orientation and the maximum distance they can detect. To ensure sufficient coverage of the surroundings, multiple distance sensors are defined for each drone.
The sensors don't cover a range from an angle to an angle but a single ray

cast in the direction of the sensor. Eight sensors, denoted as $s_i, i \in \mathbb{N}, i \leq 8$, are set up in circular pattern with uniform angular offset, as shown in Fig. 6. The sensors directions are denoted as $\sphericalangle(s_i)$. The maximum range $S_{max}$ of each sensor is set to 4 meters. The feedback is provided in distance $S$ in meters to the nearest obstacle in the respective direction.
Because of the way that the sensors have been set up, the optimal collision object should not have sharp edges and it should be relatively large compared to the drone size. Thus the obstacles used in the project are large, pillar-like, round objects.
The function $f_{s1}(S)$ is a sensors activation function. It returns the reaction intensity in $m/s$ as a function of distance $S$ detected by the sensor, it is described in Fig. 7.

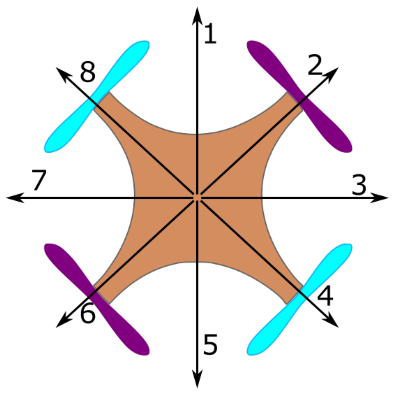


Figure 6: Quadrotor sketch, showing directions of sensors numbered 1-8

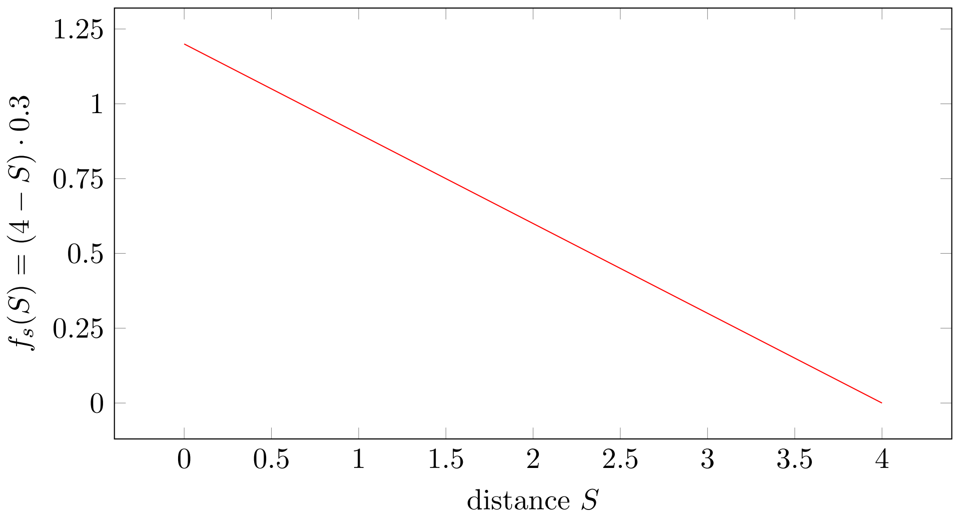


Figure 7: Sensor function

In addition to the first sensor activation function shown on Fig. 7, another function with a larger domain is made. It supports a sensor which has a slightly larger maximum detection distance. Both functions are shown in Fig. 8 for comparison.

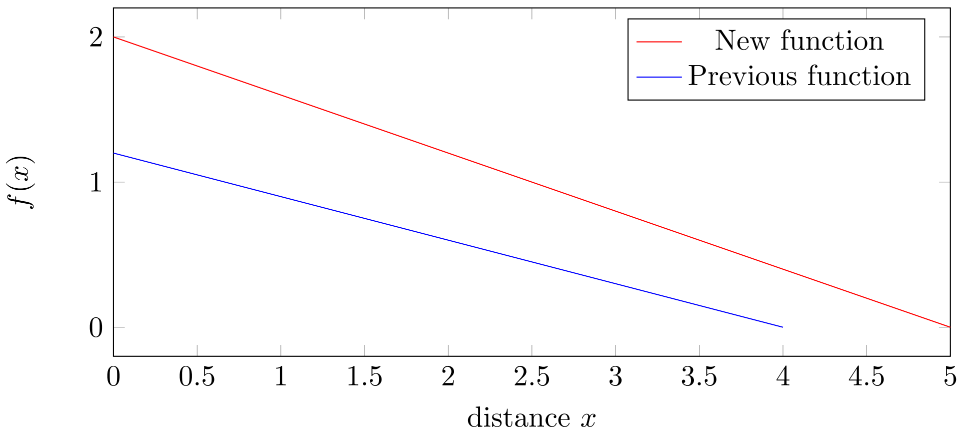


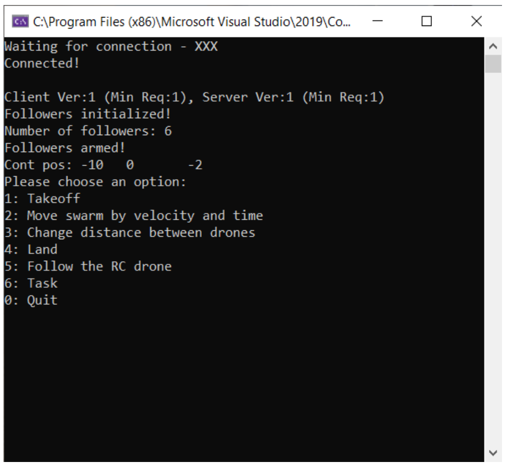
Figure 8: Larger domain sensor function

The final sensor input vector $\mathbf{C_{si}}$ for the control of the drone $d_i$ is:

$$\mathbf{C_{si}} = \sum_{i=1}^{8} f_s(s_i) \cdot \sphericalangle(s_i) \tag{3.3}$$

## 3.4 User interface

A simple user interface is made to be used in the programs console, it is shown on Fig. 9.



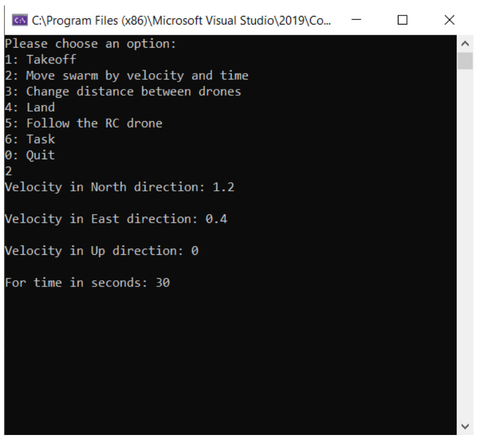
```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Co...
Waiting for connection - XXX
Connected!

Client Ver:1 (Min Req:1), Server Ver:1 (Min Req:1)
Followers initialized!
Number of followers: 6
Followers armed!
Cont pos: -10   0       -2
Please choose an option:
1: Takeoff
2: Move swarm by velocity and time
3: Change distance between drones
4: Land
5: Follow the RC drone
6: Task
0: Quit
```

Figure 9: User interface

It supports different commands, some of those are:

- Following the leader which is controlled by an RC controller
- moving as a swarm in a set direction by a set velocity for a set amount of time, shown on Fig. 10a
- changing the wanted inter-drone distance, shown in Fig. 10b



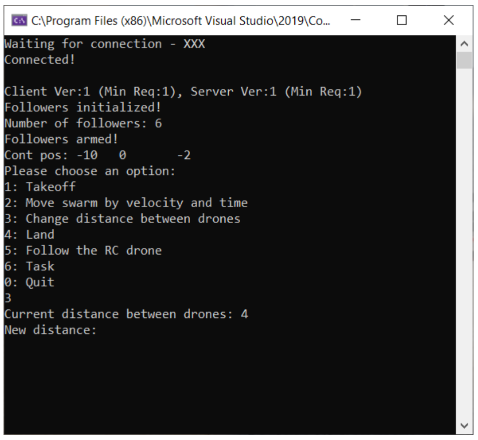
```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Co...
Please choose an option:
1: Takeoff
2: Move swarm by velocity and time
3: Change distance between drones
4: Land
5: Follow the RC drone
6: Task
0: Quit
2
Velocity in North direction: 1.2

Velocity in East direction: 0.4

Velocity in Up direction: 0

For time in seconds: 30
```

(a) Move command



```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Co...
Waiting for connection - XXX
Connected!

Client Ver:1 (Min Req:1), Server Ver:1 (Min Req:1)
Followers initialized!
Number of followers: 6
Followers armed!
Cont pos: -10   0       -2
Please choose an option:
1: Takeoff
2: Move swarm by velocity and time
3: Change distance between drones
4: Land
5: Follow the RC drone
6: Task
0: Quit
3
Current distance between drones: 4
New distance:
```

(b) Changing the inter-drone distance

Figure 10: Input for different commands

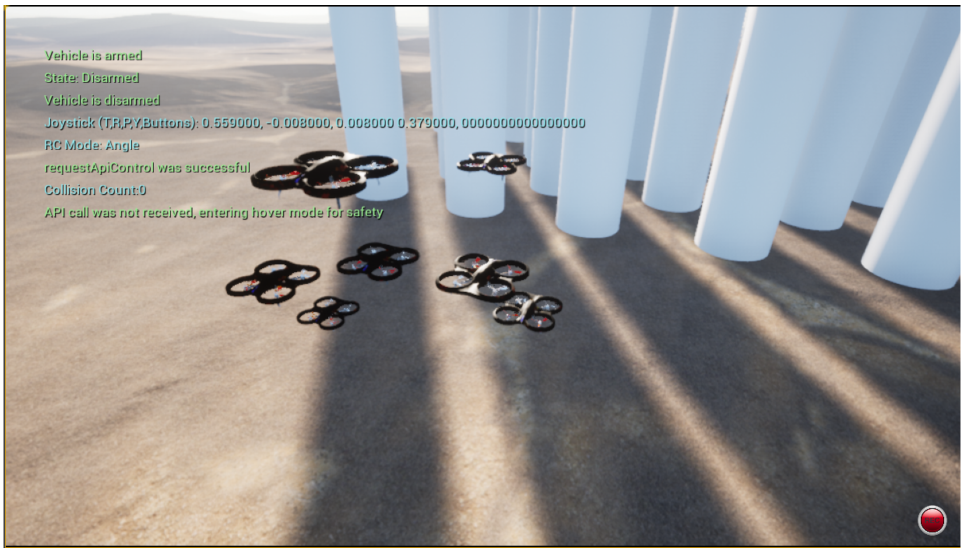
## 3.5 QUAV control

A QUAV is controlled by calling a function provided by AirSim which takes a desired velocity as parameter. The desired control velocity vector $\mathbf{V_c i}$ for the drone $d_i$ is the combination of the repulsion-attraction result from Eq. 3.2 and distance sensor result from Eq. 3.3.

$$\mathbf{V_{ci}} = \mathbf{C_{si}} + \mathbf{C_i} \tag{3.4}$$

## 3.6 Scene setup

The scene is set up with 6 drones in a defined starting position, shown on Fig. 11. Close to the drones, several large cylinders are placed to act as barriers to the drones, spaced out in a way to allow for the drones to fly in between. The formation of the barriers is shown on Fig. 12.
The timer starts and the drones start moving as a swarm in the direction of the obstacles. The drones keep moving until all drones have passed to the other side of the obstacles. The drones then start swarming together and the timer stops when the swarming condition is satisfied.
This process is then repeated 15 times to gather data on average time to complete, standard deviation of the same, and the number of collisions.



```
Vehicle is armed
State: Disarmed
Vehicle is disarmed
Joystick (T,R,P,Y,Buttons): 0.559000, -0.008000, 0.008000 0.379000, 0000000000000000
RC Mode: Angle
requestApiControl was successful
Collision Count:0
API call was not received, entering hover mode for safety
```

Figure 11: Starting formation

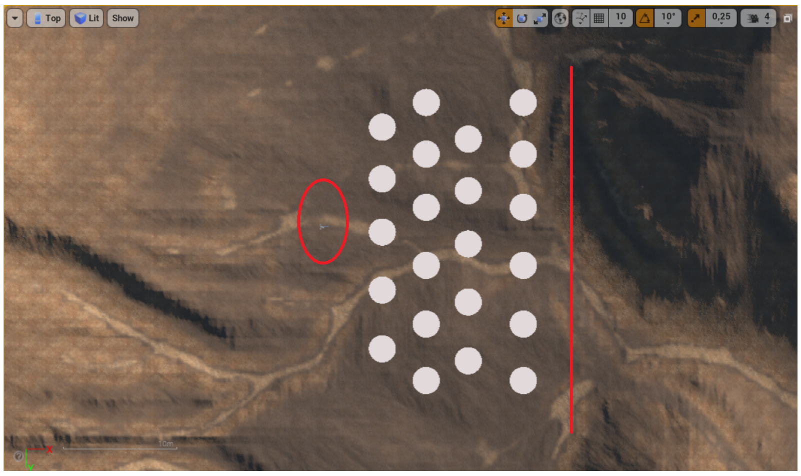


Figure 12: Top-down view of scene setup with barriers (white circles), starting are of the swarm (red circle), and finish line (red line)

## 4 Result

The results of simulations running different parameter sets are shown on table 1 below. Each parameter set was simulated fifteen times. The number of QUAVs for each set was six. The parameters that were changed are sensor activation function, wanted drone velocity, wanted inter-drone distance. Data gathered is average time to complete the task, standard deviation of the same and average collision count per run of the parameter set over the fifteen runs.
Two sensor activation functions are used, in the table denoted as $f_{s1}(S)$ and $f_{s2}(S)$, which refer to Eq. 7 and Eq. 8 respectively.
Some collisions happened during the simulation, all of which were drone against obstacle. In this case, the drone would get stuck for a short amount of time, and eventually break free and continue its journey.

Changing the sensor reaction function to one with larger a domain shown on Fig. 8, yields results shown in Fig. 14.

Keeping everything the same but increasing the velocity by 50% gives results shown on Fig. 15

And finally, changing the target distance between drones by 50% to $6m$ gives results shown in Fig. 16

Screenshots of drones travelling between obstacles are shown on Fig. 17, and after clearing the obstacles on Fig. 18.

Table 1: Simulation results table

| | **Sim 1** | **Sim 2** | **Sim 3** | **Sim 4** |
|---|---|---|---|---|
| **Drones** nr | 6 | 6 | 6 | 6 |
| **Inter-distance** | 4m | 4m | 4m | 6m |
| **Velocity** | 0.8m/s | 0.8m/s | 1.2m/s | 1.2m/s |
| **Sensor Function** | $f_{s1}(S)$ | $f_{s2}(S)$ | $f_{s2}(S)$ | $f_{s2}(S)$ |
| **Avg. task time** | 82.6s | 83.1s | 59.1s | 59.9s |
| **Standard deviation** | 2.75s | 1.49s | 4.05s | 5.92s |
| **Avg. collisions** | 0 | 0 | 0.2 | 0.27 |

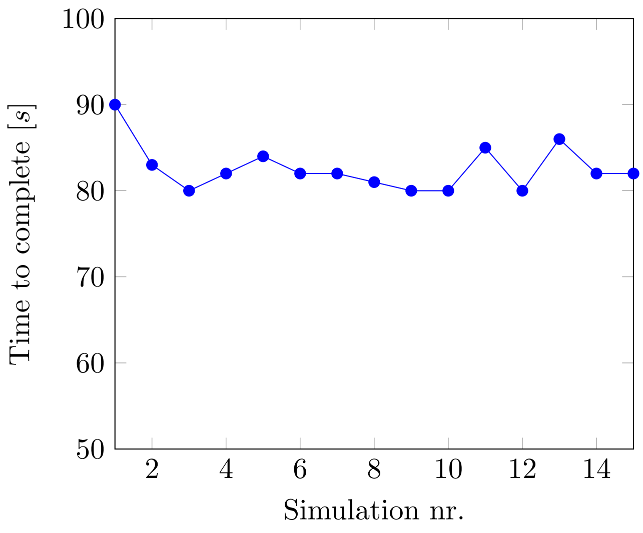


Figure 13: Time to complete for simulation 1

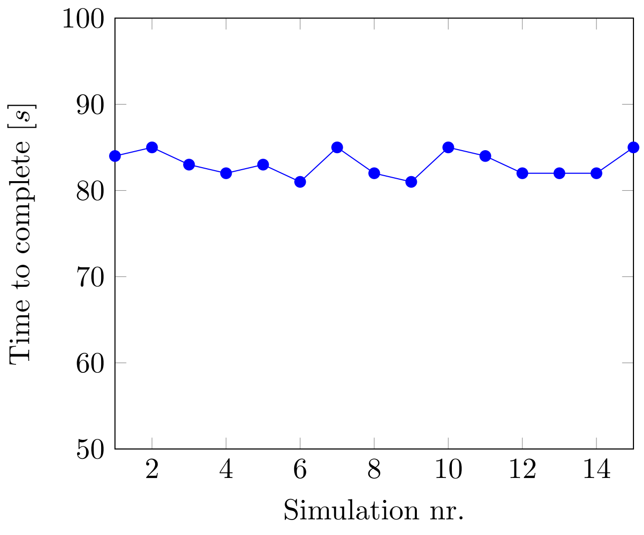


Figure 14: Time to complete for simulation 2

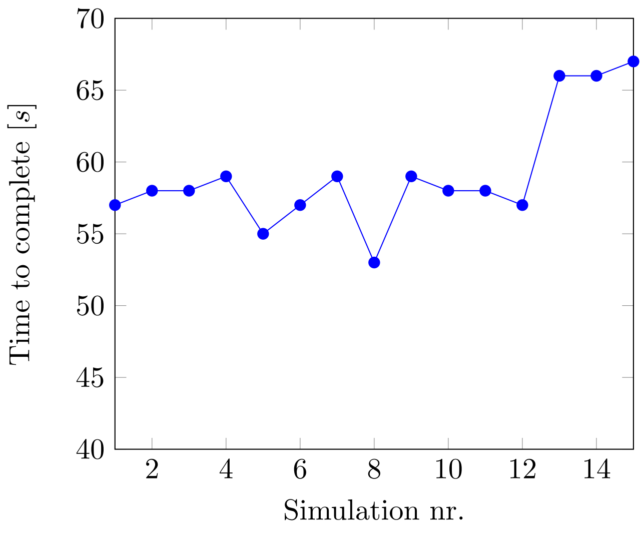


Figure 15: Time to complete for higher velocity, simulation 3

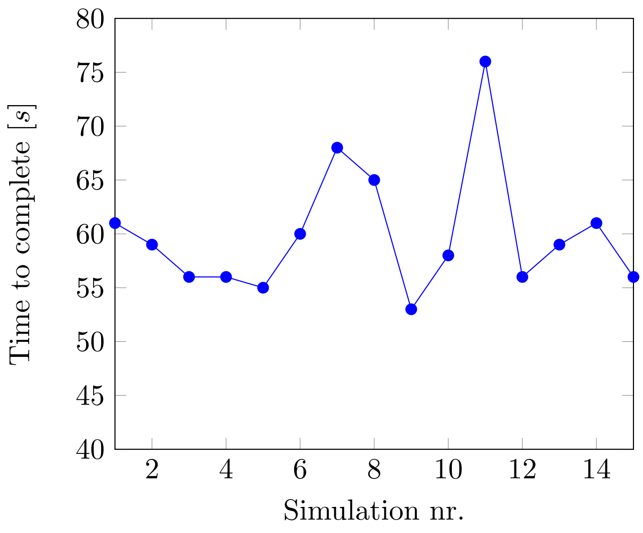
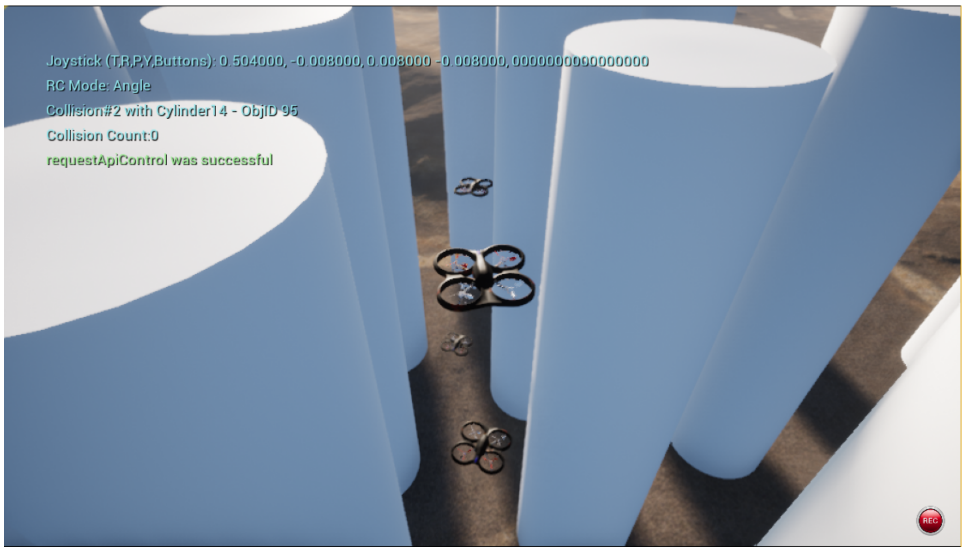
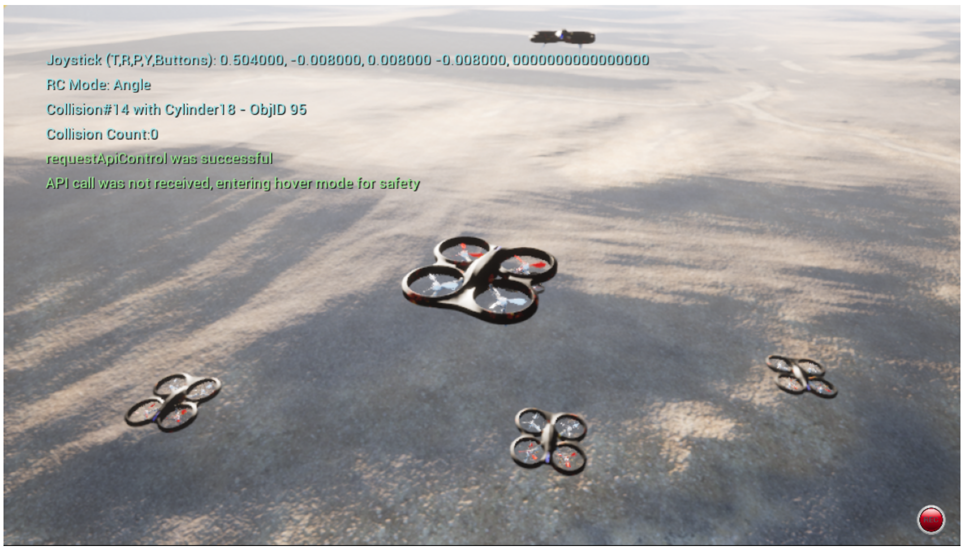


Figure 16: Time to complete for greater distance, simulation 4



```
Joystick (T,R,P,Y,Buttons): 0.504000, -0.008000, 0.008000 -0.008000, 0000000000000000
RC Mode: Angle
Collision#2 with Cylinder14 - ObjID 95
Collision Count:0
requestApiControl was successful
```

Figure 17: Drones flying between obstacles



```
Joystick (T,R,P,Y,Buttons): 0.504000, -0.008000, 0.008000 -0.008000, 0000000000000000
RC Mode: Angle
Collision#14 with Cylinder18 - ObjID 95
Collision Count:0
requestApiControl was successful
API call was not received, entering hover mode for safety
```

Figure 18: Drones after clearing the obstacles

## 5 Discussion

As has been demonstrated, a real-time simulator that is accurate and looks visually appealing is possible. There is, however, a glaring issue that came up during the development of the swarm simulator in AirSim.
The reason that the swarm size was limited to only six QUAVs is because of performance issues. The simulator struggles significantly with a greater number of drones, the framerate drops and the simulator becomes unusable. The results presented in this thesis were generated on a PC with a quad-core AMD Ryzen 3550H CPU, NVIDIA GTX 1650 GPU, and 16GB of dual-channel RAM. Perhaps a machine with more capable hardware could support a larger number of drones and make the simulation run better. This problem could also be due to AirSim not being optimized for receiving a lot of commands in a very short time and controlling multiple agents at the same time.
The simulator itself shows consistent results considering limitations. Changing the sensor activation function to one with a wider domain seems to have made an impact on the consistency of the result. This could be due to QUAVs reacting to the obstacles from farther away and thus having a smoother path in between them.
The change to drones' desired velocity seemed to have made the most impact, which shouldn't come as a surprise. It decreased the average time to complete the task significantly, A change of 50% to the wanted velocity cut the task time by about 29%. However, it came at a price of a few collisions, the wanted velocity combined with repulsion and attraction sometimes overpowered the input given by the distance sensor and a drone would collide

with an obstacle. This in turn resulted in longer runs when a collision happened, causing a higher standard deviation of time to complete the task.
Collisions at higher velocities could be fixed with further modification of the activation functions. A logarithmic function could be worth trying out, so the repulsion from the obstacle could be fine when the drone is far away from the obstacle but rise exponentially as it gets closer to the obstacle, an example is given in Fig. 19. implementing a safety distance as described in [27] is also a viable option.
The change to the swarms wanted inter-drone distance doesn't seem to have made any impact other than a slightly longer time to complete a task. This makes sense because some drones start from farther back when the inter-drone distance is set to a higher number.

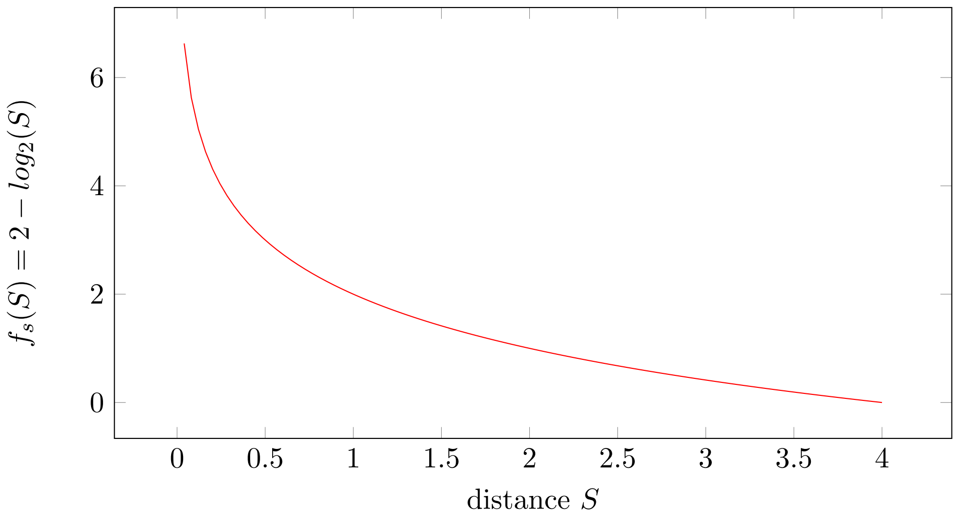


Figure 19: Logarithmic sensor function

The method of input could also be improved, as the current one offers limited possibilities. Compatibility with a controller is supported by AirSim, but it would require some modifications to work for the swarm in this case. The data from the distance sensors for obstacle avoidance is noisy, realistic, and applicable when developing a real drone swarm. However, the technique used to get the distance between drones provides the exact distance given by the simulator. There is room for improvement here, with the possibilities of simulating some existing methods of finding range between drones.

# 6 Conclusion

There is a shortage of good UAV simulators which support multiple UAVs and swarming. There are virtually no simulators that combine high graphical capabilities and multi-drone systems.
The proposed simulator delivers on the task to an extent. It shows consistent results given limited velocity and swarm size parameters. The simulator has been shown to perform well for the size of up to six UAVs.
The drones successfully avoid colliding with each other, but occasionally collide with environmental obstacles, depending on the set parameters. Nevertheless, with or without collisions, the task gets completed and the drones find their way through the cluttered course.
The movement of the drones looks smooth and has high visual quality due to the rendering process provided by Unreal Engine 4. However, this is true only for a limited number of drones, as performance issues arise quickly with the addition of more drones.

## 6.1 Future work

Several things can be improved here. AirSim can be explored more in order to improve performance, different and more efficient swarming algorithms can also be implemented.
The same can be done using other simulators such as Flightmare, which also provides great visual quality. Unity does not require hardware as strong as Unreal Engine 4 does, which could help with performance.
Developing such a simulator using Gazebo or any of its derivatives is also an option, albeit it requires more work, especially for graphics.

## References

[1] G. Chmaj and H. Selvaraj, "Distributed processing applications for UAV/drones: A survey," in *Progress in Systems Engineering*, pp. 449–454, Springer International Publishing, 2015.

[2] A. Bürkle, F. Segor, and M. Kollmann, "Towards autonomous micro UAV swarms," *Journal of Intelligent & Robotic Systems*, vol. 61, pp. 339–353, oct 2010.

[3] "Unmanned aerial vehicle definition." `http://www.oxfordlearnersdictionaries.com/definition/english/uav`. Accessed: 2021-02-15.

[4] "A brief history of us drones." `https://understandingempire.wordpress.com/2-0-a-brief-history-of-u-s-drones`. Accessed: 2021-01-24.

[5] H. V. Abeywickrama, B. A. Jayawickrama, Y. He, and E. Dutkiewicz, "Empirical power consumption model for UAVs," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, IEEE, aug 2018.

[6] J. Banks, J. Carson, B. Nelson, and D. Nicol, *Discrete-Event System Simulation*. Prentice Hall, 5 ed., 2010.

[7] D. Goldsman, R. Nance, and J. Wilson, "A brief history of simulation," pp. 310 – 313, 01 2010.

[8] "Dji flight simulator." `https://www.dji.com/hr/simulator?ite=brandsite&from=nav`. Accessed: 2021-02-15.

[9] "Zephyr simulator." `https://zephyr-sim.com`. Accessed: 2021-02-15.

[10] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: a matlab drone swarm simulator," May 2020.

[11] P. Cybulski, "A framework for autonomous UAV swarm behavior simulation," in *Proceedings of the 2019 Federated Conference on Computer Science and Information Systems*, IEEE, sep 2019.

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, IEEE, 2004.

[13] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: MORSE," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, may 2011.

[14] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," 2020.

[15] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," May 2017.

[16] T. Vicsek and A. Zafeiris, "Collective motion," *Physics Reports*, vol. 517, pp. 71–140, aug 2012.

[17] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, aug 1987.

[18] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Data from: Optimized flocking of autonomous drones in confined environments," 2019.

[19] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, pp. 1–41, jan 2013.

[20] C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, and T. Vicsek, "Flocking algorithm for autonomous flying robots," *Bioinspiration & Biomimetics*, vol. 9, p. 025012, may 2014.

[21] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, p. 500, feb 2018.

[22] E. Coumans, "Bullet physics library." `bulletphysics.org`.

[23] R. Smith, "Open dynamics engine." `http://www.ode.org/index.html`.

[24] "Simbody api reference documentation." `https://simbody.github.io/3.5.0/index.html`. Accessed: 2021-05-11.

[25] Epic Games, "Unreal engine."

[26] "Unreal engine documentation." `https://docs.unrealengine.com/en-US/index.html`. Accessed: 2021-05-11.

[27] E. Falomir, S. Chaumette, and G. Guerrini, "Mobility strategies based on virtual forces for swarms of autonomous UAVs in constrained environments," in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*, SCITEPRESS, 2017.

[28] V. Gazi and K. Passino, "A class of attraction/repulsion functions for stable swarm aggregations," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, IEEE, 2002.

[29] “Iso international standard iso/iec 14882:2020(e) – programming language c++.” https://isocpp.org/std/the-standard. Accessed: 2021-05-11.

## A Digital Attachment

The digital attachment includes all the source code for the drone swarm project, a build for running it and a readme file explaining the setup.