



University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Tello Camp – implementation of a drone simulation in Unity

IP5-IIT09 (HS 2021)

Authors:

Maarten de Laat
Pascal Hostettler

Supervisors:

Prof. Dr. Hilko Cords
Dr. Dieter Holz

School of Engineering University of Applied Sciences and Arts Northwestern
Switzerland (FHNW) Brugg-Windisch

Institute for Interactive Technologies (IIT)

January 21, 2022

Abstract

As a fifth semester project at the FHNW university of applied science, a software is created to simulate every aspect of the Tello Drone virtually.

The Tello Drone is a drone from Ryzerobotics which is used for educational purposes in coding.

A simulation of this drone has been created in a previous project. It simulates the API of the drone which makes it great for learning to program the drone. The environment is a glass cube, which greatly reduces the immersion. This new project will offer a greater immersion and a realistic drone flight simulation which will allow students to learn how to fly the drone with the software, instead of the real Tello Drone.

The result is a game, called Tello Camp, developed with the Unity game-engine, which precisely simulates the behaviour and API of the Tello Drone. It helps students learn how to program and fly the drone without the cost and risks of a real drone.

Table of Contents

1	Introduction	1
2	Background	2
2.1	Basic quadcopter physics	2
2.2	Tello Drone	4
2.3	Tello SDK	5
2.4	Tello Simulator	7
2.5	DJI Flight Simulator	9
2.6	Game Engine	10
2.7	Quaternions	12
2.8	UDP – User Datagram Protocol	13
2.9	Finite State Machine	13
3	Concept	15
3.1	Requirements	15
3.1.1	Programming education	15
3.1.2	Realistic behaviour	15
3.1.3	Enhanced 3D-World	15
3.1.4	Skill training	15
3.2	Tech Stack Decision	16
3.3	Movement	16
3.4	Rotation	19
3.5	Flying states in Tello Camp	21
3.6	Automatic balancing system	21
3.7	State Machine	22
3.8	UDP Controller	24
3.8.1	Sockets	24
3.8.2	Parsing a message	25
3.9	User Interface	25
3.9.1	Menus & Settings	26
3.9.2	Heads-up Display	26
3.10	3D-World	27
4	Implementation	29

4.1	Architecture	29
4.1.1	Class Diagram	29
4.2	Drone	29
4.2.1	Main Class	30
4.2.2	State machine	31
4.3	Controller	31
4.3.1	UDP Controller	31
4.3.2	Test Controller	34
4.4	Flying in Tello Camp	35
4.4.1	FlyingState	35
4.4.2	AutopilotState	37
4.5	Automatic balance system	39
4.6	Tilt	39
4.7	User Interface	41
4.7.1	Main Menu	41
4.7.2	Pause Menu	42
4.7.3	Heads-up Display	43
4.8	3D-World	46
4.8.1	Drone	46
4.8.2	Surroundings	47
4.8.3	Playground	48
4.8.4	Render Pipeline & Post Processing	50
5	Discussion	51
5.1	Value of Tello Camp	51
5.2	Technical challenges and opportunities	51
5.2.1	Video Stream	51
5.2.2	Quaternions	51
5.3	Outlook	52
6	Conclusion	53
	List of Figures	57
I.	Appendices	59
A.	Git Repository	59
A.1	Contents	59

A.1.1	Documentation	59
A.1.2	Builds	59
A.1.3	Unity Project	59

1 Introduction

In some computer science modules at the FHNW, the professors are using physical drones to make the lectures more practical and vivid. In STEM education, programmable robotics are always expensive and usually there are not enough for every student. This means the classes have to be rather small, so the students don't have to wait for a free drone to be available for testing. Using a real drone has a few disadvantages for development: they are expensive, break easily and need to be recharged after a short time of testing.

In this project, a software application is implemented. It simulates the behaviour of the drone and its API. The drones used by the FHNW are Tello Drones, easily programmable and for which the students create their own controller in form of a smartphone application.

In this project, we investigate what is necessary in order to create a realistic simulation of the drone's real-world behaviour and replicate the API which is used to control it. We will explore different concepts of simulating a drone with software and what solutions already exist. Finally, a prototype of a software to simulate a Tello drone in Unity is implemented and verified.

The software solution, called "Tello Camp", lets students program and fly a virtual drone. The virtual drone flies the same way as the real one. This allows students to train in a virtual environment without risking damaging the real one, saving time and cost.

2 Background

In this chapter, we explore the physics behind how autonomous quadcopters can fly and navigate. The Tello drone, which is used by the FHNW in classes, is introduced and its communication features are elaborated. Next, we will explore which software simulations already exist for drones and what software framework is most suited for the implementation of this project.

2.1 Basic quadcopter physics

A quadcopter, also called a drone, is a flying vehicle with four rotors. Most modern quadcopters are autonomus, that means they are able to stabilize itself and hover in mid-air. To achive this, a combination of accelerometers, gyroscopes and visual positioning system is used by the flight controller to control the motor speeds.

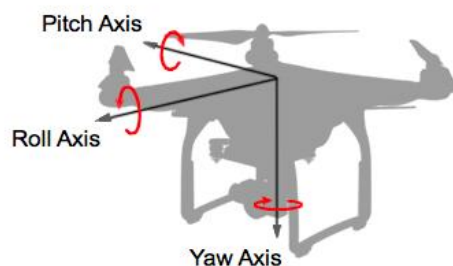


Figure 2.1: 3 Axis of the drone [1]

To achieve all six degrees of freedom for flight, the drone must be able to create force in three axes: Pitch, Roll and Yaw (Fig. 2.2)

To lift the drone from the ground, the rotors create a lift force that overcomes the force of gravity on the drone. This brings the drone up in the air. Once the desired height is reached, the rotors are slowed down until the lift force balances the weight of the drone. To make this happen, the speed of all 4 rotors is adjusted proportionally.

However, if all rotors would turn in the same direction, precession force would make the drone start rotating around its own yaw axis. To counteract this, 2 rotors are turning clockwise and two are turning counterclockwise to even out the torque.

Exactly this mechanism is also used to adjust the drone's rotation around the yaw axis. By turning the clockwise rotors faster than the counterclockwise rotors, the quadcopter starts to turn around the yaw axis counterclockwise ("Rotate left" Figure 2.2: How four rotors control the drone).

To adjust Pitch and Roll, two pairs of rotors on the same side rotate faster than the opposite ones. The higher lift force on that side brings the aircraft to pitch or roll. [2]

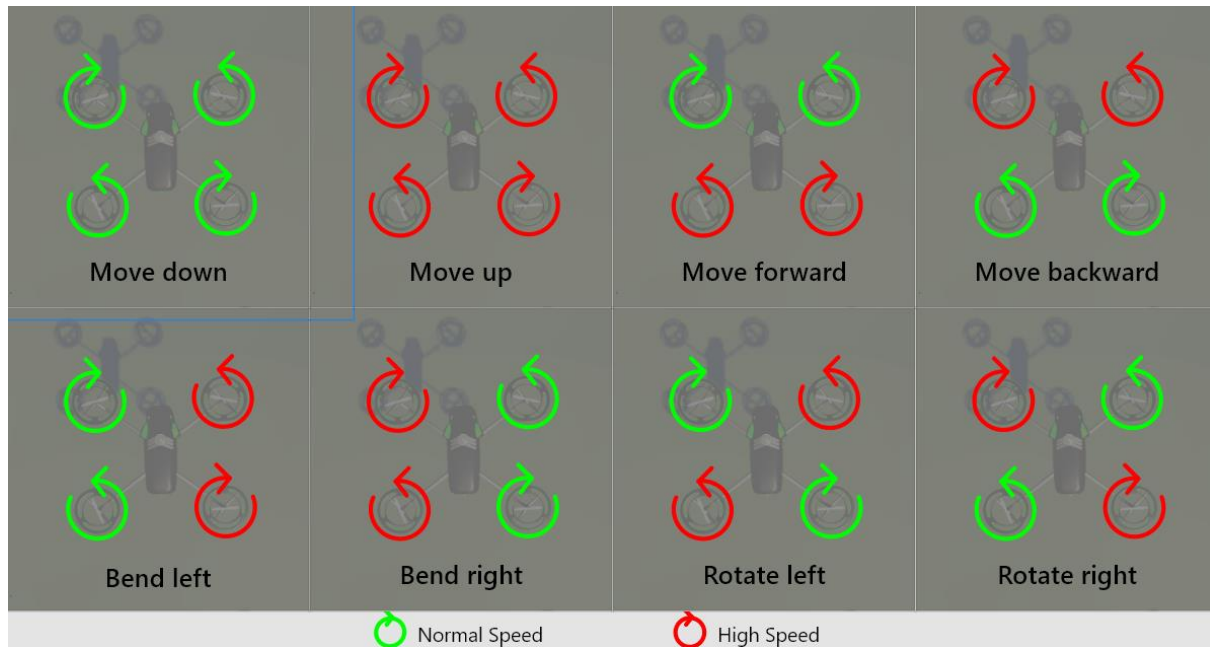


Figure 2.2: How four rotors control the drone

Let's assume we want to move the aircraft forward. We adjust the pitch level by rotating the rotors in the back faster and the rotors in front slower. The total lift force is still the same and balances the drone weight, to stay at the same height. However, the lift force is not anymore vertical to the ground, which leads to the aircraft being pushed forward. [2]

It is practically impossible to achieve such a behaviour without a computer processing the positional data of the drone hundreds of times a second and adjusting the speed of the rotors to correct for outside influences. Otherwise, any slight gust of wind would make the drone tumble.

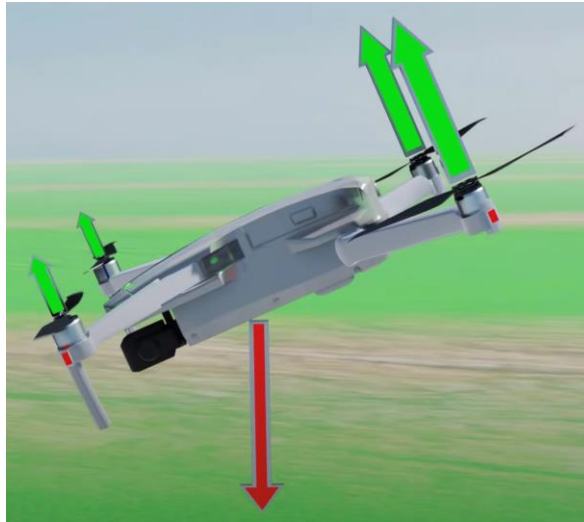


Figure 2.3: Forces of a flying drone [3]

2.2 Tello Drone

The Tello Drone by Ryze Robotics is a small, programmable drone built for educational purposes [4]. The drone has a camera which can send its video stream to a controller (i.e a smartphone app). This drone is also used in some courses at FHNW. In this section, we analyze how the drone behaves in the air with the goal to be able to replicate a similar behaviour. We also investigate its Software Development Kit (SDK) to create an overview of the drone's programmable capabilities.



Figure 2.4: Tello Drone [5]

A physical drone is expensive, easily breaks and can only fly for 13 minutes at one time before it needs to be recharged. Our project aims to solve these shortcomings by creating a virtual replica of its behaviour in software.

2.3 Tello SDK

To control the Tello Drone, one can use the Tello app available for iOS and Android. The big advantage of the Tello Drone is that it also offers a very simple SDK [6] to program the drone and control it by a self-programmed controller. The FHNW is using the Tello SDK to program the drone in their lectures.

The Tello Drone uses Wi-Fi to connect to another device. It uses the User Datagram Protocol (UDP, see section 2.8) to send and receive data on 3 different channels:

Send commands and receive responses

The SDK allows to send commands to control the drone. These commands are plain text commands, which are acknowledged with an “ok” text back from the drone.

The SDK differences between 3 types of commands:

Control commands:

Are used to send a command like “takeoff” which lifts the drone up in the air. These commands can also be used to automate the drone with commands like “forward 50” which moves the drone 50cm forward.

The drone returns “ok” if the command was successful or “error” if the command failed.

Set commands:

Set commands will set parameters about the drone’s behaviour. For example, the command “speed 50” will limit the drone’s maximum speed to 50cm/s.

The drone returns “ok” if the command was successful or “error” if the command failed.

Read commands

Read commands can be used to ask the drone for information about its status. For example, the command “battery?” causes the drone to send back the current battery level.

The drone returns the requested value.

Receive Tello state

The Tello Drone sends state updates every 100ms through a separate UDP socket. This message contains information about the drone’s current rotation, speed, height and more. It is one data string, separated by a semicolon.

Receive Tello Video Stream

The Tello SDK also gives access to the Tello Drones front facing camera. It is send out through another UDP socket.

2.4 Tello Simulator

Tello simulator is a software application written in JavaFX which was created in a previous student project. It is currently used in classes to test applications written by students to control a physical drone. It allows them to connect to the drone simulator via a UDP port and fly the virtual drone with Tello commands. Its environment is a glass cube in which the drone is contained.

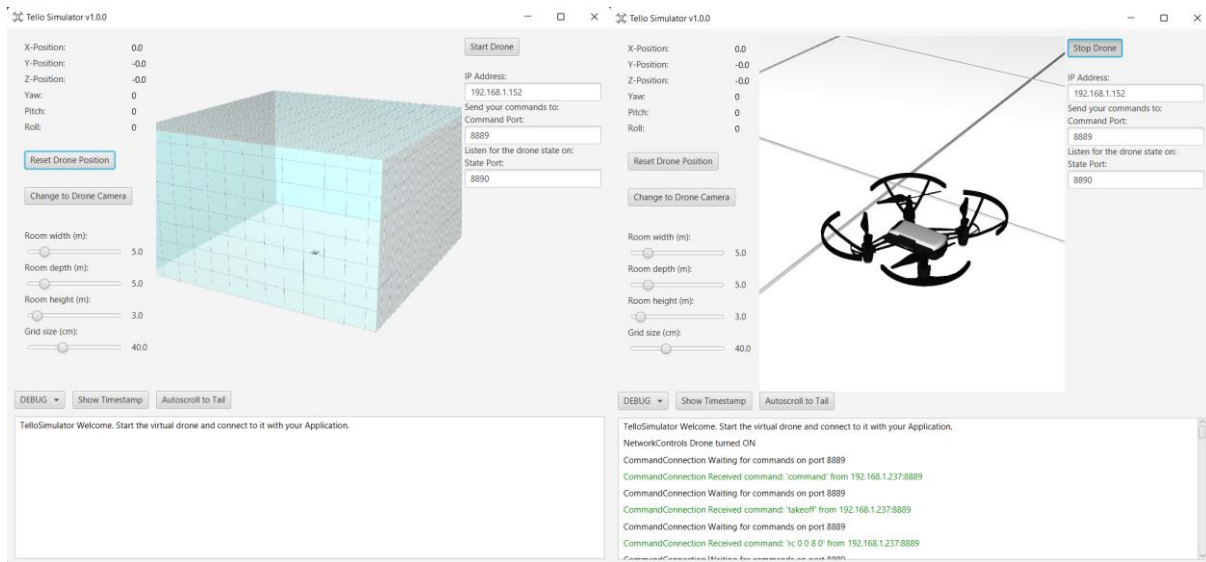


Figure 2.5: Tello Simulator screenshots

Using this software allows for students to learn how to connect to the drone and control it with commands. This software feels like a development kit only used for creating a controller which can steer the drone. It has many great features the new simulation will keep:

1. It shows the IP address and UDP ports to connect to. It is also possible to change them, which is very important because 20 students on the same network cannot use the same ports
2. It has 2 views: one is standing and looking at the drone, the other is from the drone's camera
3. It is possible to reset the drone, for when mistakes happen
4. A textbox which shows the last commands, which is great for debugging and programming the drone

Our project will keep these features and provide a few more for a more immersive and complete simulation:

- A full screen view for immersive steering
- A drone behaviour very close to the real drone, so that the experience of flying is nearly indistinguishable between the two
- A more realistic environment with trees and obstacles to fly around

2.5 DJI Flight Simulator

DJI is not only the manufacturer of Tello but also of other small consumer drones and big commercially used drones. DJI created the DJI Flight Simulator [7].with the goal to train drone pilots.

The DJI Flight simulator offers a huge variety of different drone models to choose from. If the user has a supported DJI drone controller, it can easily be connected and be used to explore the different available worlds. Those worlds come in realistic graphics, physics and even different wind and weather conditions, to make the simulation as realistic as possible

The DJI Flight Simulator gives you 3 different training modules:

Skills training

This gives the user the possibility to learn basic skills such as lift-off, flight route training, and hovering.

Free Flight

Gives the possibility to explore different worlds and environments to practice.

Application training

The application training offers different scenarios for commercial drones, for example, a sequence on how to do a powerline inspection.

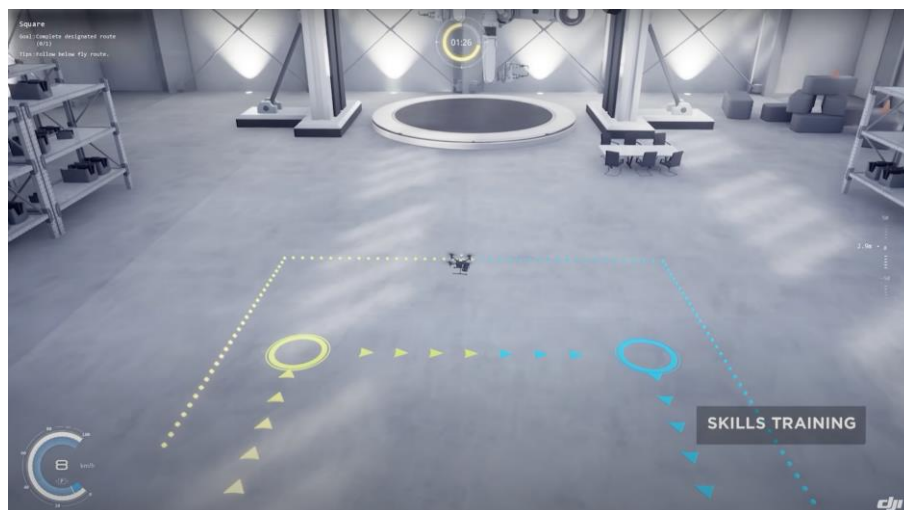


Figure 2.6: Screenshot of the DJI Flight Simulator [7]

The DJI Flight Simulator is a powerful and super realistic tool to learn how to fly a DJI drone. However, it doesn't have the needed SDK interface to use for programming education. Useful for our project is to see that the DJI Flight Simulator have a skills training section where the user needs to fly a path, and a free flight mode to explore the world.

2.6 Game Engine

For this simulation to feel real, many physical operations need to be simulated in a well way. There exist many technologies and software on the market which are proficient in creating a 3D-worlds. A very popular solution for this sort of problem are game engines. Game engine is a software which specialises in creating a game, which can be two or three dimensional. Unity is such a game engine which is very beginner friendly and includes physical operations out of the box.

Unity

Unity is a cross-platform game engine developed by Unity Technologies. It is a tool for creating games for many existing platforms like iOS, Android, Windows, macOS, Linux, PlayStation, and Xbox. Unity supports the creation of many different sorts of games. It is possible to make 2-Dimensional and 3-Dimensional, as well as Virtual- and Augmented Reality games.

The game engine is very friendly to newcomers, has an incredibly vast user-base, countless tutorials available on the internet and is extremely modular thanks to its components-based system. This allows for a steep learning curve and freedom of creativity for this project. Unity has a store with a large amount of assets which can be imported directly into a project, that permits more time to be invested into the realistic experience for the user, instead of 3-D modelling objects.

In this project, Unity will be used to create a 3-Dimensional game, in which the user can control the drone via the Tello SDK.

Game Loop

It is important to understand the unity game loop and how the principles behind it work, to make an object look like its moving.

One way to add custom behaviour to an Object in Unity are scripts. Lines of code written in C#¹, which can access Unity frameworks and manipulate components. Scripts are modular components attached to add functionality or behaviour to a GameObject.

A script in unity can implement countless available interfaces, and the one we use in this project for physical operations is called “MonoBehaviour”.

Every script that implements MonoBehaviour uses three main functions of unity. These are called Start(), Update() and FixedUpdate().

¹ <https://docs.microsoft.com/en-us/dotnet/csharp/>

Start()

This function is the initialiser. It is the very first function that is called when the script is enabled [8]. Usually, that happens as soon as the GameObject is created where the script is attached to.

Update()

This is the main loop in Unity. Update() is the function that is called every single frame. This means, if the game runs at 60 frames per second, Update() is called 60 times in one second. Update() should not be used for physics-related operations, because the framerate of a game is highly unstable. Depending on the Computer, it may run at way higher or lower frames per second.

FixedUpdate()

This function is called at a fixed rate based on time [8]. By default, it is called every 0.02 seconds (50 times per second). This makes sure, no matter on what hardware a game runs on, the physics-related operations are always calculated at the same rate, creating the same experience across different hardware.

Rigidbody

The Rigidbody is a component in Unity, which puts its motion under the control of Unity's physics engine [9]. By accessing the component force and torque can be added to the object, its velocity can be read, meaning how fast its moving or rotating and other properties related to its simulated physics can be controlled. It simplifies the implementation of physics-related behaviours because many physical laws and behaviours (like newton's first law) are Simulated by the game engine and can be accessed through this component.

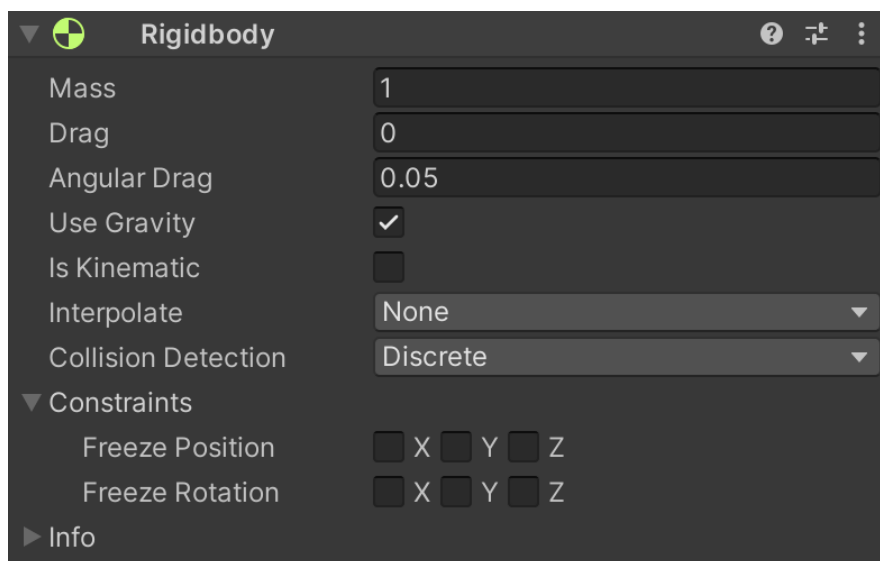


Figure 2.7: Rigidbody component in Unity

2.7 Quaternions

Most 3D game engines base the calculations of rotations on quaternions. The exact mathematical explanation of how quaternions work exceeds scope of this project, however, this section will explain their unique advantages and counterintuitive behaviours. The most practical way of rotating a vector in unity is by multiplying it by a quaternion.

Euler angles greatly lose accuracy in its ability to describe rotations when it is about 3-Dimensional spaces. Quaternions also don't suffer from Gimbal-lock, which is the loss of a degree of freedom because one or more rotational axes overlap. [10]

A (Euler) angle is measured in either degrees or radians. Degrees range from 0 - 360, radians range from 0- 2π . For the sake of familiarity, this explanation will use degrees. 0° means no rotation, and 360° means one complete revolution. 720° describes 2 complete revolutions etc. Using Euler angles, it is possible to describe any amount of turns in any direction (to the left or to the right). The number of degrees to turn is between $-\infty$ to $+\infty$.

Rotating with Quaternions cannot do this in one step. Quaternions always describe the shortest path to the desired angle. If the values of quaternions were converted to degrees, it would mean that a rotation can only have values between 180° and -180° , where these represent exactly one-half rotation to the right and respectively to the left. Due to the nature of quaternions, it is impossible to do a 270° turn, because the shortest path is a 90° turn in the opposite direction. [11]

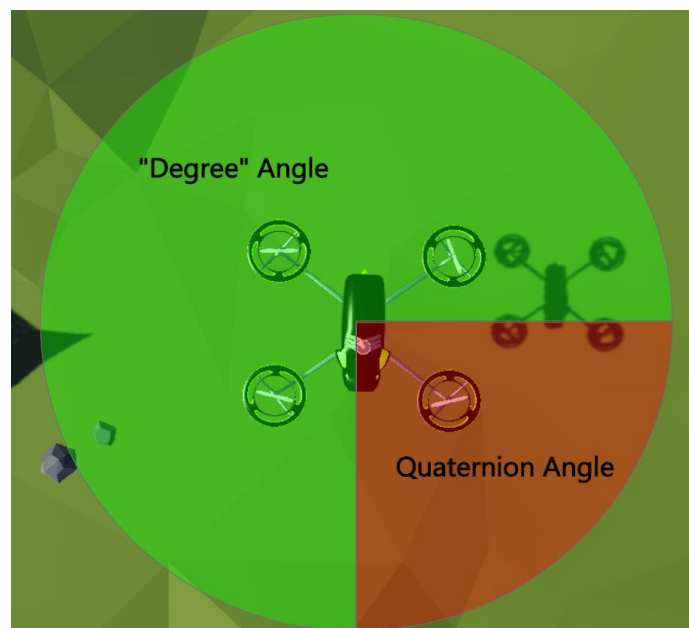


Figure 2.8: How Quaternions interpret 270°

2.8 UDP – User Datagram Protocol

The User Datagram Protocol (UDP) is a transport layer networking protocol which prioritizes speed over safety. Data packets are sent over the Internet Protocol to a certain UDP port without any form of “handshake”, meaning the receiver does not confirm the reception of the message. UDP also does not check whether a data package is missing or corrupted. If a packet is lost or not transmitted completely, it cannot be recovered and must be sent again. UDP is for these reasons much faster, because these safety checks are not provided. [12]

2.9 Finite State Machine

A finite state machine is a computing machine that has a fixed set of possible states, a set of inputs that change the state, and a set of possible outputs. [13]

A finite state machine allows for the same input to have different outcomes depending on which state the machine is in. This enables a machine to have specific behaviours depending on the state, all while using the same inputs.

Example: A light switch has one input. Depending on the state of the light (on/off), pressing the switch either turns the light on or off. The same input causes different actions.

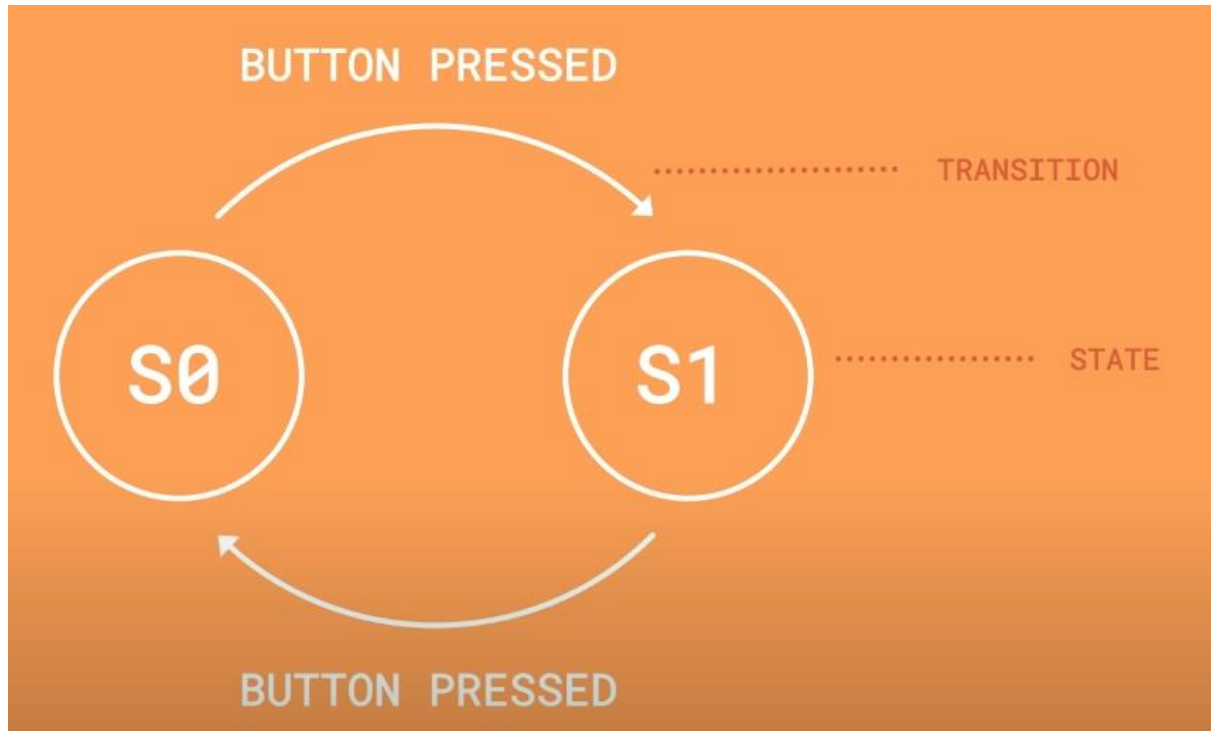


Figure 2.9: example of a state diagram [13]

In a visual representation of a state machine, a state diagram, the states are encircled, while the transition from one state of another is an arrow. The arrow is labelled with the action that causes the transition.

The reason why state machines are vastly used in software engineering is that they help in structuring the project. Implementing different behaviours on the same input can lead to huge if statements without a state machine. However, once the state machine is in place, it is easy to react on inputs depending on states.

3 Concept

This chapter explains the concepts used in the project and the decision-making process as to why these concepts greatly meet the requirements.

3.1 Requirements

There are 4 main requirements Tello Camp needs to fulfil to become usable in the lectures at the FHNW and therefore being a successful project:

3.1.1 Programming education

The Tello Drones available at the FHNW are primarily used as an educational tool. As described in the Chapter about the Tello Drone, it has some disadvantages, like its price and availability. Tello Camp should have similar programming capabilities as the Tello Drone and the old Tello Simulator have. Therefore, the goal of Tello Camp is to offer the same programming interface as the real Tello Drone.

Another criteria is, that Tello Camp should run on Windows and macOS, so most of the students can use it. Ideally it would also run on Linux.

3.1.2 Realistic behaviour

The previous Tello Simulator allows students to learn how to program, however it is not a realistic implementation of a drone (see section 2.4). Tello Camp should look and feel realistic.

3.1.3 Enhanced 3D-World

Tello Simulator simulates drones' behaviour inside a white box. Tello Camp should offer an aesthetically pleasing 3D-World, where it is enjoyable to simulate a drone flight in.

3.1.4 Skill training

As the old Tello Simulator does not provide a realistic flying environment, it cannot accurately represent flying a real Tello Drone. It is important for the students to become skilled flyers and be able to assess how the drone may react to natural circumstances, as real drones break easily. A limited training course is to be implemented, to train new drone pilots.

3.2 Tech Stack Decision

There are a few different ways to fulfil the given requirements. One option would be to extend the old Tello Simulator with the required functionality. Physics and a good-looking 3D-world would have to be implemented using JavaFX. That would take a lot of time and effort and JavaFX isn't the best solution to build such an application.

A better way is using a Game engine. Most game engines offer realistic physics right out of the box and give the option to use 3D-models to create a world. Creating a realistic drone simulation, limited by the laws of physics, in an enhanced 3D-World with a training course wouldn't be an issue. Implementing the UDP connection shouldn't be a problem as well, as the .NET Framework, which is included in Unity, already provides standard socket implementations [14].

We decided to use Unity as our platform to build Tello Camp for several reasons:

Multiple platforms

Unity supports building for Windows, macOS, Linux and even mobile platforms. This is great to offer support for as many students as possible.

Unity Asset Store

Unity has an asset store, where 3D-Models can be downloaded and easily imported into the project.

Beginner friendly

Unity has a huge community and a lot of guides available for beginners. The FHNW also offers classes in Unity.

3.3 Movement

Simulating the movement of the drone can be done in many ways. In this section we have a look at different ways of controlling a drone and explain which option is the most useful for our project.

Ignoring Physics

A simple way would be to have the drone float in space and ignore gravity. This means the force a drone applies with its rotors in order to counter the gravity force of the earth does not need to be simulated and calculated. To move the drone in a certain direction, a simple change of its position vector in each frame would suffice. It would "teleport" a tiny bit each frame, like a stop motion movie. The drone would appear to be flying and the user can learn to operate the drone. A challenge in this solution would be to fine-tune the drone's movement to the real one.

Exact replica of a real drone

A more advanced implementation could be simulating all physical forces applied to the real drone as described in section 2.1.

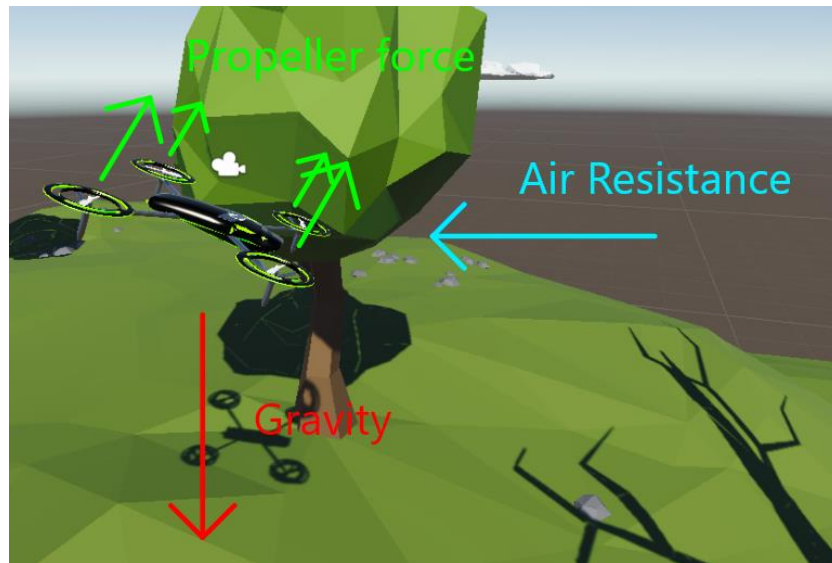


Figure 3.1: different forces acting on the drone

When increasing the force of the rotor in the back, the drone would be tilted to the front. By doing that, a portion of the rotor force (the x component in a coordinate system) will be used to push the drone forward. If this force is larger than the air resistance, the drone will accelerate and increase in speed. If the force of the air resistance is larger, the drone will brake and slow down. If they are the same, the drone will stay at its current velocity and will have reached its maximum speed. Adjusting the drone's movement speed and acceleration would be achieved by changing the maximum force the rotors can provide and the weight of the drone.

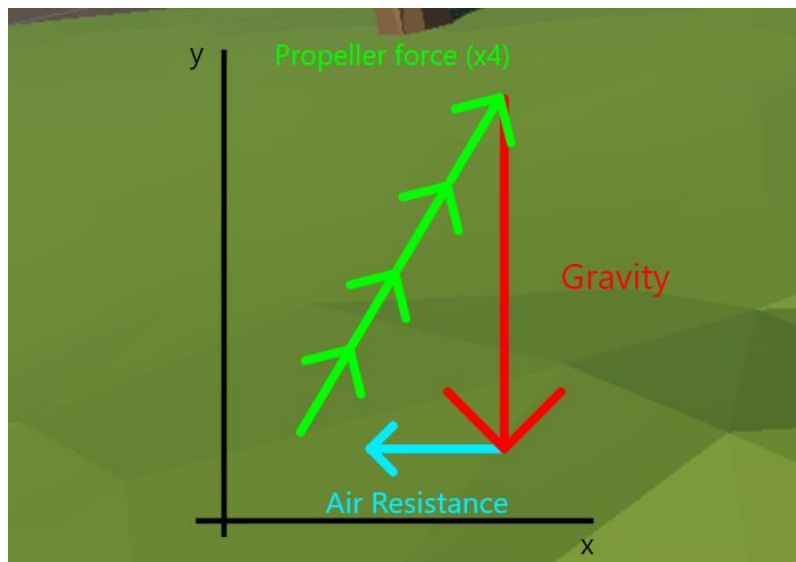


Figure 3.2: Brief calculation of the forces in Figure 3.2

Using a limited physics simulation

A third way would be a combination of those two extremes. Many laws of physics that influence the movement of the drone can be simulated using a Rigidbody. Gravity can be ignored, so the drone doesn't need to apply force on the rotors all the time. That way, force only needs to be applied in the direction of the movement of the entire drone. In this simplified simulation, the drone doesn't tilt when moving, which makes it look very unrealistic. However, this can be solved by changing the tilt of the drone model manually to appear realistic in the 3d view. This option is the best fit for our project.

Tilt

A simple and very effective way to simulate the tilt is to map the velocity of the drone to a value between $0 - 25^\circ$. When the drone is flying at maximum speed, it will tilt 25° in the corresponding direction.

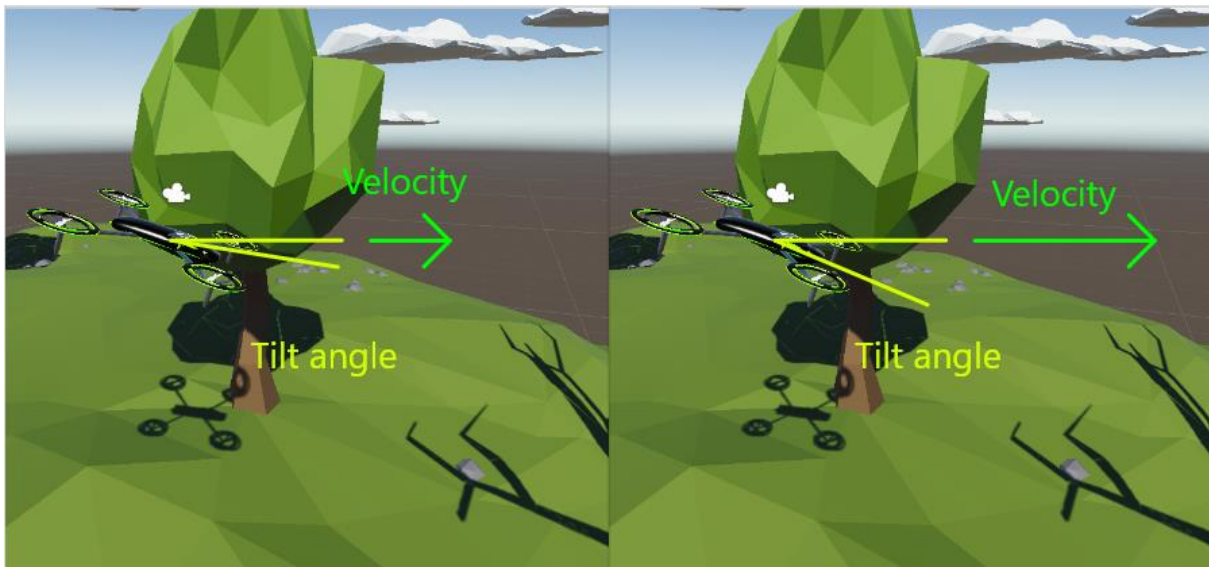


Figure 3.3: Different velocity require a different tilt angle

In order to tilt the drone in the correct direction, the vector to rotate around must be calculated first. The drone is always inclined in the direction of its movement, which is its velocity vector. Inclining in the direction of movement is the same as rotating around the vector / axis that is perpendicular to its velocity vector.

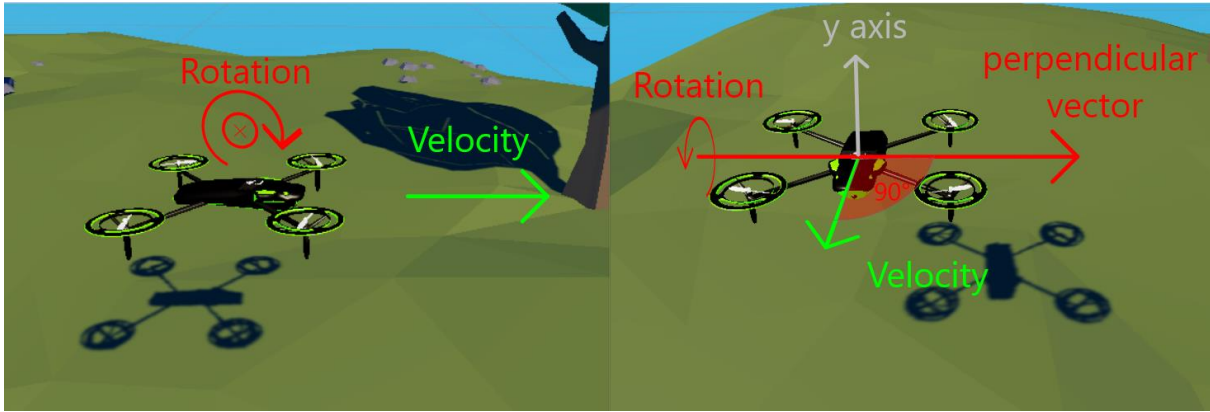


Figure 3.4: Visual display of the axis to rotate around

Calculating the vector to rotate around can be done with the cross product of two vectors. Two vectors always span a plane (unless they are identical). The cross product of two vectors is always perpendicular to said plane. If the right plane is found, the perpendicular vector is easily calculated. Since the drone only tilts when moving horizontally, the right plane is the one spread by the velocity vector and the y-axis.

$$V_p = V_v \times Y_{axis}$$

V_p = perpendicular vector, V_v = velocity vector, Y_{axis} = vector along the y axis

Now that the vector to rotate around has been found, a mapping of the speed and the rotation of the drone will create a realistic tilt, which gives the impression the drone is moving because of its tilt.

3.4 Rotation

Tello SDK allows for a command which lets the drone rotate between 0–360° at once. As described in the section about Quaternions (2.7), anything above 180° turns the rotation in the opposite direction. 270° turn into a -90° turn etc.

Two different solutions will be explored in this chapter:

Calculating Time

The first is to calculate the amount of time needed to make the rotation, and then let the drone rotate in this direction.

Example: The drone turning speed is 90° per second. Let's say a turn of 270° is requested. The solution would be to let the drone rotate for 3 seconds because 3 times 90° equals 270°.

Splitting the angle

Another solution that would avoid the limits imposed by using Quaternions is to split each movement into multiple smaller movements which do not extend beyond 180° . This means the angle needs to be split up inside the simulation. According to the SDK, the angle to rotate cannot be larger than 360° . If every angle is split into 3 separate movements, then the simulation would never rotate more than 120° at once. This is desirable because the Quaternion would always interpret the angle in the correct direction, since 120° is smaller than 180° .

Example: Once again, we rotate the drone by 270° . Now we split it by three, which equals 90° . Now we rotate three times 90° , and we successfully rotated the desired amount. This lets us use in-house methods for rotation in unity.

Splitting the angle is the better solution than calculating the time, because it is much more accurate. The error margin with this method is within 0.1%.

The other method is more difficult because the movement speed of the drone is not constant. When rotating, the drone accelerates to its maximum angular velocity, rotates at constant speed, and slows down until it stops. Because it is not a constant speed, it becomes very hard to precisely predict the amount of time necessary for a rotation. The error margin for this method is about 1%, which we deemed too high compared to the alternative.

3.5 Flying states in Tello Camp

The drone has two different states in which it can fly. One is the FlyingState, which is accepting “RC commands”, the other state is AutopilotState, which is used for moving fixed distances or rotating a fixed amount. Gravity is turned off for the drone as soon as it starts flying and all movements are done by adding forces. Which means, Newton's first Law applies.

$$f = ma \rightarrow a = \frac{f}{m}$$

In Unity, there are 3 factors which influence the impact force has on our drone. These factors are:

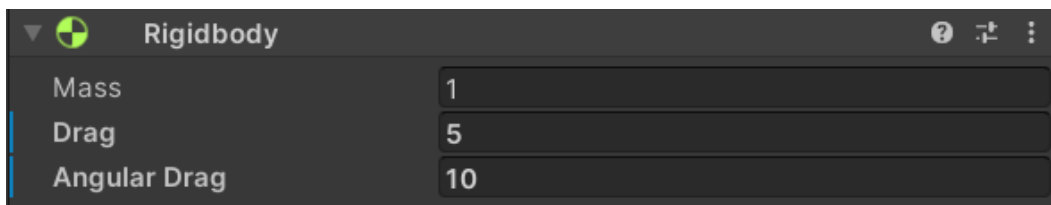


Figure 3.5: cut-out of the Rigidbody component on the drone

Mass

The heavier the object, the more force is necessary to achieve a certain acceleration

Drag

Can be interpreted as air resistance. Drag slows down a moving object by a certain percentage of its velocity. The faster the rigidbody is, the stronger it is slowing down.

Angular Drag

The same thing as drag, but for rotations.

3.6 Automatic balancing system

A feature we added is the automatic balance system. If the drone collides with an object, it loses its balance and might fall upside down. As soon as the real drone collides with a wall or really anything, it goes into stop mode and lands on the ground. The purpose of our project is to learn how to program and how to fly the drone. If every little mistake caused the drone to land, then the user experience of a beginner would be very disruptive.

Without any automatic balance system in place, the drone would be tilted by any collision and stay in that angle. This would break the illusion of the drone simulation, as the drone wouldn't move anymore in the direction its rotors are facing.

3.7 State Machine

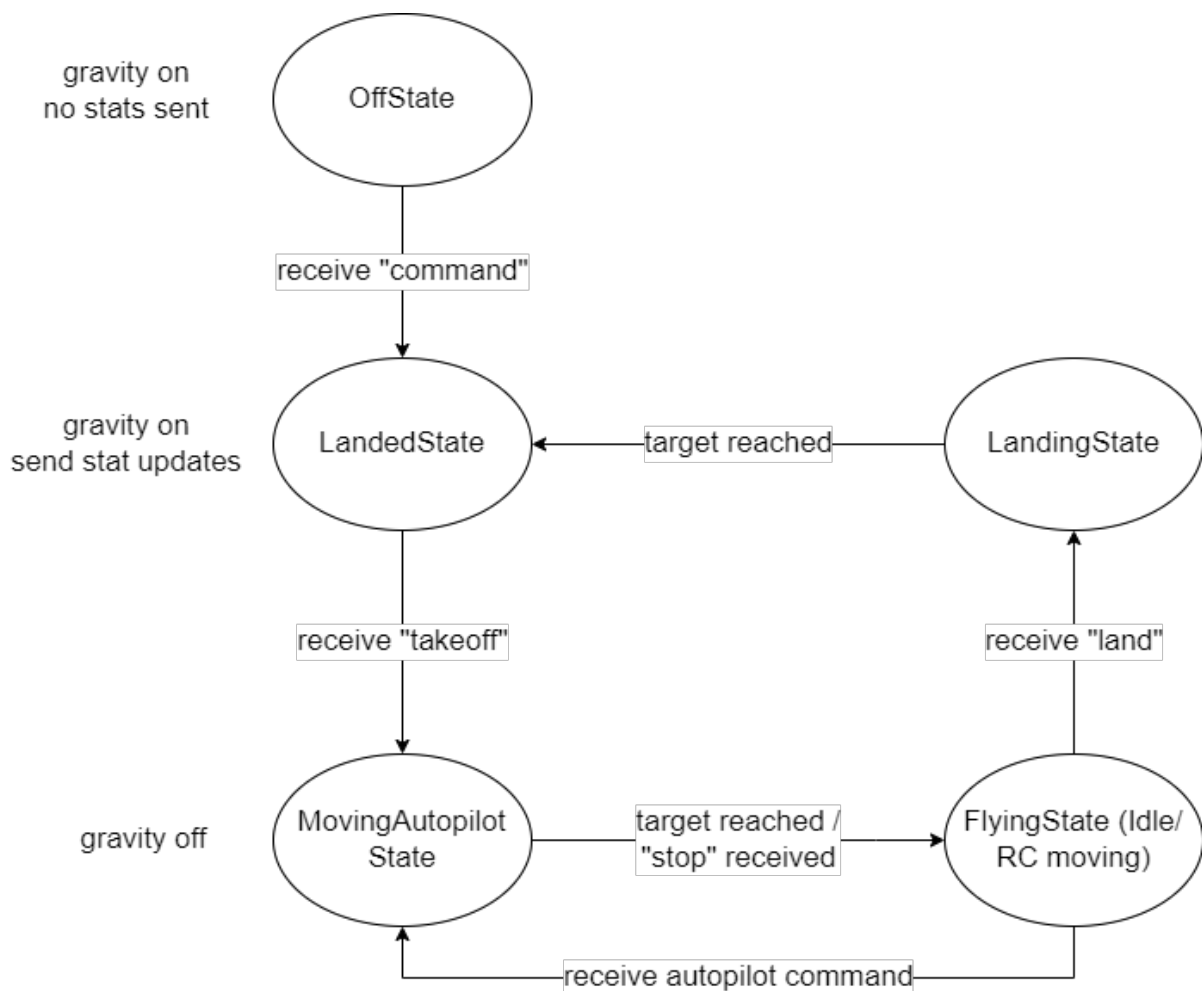


Figure 3.6: State machine of the drone

Using the Tello SDK gives the possibility to move the drone programmatically. However, the drone needs to be in the SDK mode to accept commands, and not all commands are accepted. For example, the drone can't land, if it hasn't taken off yet. This can be simulated by using a Finite State Machine (see section 2.9).

Off State

The drone starts in the "off state". In this state, the drone only accepts the command "command", which then enables the SDK Mode and changes to the "landed state". When changing to the SDK Mode, the drone also starts sending out status updates every 100ms.

Landed State

In the landed state, the drone is connected and ready to receive the “takeoff” command, which again changes the drone's state into the “Autopilot State”.

Autopilot State

In this state, the drone automatically moves to the requested position. It also includes rotating to a certain angle. Once the target position is reached, it will switch back to the “flying state”.

Setup move

When receiving a command that requires the autopilot to fly, the target position needs to be calculated. This happens before switching to the autopilot state and is simply done, by summing up the current position of the drone with the value received from the command.

Flying State (Idle / RC moving)

If the drone is just hovering, it is in the flying state. In this state, it accepts RC commands that are used to fly around with joystick controls. It is also accepting autopilot commands that would change to the “autopilot state”. In this state, we also accept the “land” command which is changing to the “landing state”.

Landing State

Is a modified version of the “autopilot state”. It calculates the distance to the ground and moves there the same way as the autopilot would. The only difference is that when the target is reached, it changes to the “landed state” instead of the “flying state”.

3.8 UDP Controller

The UDP Controller handles the UDP sockets and simulates the Tello Drone API. The commands are converted, and the methods of the Drone simulation are called accordingly.

3.8.1 Sockets

The Tello SDK has 3 UDP sockets. They are responsible for receiving and sending messages.

Command socket

The command socket is a simple UDP client that can send and receive data via UDP. It is responsible to receive commands and sending back a response.

State socket

The state socket is responsible for sending out the current state of the drone every 100ms. This is the format of the string:

```
"pitch:%d;roll:%d;yaw:%d;vgx:%d;vgy:%d;vgz:%d;templ:%d;temph:%d;tof:%d;h:
%d;bat:%d;baro:%.2f;time:%d;agx:%.2f;agy:%.2f;agz:%.2f;\r\n"
```

This table shows the meaning of the different values, as well as the format [6]. It doesn't make sense to implement all the values into Tello Camp directly. For example, the drone in Tello Camp doesn't have a battery, so we send back a hardcoded value.

Name	Description	Format	Source
pitch	The degree of the attitude pitch	Integer	Rigidbody
roll	The degree of the attitude roll	Integer	Rigidbody
yaw	The degree of the attitude yaw	Integer	Rigidbody
vgx	The speed of "x" axis	Integer	Rigidbody
vgy	The speed of the "y" axis	Integer	Rigidbody
vgz	The speed of the "z" axis	Integer	Rigidbody
templ	The lowest temperature in degree Celsius	Integer	Hardcoded value
temph	The highest temperature in degree Celsius	Integer	Hardcoded value
tof	The time-of-flight distance in cm	Integer	Timer
h	The height in cm	Integer	Rigidbody
bat	The percentage of the current battery level	Integer	Hardcoded value
baro	The barometer measurement in cm	Integer	Hardcoded value
time	The amount of time the motor has been used	Integer	Timer
agx	The acceleration of the "x" axis	Float .2f	Calculated
agy	The acceleration of the "y" axis	Float .2f	Calculated

agz	The acceleration of the “z” axis	Float .2f	Calculated
-----	----------------------------------	-----------	------------

The acceleration (agx, agy, agz) can’t be read directly from the rigidbody. It is calculated by dividing the change in velocity by the change in time.

$$a = \frac{\Delta v}{\Delta t}$$

Video stream socket

The video stream socket sends out the video signal from the drone. As implementing the video stream would take a lot of time, it is out of the scope of this project.

3.8.2 Parsing a message

Once the UDP Controller receives a message from the command socket, it parses the message and saves it into different variables. There are 3 different command types:

Type	Format	example	Output
Control command	“xxx”	“takeoff”	command = takeoff
Set command	“xxx a”	“Forward 300”	Command = forward Value = 300
RC command	“rc a b c d”	“rc 0 100 0 0”	Command = rc ValueA = 0 ValueB = 100 ValueC = 0 ValueD = 0

Figure 3.7: List of accepted commands

Once such a command is received, the controller is calling the according function on the drone.

3.9 User Interface

The user interface should be intuitive and give some useful insights when programming the drone.

3.9.1 Menus & Settings

Main Menu

The main menu is the welcome screen for the user. It should give a good first expression and have a button to start the actual game or quit the application. It should also have a place, where the user can change some settings.

Pause Menu

The pause menu is needed to pause the game and quit to the main menu or the game completely.

Settings

In some cases, it might happen that the required UDP port is blocked. Therefore, the option to change the default ports is needed. The settings is the place where the user can change the ports. It needs to be in the Main Menu screen, as the ports can't be changed when it is already in use.

Those settings should be saved individually for every user, so they don't need to change it every time they restart Tello Camp.

3.9.2 Heads-up Display

As Tello Camp is mainly needed for educational reasons, it should be as helpful as possible and give the user the needed information. To have it available all the time, a HUD (Heads-up Display) is a good way to display what the user needs.

Connection information

To connect a controller via UDP to Tello Camp, the IP Address and the used ports are necessary. If we display them directly, the user doesn't need to look for it themselves.

Debug information

Once the user is connected and has sent a message, it is useful to know the package the was received. Another useful information is the current state of the drone, as not all messages are accepted in all the states of the drone.

Instruction

There will also be some functions like resetting the drone or switching between the cameras. This can be done by pressing buttons on the keyboard, however the user needs to be aware of that possibility. Therefore, it can also be displayed on the HUD.

Action

Especially in the beginning, the user probably doesn't know exactly how the drone and the Tello SDK works. As very first step, before accepting any other commands, the "command" command needs to be sent to enable the SDK mode. It might be frustrating when someone forgets to send that command in the beginning.

Therefore, we want to inform the user of what he has to do, until the drone is in the air and can be moved.

3.10 3D-World

The 3D-World has two main purposes. First, it needs to look good, so students enjoy working with Tello Camp. Secondly, it needs to offer a small training course, to teach how to fly a drone.

Enjoyable

The world doesn't have to be photorealistic. To run on as many different computers as possible, it would make sense to keep the graphics simple. Using post processing is an easy way to enhance the graphics to make it look more appealing.

The default sky in Unity doesn't look realistic and makes Tello Camp look cheap. There are a lot of better options available in the asset store.

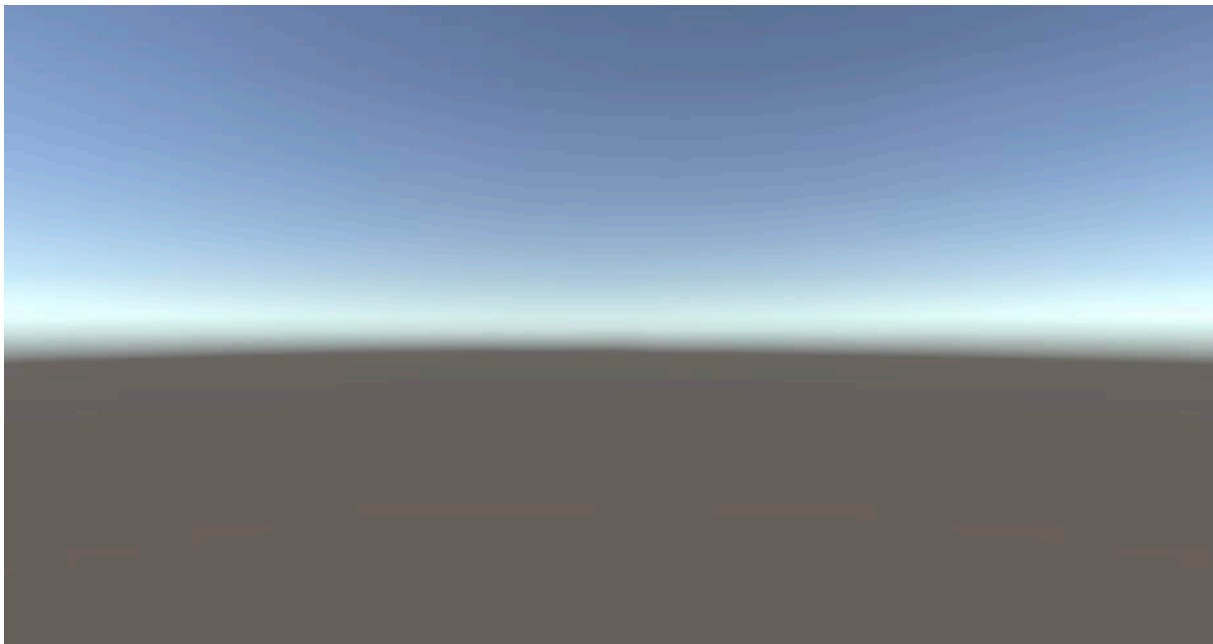


Figure 3.8: Default sky in Unity

The drone model also doesn't need to be an exact replica of the Tello Drone. It can be any quadcopter, about the same size. Important is, that it behaves like the Tello Drone and that the user can distinguish between the front and the back, to know in which direction they are flying.

Obstacle course

The obstacle course should help the learn how to fly a drone. The DJI Simulator [7] offers several different training sequences to absolve. Building multiple maps with

raising difficulty, would be helpful to train good pilots, however, because of time constraints, we only create the basis of multiple possible courses, and one example.

To get a good feeling for the drone, it is helpful to have a small playground, where one can fly around and explore things. Different obstacles in the world should encourage the user to fly the drone in all directions. Therefore, it needs something to fly up, down, and make turns. Also having something to fly through is a great obstacle, as one needs to be focused and having the drone under control, to not crash it.

4 Implementation

Tello Camp is built with Unity and uses C# and the .Net framework. In this chapter we explain how we implemented the concept introduced in the previous chapter.

4.1 Architecture

4.1.1 Class Diagram

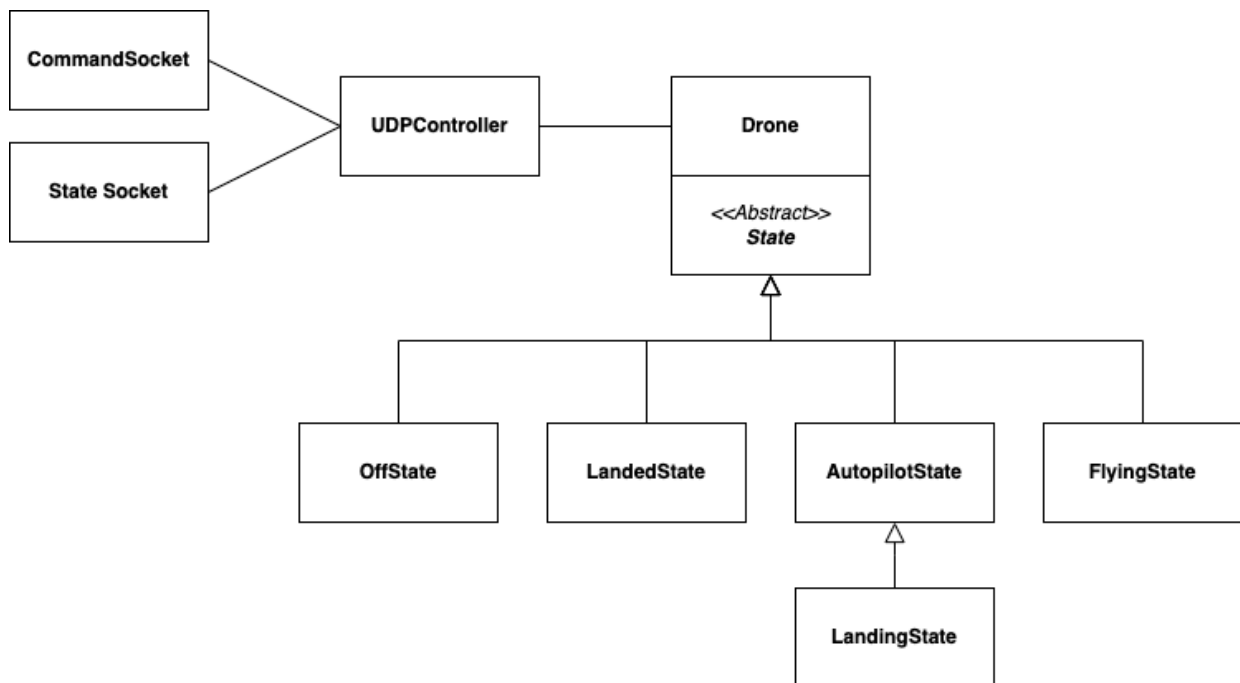


Figure 4.1: Class Diagram

The drone with its state machine is encapsulated. It has its own API to control it, which means the **UDPController** can be easily replaced with any other kind of controller for a higher adaptability to other forms of connections.

4.2 Drone

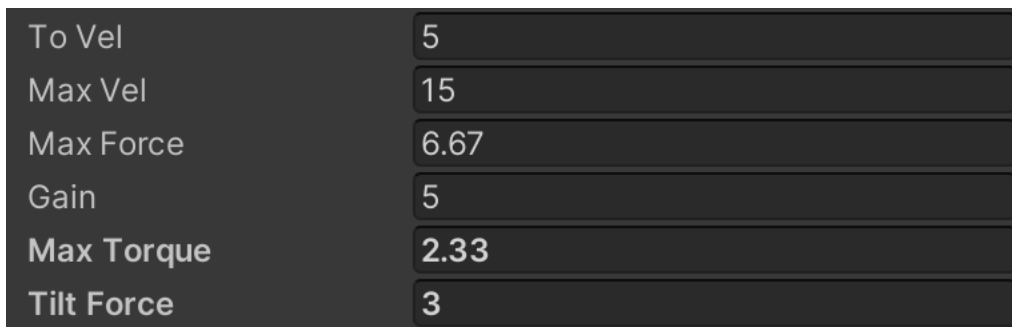
The drone is our main object in our application. Any function the controller is calling, will be on the drone. In Unity the **GameObject** contains the script and a **rigidbody** for the physics. The children are the drone 3d model with colliders and a camera.

4.2.1 Main Class

This is the script attached to the drone GameObject.

Variables

The C# class holds all the variables necessary to tweak the drone. These fields are private however, they are marked as serializable, which allows the field to be modified in the Unity editor. [15]

A screenshot of the Unity Inspector window showing a list of variables and their values. The variables are To Vel, Max Vel, Max Force, Gain, Max Torque, and Tilt Force. The values are 5, 15, 6.67, 5, 2.33, and 3 respectively. The values are displayed in a dark-themed interface with light-colored text.

To Vel	5
Max Vel	15
Max Force	6.67
Gain	5
Max Torque	2.33
Tilt Force	3

Figure 4.2: Serializable values in Unity

We also have all the variables needed for the state update, which is sent out every 100ms.

Functions in FixedUpdate()

The drone class also contains few functions, that are generally needed by all the states. These are namely the AutoBalance function and the Tilt function.

Setup functions

Setup functions are needed to prepare the move that the autopilot is going to do when changing the state. These functions are calculating the target.

Setup Move

In this function, the drone takes its current position and adds the amount in the direction received from the controller, to get the TargetPosition.

Setup Turn

This function does basically the same thing for rotation, however it calculates 3 targetRotations, to have the turn in 3 steps, as described in the concept.

4.2.2 State machine

The state machine is a part of the drone class. The drone keeps track of its current state and when a function is called, it behaves depending on the state.

All the different states inherit from the abstract class State. All the other state classes can implement the functions defined in the abstract class, if they want to have a specific behaviour.

4.3 Controller

The main purpose of Tello Camp is that it can be controlled via UDP. Therefore, our main controller is the UDP controller. However, as our drone is encapsulated from the controller, any other controller can be built as well. We used that to build a test controller to test the drone while developing and tweaking.

4.3.1 UDP Controller

The UDP Controller creates the UDP Sockets, handles the incoming messages, forwards it to the drone and sends back a response.

Handling commands

During the initialization, it creates a new Command Socket object, with the port set in the settings (see chapter 4.7.1 Main Menu). The command socket is a simple UDP client that can send and receive data via UDP.

When the socket is receiving a message, it gets the message as a string and the IP address as IPAddress object. The IP address of the sender is needed, to establish a connection for the state socket.

The message string is saved into different variables as described in the concept. To convert the message into the correct command for the drone, we use a switch statement. If the command couldn't be recognized, we send back "Error".

Sending state updates

When the command socket is receiving the message "command", it creates the state socket with the received IP address. We then start a coroutine, that is sending out the state of the drone, ever 100ms. The states are coming directly from the drone and are either read from the rigidbody or hardcoded, as described in the Context.

Accepted Commands

Due to the duration of the project, we only implemented the most important commands and not the full Tello SDK. This is the list of the accepted commands and what they trigger on the drone, once received with a correct value. The reply is always either “OK” after successfully completing the action or “Error” if the command wasn’t recognized.

Command	Range	Accepted in State	Description
command		offState	enables SDK mode, opens StateSocket & starts sending states. Drone switches into “landedState”
takeoff		landedState	Auto Takeoff. Drone switches into “autopilotState” and flies up. After reaching the target, it switches to “flyingState.”
land		flyingState	The drone lands and switches into “landedState”
up x	x = 20-500	flyingState	Switches into “autopilotState” and flies up x cm. After reaching the target it switches back to “flyingState”
down x	x = 20-500	flyingState	Switches into “autopilotState” and flies down x cm. After reaching the target it switches back to “flyingState”
left x	x = 20-500	flyingState	Switches into “autopilotState” and flies left x cm. After reaching the target it switches back to “flyingState”
right x	x = 20-500	flyingState	Switches into “autopilotState” and flies right x cm. After reaching the target it switches back to “flyingState”
forward x	x = 20-500	flyingState	Switches into “autopilotState” and

			flies forward x cm. After reaching the target it switches back to "flyingState"
back x	$x = 20 - 500$	flyingState	Switches into "autopilotState" and flies back x cm. After reaching the target it switches back to "flyingState"
cw x	$x = 1 - 360$	flyingState	Switches into "autopilotState" and turns x degree clockwise. After reaching the target it switches back to "flyingState"
ccw x	$x = 1 - 360$	flyingState	Switches into "autopilotState" and turns x degree counterclockwise. After reaching the target it switches back to "flyingState"
stop		takeoffState flyingState autopilotState landingState	stops the current action and goes into flying state.
emergency		all states	Switches into "offState". The drone turns off and gravity is enabled -> The drone will crash into the ground.
rc a b c d	a = -100 – 100 (left / right) b = -100 – 100 (forward / back) c = -100 – 100 (up / down) d = -100 – 100 (yaw)	flyingState	Stays in "flyingState". This command is used to control the drone with a remote controller via four channels. This is the only case where no response is sent.
speed x	$x = 10 - 100$	all states	limits the speed of the drone to x cm/s

Unsupported commands

Tello Camp does not support all the commands listed in the Tello SDK, as some of them were out of the scope of this project. The most important unsupported functions are:

- Flying curves
- streamon / streamoff
- Read commands like “speed?” or “battery?”

Thread safety

Our UDP Socket implementation is creating new threads to receive data. Unfortunately, the Unity API is not thread-safe and therefore doesn't allow calling unity functions from a different thread than the main thread [16]. To solve this problem, we use the “UnityMainThreadDispatcher” by @PimDeWitte². This allows us to dispatch the message to the main thread.

4.3.2 Test Controller

We have also created a small test controller that allows us to test & tweak the drone implementation without using the UDP Controller. This is a script that launches a command every few seconds.

Implementing a test controller saved us a huge amount of time while developing the movement of the drone. Using the UDP Controller, means that every time we want to test a change, we needed to open the terminal, open a UDP connection, send the command & takeoff command before we could even test anything. Also, manual testing would always be slightly different, and we would never test all the functions.

The Test controller is a small script, attached to the UDP Controller object. When enabled, it would launch all the commands we have implemented at the time and play through the state machine. If anything wouldn't work, we would see it immediately.

² <https://github.com/PimDeWitte/UnityMainThreadDispatcher>

4.4 Flying in Tello Camp

The main movement of the drone happens in the states `FlyingState` and `AutopilotState`, that are described in this section.

4.4.1 `FlyingState`

The drone goes into `FlyingState` once it took off and is hovering above the ground. In that state, RC commands are accepted. RC commands is the have the control command “rc” followed 4 values, namely a, b, c and d. The values represent a percentage of max acceleration in our code. This means, the value stored inside variables is divided by 100, so it ranges between 0 and 1.

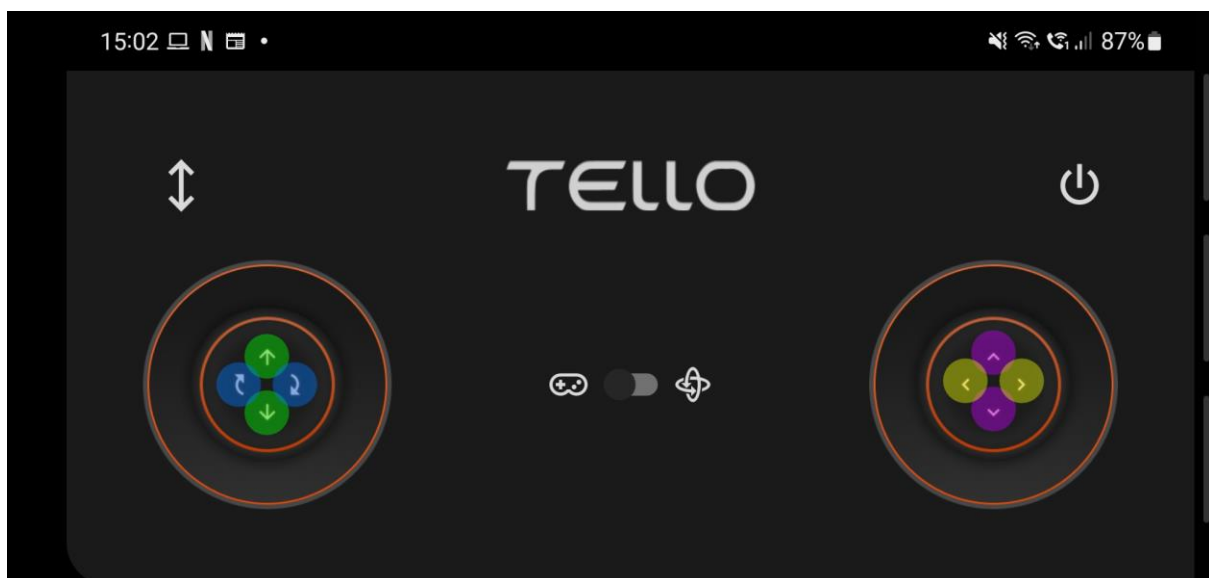


Figure 4.3: Example of a controller

a (yellow): left/right with values ranging -100 to 100.

b (violet): backward/forward with values ranging -100 to 100.

c (green): down/up with values ranging -100 to 100.

d (blue): rotate left/right with values ranging -100 to 100.

*Example: if value b is 100, it means the drone flies forward with 100% of its acceleration.
If it is 50, the drone flies forward half as fast as it can.*

4.4.1.1 Movement

Because this is a physics related operation, the movement is calculated in the FixedUpdate() loop. Moving the drone uses a native function of the rigidbody component. This allows for moving the drone in all three directions. up/down, forward/backward and right/left.

This code adds force in the desired direction, accelerating the drone:

```
drone.rb.AddRelativeForce(drone.MoveDirectionRC * drone.maxForce);
```

The force applied is calculated with this multiplication:

```
drone.MoveDirectionRC * drone.maxForce
```

drone.MoveDirectionRC

This is always a vector with a length between 0 and 1 and the desired flying direction.

drone.maxForce

This is the force which allows the drone to reach its maximum velocity. The standard velocity of the drone is 1m/s which is possible with a maxForce set to 6.67 and drag of 5. A combination of drag and force allows us to set a maximum velocity of the drone. Once the drone reaches maximum velocity, the force created by drag counters our maximum force which is added every update, meaning the drone experiences 0 force and halts its acceleration. When we wish to stop the drone, it is as simple as to stop adding force, and the drag will slow the drone down and make it halt.

The stronger the force and drag is, the faster the drone accelerates and brakes. These values have been tuned in order to feel very similar to the real drone.

4.4.1.2 Rotation

Rotating the drone is done in a similar way and also in the FixedUpdate() loop:

```
drone.rb.AddTorque(drone.transform.up * drone.maxTorque *  
drone.RotationRC);
```

This function is also a native function of the rigidbody component.

Drone.transform.up

This is the vector around which we rotate. Also called the yaw of the drone.

drone.maxTorque

The maximum torque which also sets its maximum rotation speed combined with angular drag.

drone.RotationRC

Similar to `drone.movingRC`. The direction into which we rotate, with a length between 0 and 1. A short length of this vector means a slow rotation, and a length of 1 allows for a rotation at maximum torque.

The explanation for rotation is the same as for moving. Instead of adding force to the drone, we add torque. A simple analogy would be to say that it is a rotational force which is applied to the drone.

4.4.2 AutopilotState

This state is used for moving the drone a certain distance or rotating it a certain angle. It is more challenging because its braking distance needs to be calculated in order to reach a certain distance very precisely. Our solution calculates the velocity of the drone based on the distance it has yet to travel to reach its destination. In simpler terms: the further away the drone is from its target, the faster it flies (capped at a maximum speed), and the closer it gets to the target, the more it slows down. This is also called a PID Controller (proportional–integral–derivative controller).

4.4.2.1 Movement

```
Vector3 dist = drone.TargetPosition - drone.rb.position;
// calc a target vel proportional to distance (clamped to maxVel)
Vector3 tgtVel = Vector3.ClampMagnitude(drone.toVel * dist, drone.maxVel);
// calculate the velocity error
Vector3 errorPos = tgtVel - drone.rb.velocity;
// calc a force proportional to the error (clamped to maxForce)
Vector3 force = Vector3.ClampMagnitude(drone.gain * errorPos,
drone.maxForce);
drone.rb.AddForce(force);

if (Vector3.Distance(drone.rb.position, drone.TargetPosition) < 0.01f)
{
    TargetReached();
}
```

Figure 4.4: Code for flying in 3 dimensions by adding force

This algorithm calculates the necessary force to add for the drone to reach our target velocity. If the drone flies too slow, force is added to increase its speed, and if it flies too fast, force added is negative, slowing the drone down. The meaning of every value is explained below:

Vector3 dist

The vector from the position of the drone to the target position. Its length represents the distance the drone has to travel until its destination.

Vector3 tgtVel

The velocity the drone is accelerated to. The function “Vector3.ClampMagnitude” takes 2 arguments. The first is the desired velocity (based on distance in our case), and the second is the maximum value this vector is allowed to have.

Vector3 errorPos

This vector is the difference in the drones actual velocity, and its desired velocity. If the drone needs to slow down, because it is getting closer to its destination, then this vector will go in the opposite direction we’re moving in. Meaning, the drone should slow down.

Vector3 force

For the force, we once again use Vector3.ClampMagnitude. The force which accelerates the drone cannot exceed drone.maxForce, meaning our drone has a limit to its acceleration. The first argument multiplies the errorPos with drone.gain. Drone.gain magnifies the errorPos vector. It is necessary to speed up the acceleration when the drone is close to its targeted velocity, because otherwise before the drone reaches its maximum speed, it would already have to slow down because it has almost reached its destination.

The last line of code determines whether we have reached our target. When the distance to the target is smaller than 0.01, it is considered achieved. The function TargetReached() resets certain values so they are ready to be set in the next command.

4.4.2.2 Rotation

The rotation of the drone works the same way, but it is done by adding torque instead of force. The further it is away from its goal, the faster it rotates (limited by a maxTorque variable). The closer it is to the desired angle, the more it slows down. The angle of the drone is split in 3 parts, to avoid the issues mentioned in the chapter “Quaternions”.

```
//Find the rotation difference in eulers
Quaternion diff = Quaternion.Inverse(drone.rb.rotation) *
partRotation[pointer];
Vector3 eulers = OrientTorque(diff.eulerAngles);
Vector3 torque = eulers;
//put the torque back in body space
torque = Vector3.ClampMagnitude(drone.rb.rotation * torque,
drone.maxTorque);
drone.rb.AddTorque(torque);
```

Figure 4.5: Code for rotating a set angle

4.5 Automatic balance system

```
public void AutoBalanceDrone()
{
    float zangle = rb.rotation.eulerAngles.z;
    float xangle = rb.rotation.eulerAngles.x;
    if (zangle > 1 && zangle <= 180 || zangle < 359 && zangle > 180 ||
xangle > 1 && xangle <= 180 ||
        xangle < 359 && xangle > 180)
    {
        StartCoroutine(RotateToZero( new Quaternion(0, rb.rotation.y, 0,
rb.rotation.w), 1f));
    }
}
```

Figure 4.6: Code which automatically balances the drone

First thing we do, is eliminating the y axis from the rotational vectors. Only x and z angle are important because these axes are responsible for keeping the drone horizontally stable.

If the drones x and z rotations are not within 1° of their natural position (which is 0°), the drone starts to rotate towards its normal state.

4.6 Tilt

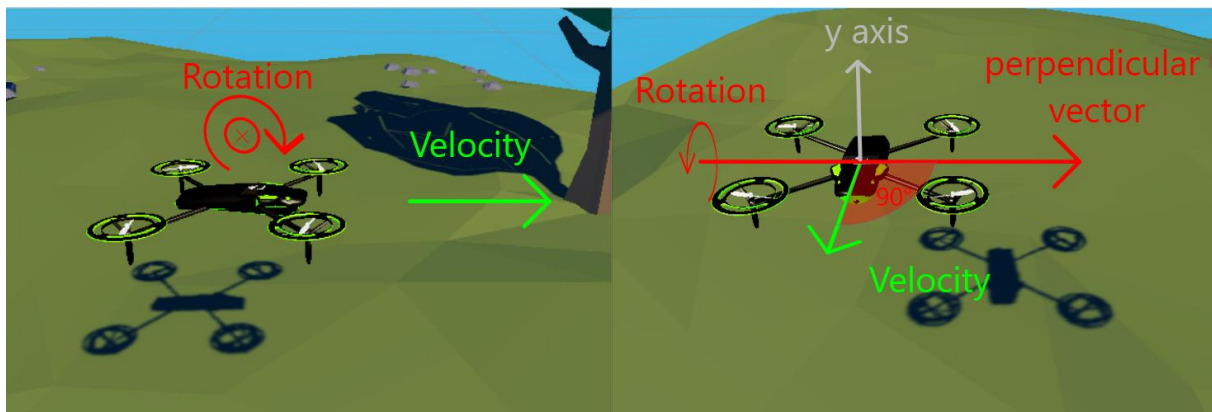


Figure 4.7: visualisation of "tilt" vector

The inclination of the drone is mapped to 3 times the velocity. This means, in order to fly 1 m/s, the drone tilts 3° . If it flies 3 m/s, the drone tilts 9° , as the real Tello Drone also does [17].

We tilt only the model of the drone, so we don't mess with the rigidbody and our auto balance function.

Tilting the drone is done in two steps. First, we calculate the perpendicular vector, and then we rotate the drone for a certain amount in the correct direction.

CalculateTiltQuaternion(Vector3 velocityForTilt)

Calculates the rotational quaternion to multiply our droneModel.rotation with.

```
Quaternion CalculateTiltQuaternion(Vector3 velocityForTilt)
{
    Vector3 tiltAxis = Vector3.Cross(velocityForTilt, Vector3.up);
    float angle = Mathf.Clamp(-velocityForTilt.magnitude, -25, 25);
    Quaternion targetRotation = Quaternion.AngleAxis(angle, tiltAxis) *
rb.rotation;
    return targetRotation;
}
```

Figure 4.8: function which returns the perpendicular vector from Figure 4.7

tiltAxis

This is the cross product of the velocity vector of the drone (velocityForTilt) and a normalised vector on the y axis (Vector3.up = (0, 1, 0)). tiltAxis is thus our perpendicular vector marked red on Figure 4.4.

angle

Is the target angle we want the drone to tilt. It is clamped (limited) to -25° and $+25^\circ$.

targetRotation

This is the quaternion which describes the desired rotation of our droneModel from the rigidbody.

TiltToVelocity()

This function is called in the fixedUpdate() loop of the drone and tilts the drone as soon the drone is moving.

```
void TiltToVelocity()
{
    Vector3 velocityForTilt = rb.velocity * tiltForce;
    Quaternion targetRotation = CalculateTiltVector(velocityForTilt);
    droneModel.transform.rotation =
Quaternion.Lerp(droneModel.transform.rotation, targetRotation, 10 *
Time.deltaTime);
}
```

Figure 4.9: function which tilts 3° per 1m/s in the direction of flight

velocityForTilt

Is our “mapping” mentioned in chapter 3.1 Movement. For every 1 m/s velocity the drone tilts 3° . tiltForce is a float which equals 3.

targetRotation

is the calculated rotation from CalculateTiltVector().

droneModel.transform.rotation

Rotating the drone is done by rotating only a bit towards the goal. Quaternion.Lerp allows for the creation of automatic steps between two rotations. In this case, one fifth at once is chosen ($10 * \text{Time.deltaTime} = 10 * 0.02$). Every fixedUpdate() in unity, the droneModel rotates 20% towards the targeted rotation.

4.7 User Interface

This section explains the implementation of the user interface.

4.7.1 Main Menu



Figure 4.10: Tello Camp start screen

The main menu is the first thing the user sees. It is designed simply and should guide the user to the right place. In the main menu we have 3 buttons:

Start

Loads the scene and starts the actual Tello Camp Playground (see section 4.8).

Settings

This toggles the settings on the right-hand side. The only settings available are configuring the ports for the command socket and state socket. Once the user changes those settings, they are saved in the PlayerPrefs. PlayerPrefs is a unity class that can save values between game sessions [18]. This way, the user changes the settings once and doesn't need to update them every time he launches the game.

Exit

The exit button closes the application.

4.7.2 Pause Menu

The pause menu can be opened by pressing the “escape” key. This also pauses the game completely, by setting the `Time.timeScale` to 0. This changes how often the `FixedUpdate()` function is called and therefore the loop is stopped [19]. Also here, we have 3 buttons:

Continue

This resumes the game. Alternatively, one can press the “escape” button again. If the drone was moving in autopilot mode, it would continue moving to the target position.

Main Menu

This brings you back into the main menu and terminates the command socket and state socket. This is important if you want to change the ports in the settings.

Exit

The exit button terminates the command socket and the state socket and closes the application.

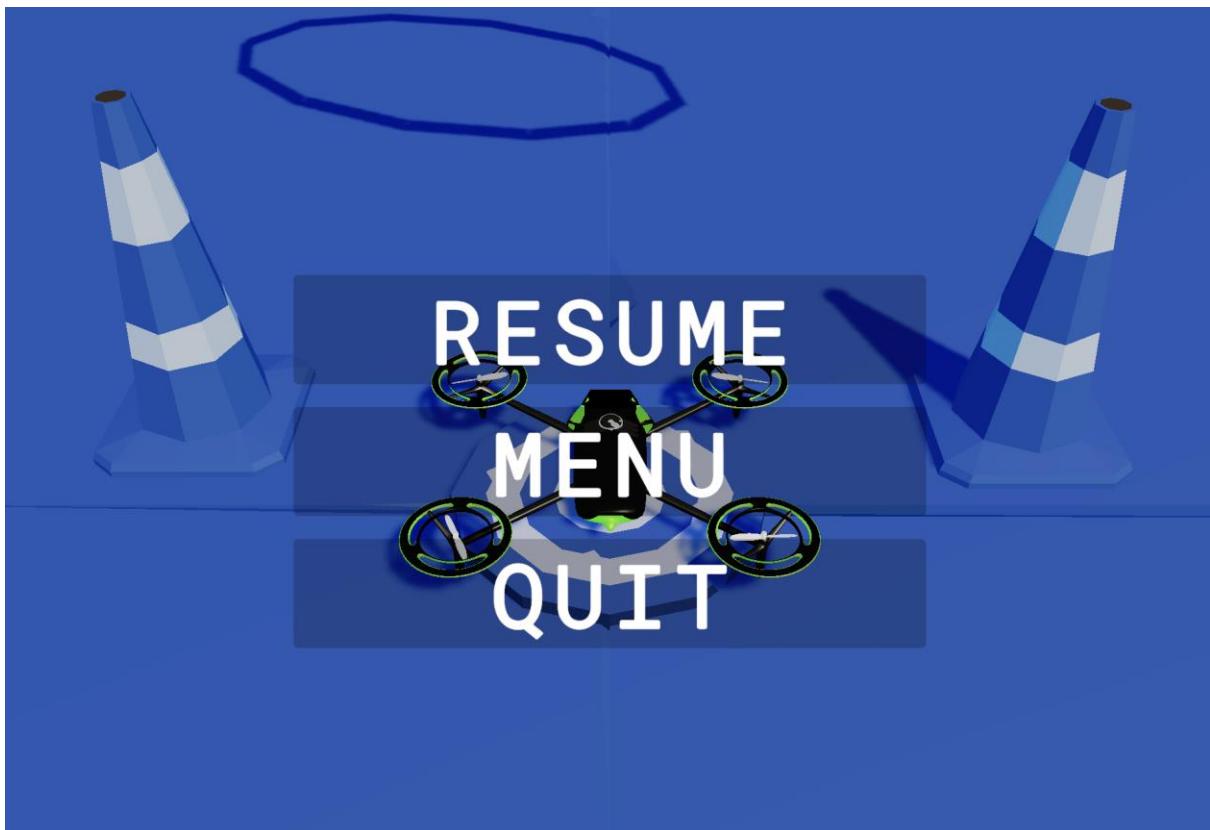


Figure 4.11: Pause menu

4.7.3 Heads-up Display



Figure 4.12: Heads-up Display

The Heads-up Display (HUD) is visible when starting the game. It contains useful information for the user. The information is sorted into 5 different categories:

Connection

Shows the IP Address of the device where the simulator is running on. This is needed so the external controller can be easily connected to the correct machine. We also show the command port and state port for the same reason.



Figure 4.13: Connection UI

Debug

This shows you the last package that the UDP controller received. It also displays the current state of the drone, so that the user knows what commands are accepted.

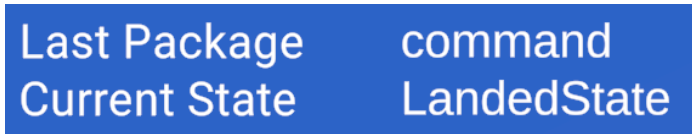


Figure 4.14: Debug UI

Instructions

By pressing the “tab” button, the user can hide and show the information displayed on the HUD. The only thing that is always visible is the speed. Hiding the information is especially useful once the programming education is completed and the user wants to focus on learning how to fly the drone.

Pressing the “R” button resets the drone. This will teleport the drone back to the starting position and switch the drone back into the “off” state.

The user can also switch between two cameras by pressing the number 1 or 2. The default camera is positioned behind the starting position of the drone. This camera simulates the view from a standing person, controlling the drone. It always looks in the direction of the drone but doesn’t move. The second camera is mounted on the drone. This view gives the user a first-person view from the drone’s camera. It is the same view as the Tello drone sends on its video stream.



Figure 4.15: Instructions UI

Action

The “action text” informs the user what to do next. This is especially useful when using the Tello Camp and the Tello SDK the first time. As described in the state machine chapter, the drone not always accepts all the commands. Therefore, we display a message in the “offState” telling to send the “command” command. Once this is done, the drone switches to the “landedState” and we display that the user should send “takeoff”. After that, we don’t display any message, until the user is in one of the two states.

We use the action text also to display error messages when the UDP Sockets couldn’t be initialized.



Figure 4.16: Action UI

Speed

We display the current speed of the drone in the bottom right corner. The speed is directly calculated by the rigid body is displayed in meters per second.

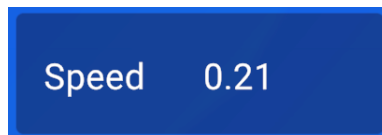


Figure 4.17: Speed UI

4.8 3D-World

The environment can be structured into the surrounding world and the small playground we designed, which is helping to learn how to fly the drone. In the end we also applied some post processing to make the world look good.

Our world is created in a low poly style. Low poly means that the objects are designed with a low number of rectangles. We took this decision as it comes with better performance than photo realistic worlds, but still is enjoyable to look at. We mainly used the free “POLYGON Starter Pack - Low Poly 3D Art by Synty”³ from the Unity Asset Store. This pack contains enough assets to create a world and putting some obstacles in it to fly through and around.

4.8.1 Drone

The drone model is from the “Simple Drone”⁴ asset by “GameAnime”. The asset comes with animated fans that easily can be turned on and off. In a perfect simulation the fans would turn in a different speed depending on the movement. This has a high implementation effort and only a low return on realism, therefore we didn’t prioritize that.

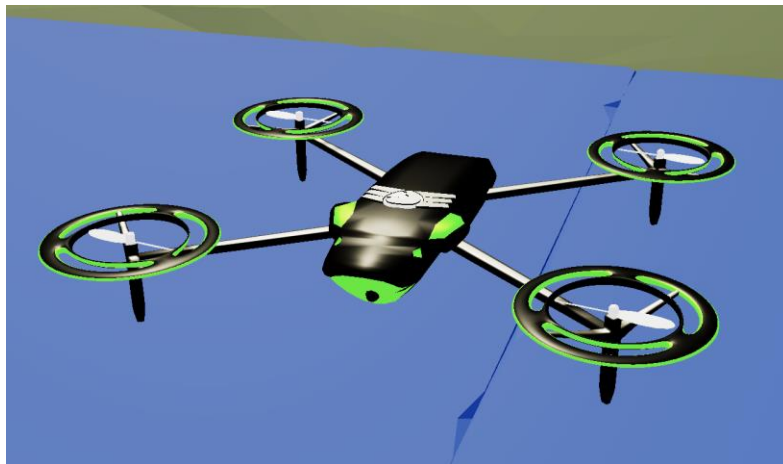


Figure 4.18: Drone model

³ <https://assetstore.unity.com/packages/3d/props/polygon-starter-pack-low-poly-3d-art-by-synt-156819>

⁴ <https://assetstore.unity.com/packages/3d/vehicles/air/simple-drone-190684>

4.8.2 Surroundings

The world is basically a green grass ground, with a few trees and mountains. This gives it a friendly look without being distracting. The sky around the world is from the “FREE Skybox Extended Shader”⁵. It gives a beautiful sky that fits perfectly into the scene.



Figure 4.19: Surroundings

⁵ <https://assetstore.unity.com/packages/vfx/shaders/free-skybox-extended-shader-107400>

4.8.3 Playground

The playground is marked by blue tiles on the ground. It is designed like a small obstacle course the user can use to get a feeling for the drone, but it doesn't have any limitations. One can also just fly around, without following the course.

The parkour has different obstacles to fly through and around:

Ring

The ring hovers in the air and the drone can fly through.

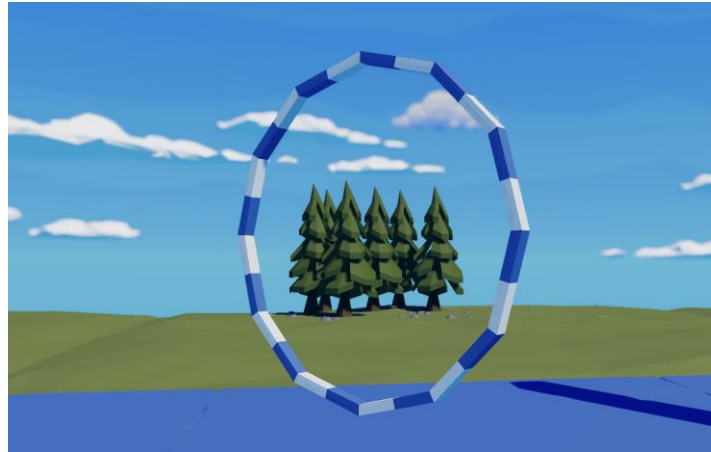


Figure 4.20: Ring

Pillar

A high pillar that can be used to fly around.

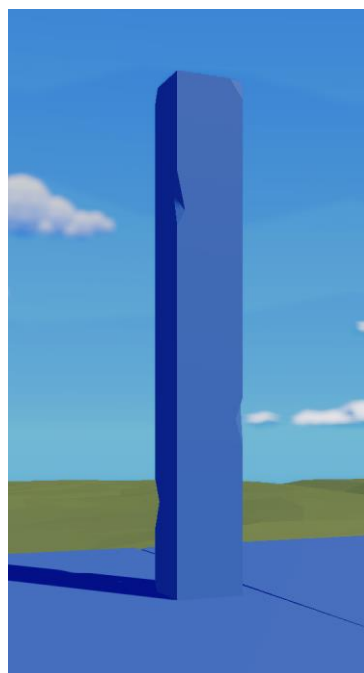


Figure 4.21: Pillar

Stairs

The stairs are helpful to learn how to ascend and fly forward at the same time.

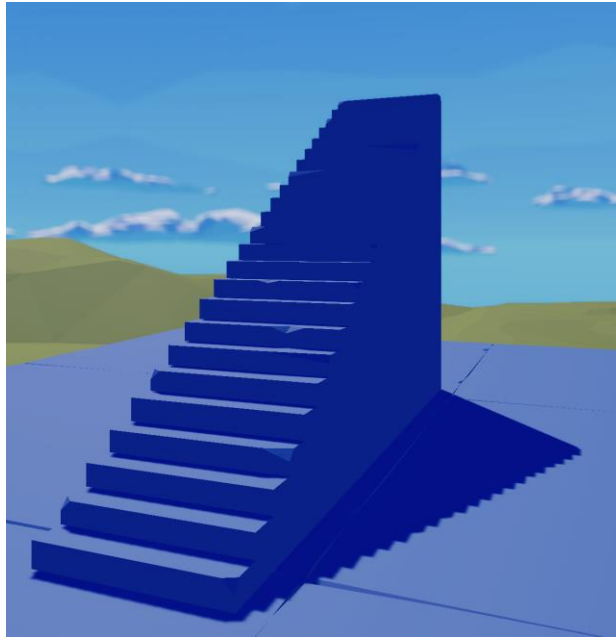


Figure 4.22: Stairs

Pipe

The pipe is like the ring, however as it is longer, the user needs to fly more precisely.

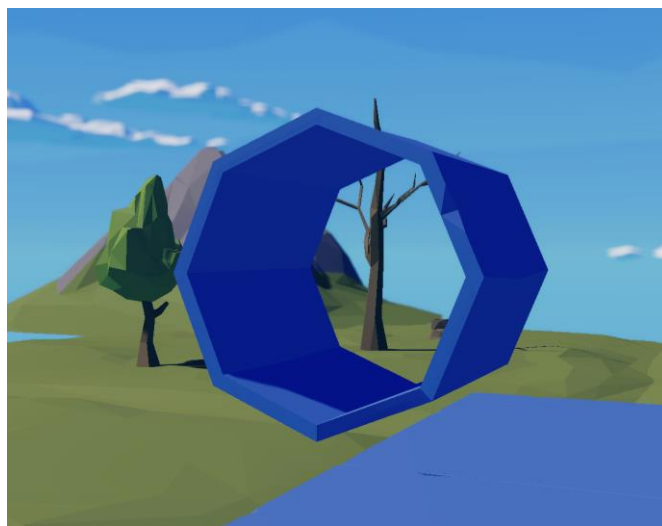


Figure 4.23: Pipe

4.8.4 Render Pipeline & Post Processing

To give the scene a better look, we switched from the Built-in render pipeline to the Universal Render Pipeline (URP). Using a different render pipeline is a great way to improve the looks in a game. We selected the URP as it is a great trade-off between look and performance [20].

Using URP gives us the option to enable post processing, which adds effects to the image of Tello Camp to make it look different. We added the following effects:

Tone mapping

This is by far the most important change to make the Tello Camp look more appealing. The tone mapping options in Unity gives us the option to choose from “Neutral” and “ACES”. Neutral has only a minimal impact on hue and saturation, whereby ACES gives us a more cinematic look [21]. We’ve selected “ACES” as it changes the color hue and saturation in a way that looks more visually appealing.

Motion Blur

Motion Blur is a subtle effect that is only visible when a moving object is faster than the camera’s exposure time [22]. It blurs the surroundings which leads to a more realistic effect.

Color Adjustments & Color Curves

We slightly adjusted the colors to make the scene more vivid.

Depth of Field

We use this effect to blur the elements that are far away. This makes it less obvious that the world has an end.

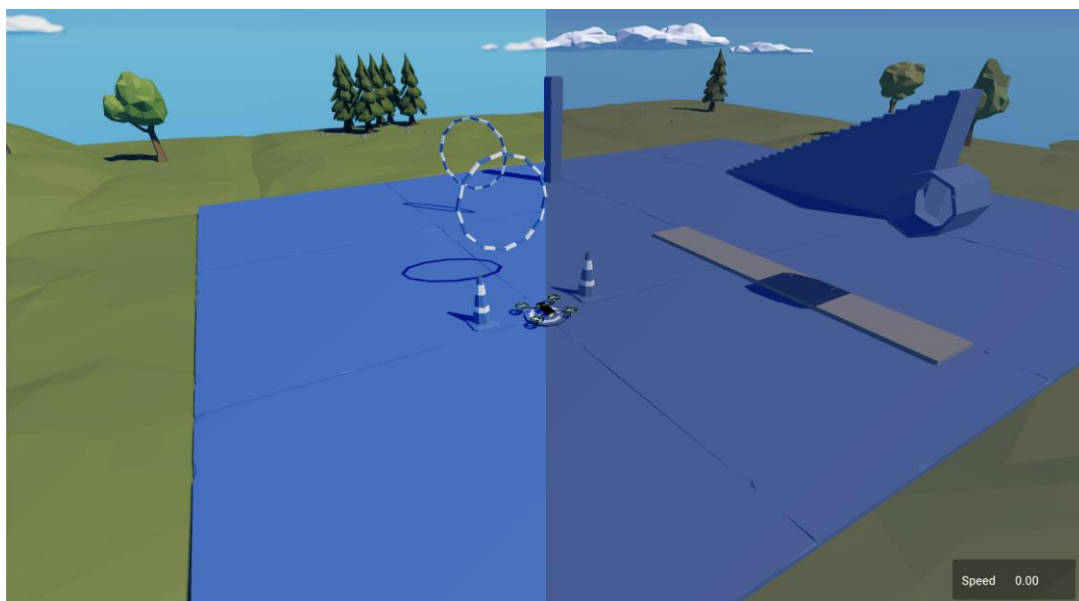


Figure 4.24: Left – after processing, right –before processing

5 Discussion

5.1 Value of Tello Camp

Tello Camp enables students an immersive experience flying the Tello drone with their self-made controller. Thanks to the realistic implementation of the drone's behaviour, any program they write for Tello Camp will work the exact same way on the real drone.

Students can learn to fly the drone using the software, without the fear of damaging the drone and having to replace its rotors, or in the worst case, the entire drone. The virtual drone also does not run out of battery, meaning a learning session will be longer than the usual 13 minutes flight time of the real drone. Because the software was developed to mimic the Tello Drone as precisely as possible, students are much more familiar flying the real drone the first time than ever before.

5.2 Technical challenges and opportunities

5.2.1 Video Stream

The real Tello Drone has a camera which lets the user see from a first-person view. Unfortunately, the time for this project is limited and the video stream did not make it into the final product. We expect most users to fly the Tello Drone while observing and not the camera feed on their smartphones. Therefore, the video stream had a lower priority and was not implemented.

5.2.2 Quaternions

Quaternion angles are useful and don't face the most important issue Euler angles do: The gimbal lock. The fact that unity and most other game engines use quaternions means a new understanding of how rotations are done was necessary.

Rotating a vector by a certain angle is done by multiplying a quaternion with the vector. Learning about quaternions was a great opportunity because it is used in almost any kind of game-engine worldwide.

5.3 Outlook

The results show that it is possible to create an immersive simulation of the Tello Drone which aids students in learning how to program the drone. The software could benefit from a few more improvements and additions, namely:

Additional maps

For the time being, students can only access one course and fly within its perimeter. Having additional maps with different obstacles enables more situations students might encounter with the real drone.

Time trials

The scenes in which the students fly can be changed into a time trial parkour. For example, students have to fly through a specific number of checkpoints before reaching the end, creating a more competitive and fun way to learn.

Additional commands

Tello Camp does not yet support flying a curve with a autopilot command, and a few others are missing as well. The simulation would be more complete with more/all commands implemented.

Video stream

Tello Camp does not yet support a video stream, meaning students cannot see the view of the drone on their controller. If a student wishes to test their video stream on their controller, they would have to return to the old Tello Simulator, which creates a less immersive and complete experience than our software.

6 Conclusion

We have explored different simulators on the market, investigated multiple concepts for our requirements and implemented the most suitable ones in Tello Camp.

Tello Camp is a game developed in Unity, which accurately simulates the behaviour of a real Tello Drone. It allows students to program and develop a controller for it. Everything that can be programmed for our Tello Camp software, also works with the real Tello Drone.

The environment is much more realistic than in the previous software, Tello Simulator, and flying through it feels just like the real deal. Students can train flying with Tello Camp, making them less likely to damage the real drone when they first fly with it. The immersive user experience of Tello Camp makes it a wonderful training tool for students, whether it is for programming or learning to fly.

This project shows how it is possible to virtualise a real product like the Tello Drone with a game engine like Unity.

Bibliography

- [1] "DJI Flight Controller Concepts," DJI, 2016. [Online]. Available: https://developer.dji.com/mobile-sdk/documentation/introduction/flightController_concepts.html. [Accessed 29 December 2021].
- [2] C. Allen, "How a Quadcopter Works," 2014. [Online]. Available: http://ffden-2.phys.uaf.edu/webproj/212_spring_2014/Clay_Allen/clay_allen/works.html. [Accessed 29 December 2021].
- [3] "Drones | The complete flight dynamics," 2020. [Online]. Available: <https://www.youtube.com/watch?v=CoKBu2ihp-s>. [Accessed 29 December 2021].
- [4] "Tello Edu," Ryzerobotics, 2022. [Online]. Available: <https://www.ryzerobotics.com/tello-edu>. [Accessed 29 December 2021].
- [5] "Tello," DJI, 2022. [Online]. Available: <https://store.dji.com/ch/product/tello?vid=38421>. [Accessed 20 January 2022].
- [6] "Tello SDK," Ryzerobotics, 2018. [Online]. Available: <https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf>. [Accessed 28 December 2021].
- [7] "DJI Flight Simulator," DJI, 2022. [Online]. Available: <https://www.dji.com/ch/simulator>. [Accessed 29 December 2021].
- [8] "Important Classes - MonoBehaviour," Unity Technologies, 15 January 2022. [Online]. Available: <https://docs.unity3d.com/Manual/class-MonoBehaviour.html>. [Accessed 18 January 2022].
- [9] "Rigidbody," Unity Technologies, 15 January 2022. [Online]. Available: <https://docs.unity3d.com/Manual/class-Rigidbody.html>. [Accessed 18 January 2022].
- [10] "Euler (gimbal lock) Explained," Youtube, 14 01 2009. [Online]. Available: <https://www.youtube.com/watch?v=zc8b2Jo7mno>. [Accessed 20 01 2022].

-
- [11] “Quaternions and 3d rotation, explained interactively,” Youtube, 28 October 2018. [Online]. Available: <https://www.youtube.com/watch?v=zjMuIxRvygQ>. [Accessed 30 December 2021].
- [12] “What is UDP?,” Cloudflare, 2022. [Online]. Available: <https://www.cloudflare.com/en-gb/learning/ddos/glossary/user-datagram-protocol-udp/>. [Accessed 20 01 2022].
- [13] “A-Level Comp Sci: Finite State Machine,” Youtube, 22 May 2017. [Online]. Available: <https://www.youtube.com/watch?v=4rNYAvsSkwk>. [Accessed 18 January 2022].
- [14] “UdpClient Class,” Microsoft, 2022. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.udpclient?view=net-6.0>.
- [15] “Serializable,” Unity Technology, 15 January 2022. [Online]. Available: <https://docs.unity3d.com/ScriptReference/Serializable.html>. [Accessed 20 January 2022].
- [16] “The safety system in the C# Job System,” Unity Technologies, 15 June 2018. [Online]. Available: <https://docs.unity3d.com/Manual/JobSystemSafetySystem.html>. [Accessed 14 January 2022].
- [17] “Tello User Manual,” Ryzerobotics, 2018. [Online]. Available: <https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20User%20Manual%20v1.4.pdf>. [Accessed 14 January 2022].
- [18] “PlayerPrefs,” Unity Technologies, 15 January 2022. [Online]. Available: <https://docs.unity3d.com/ScriptReference/PlayerPrefs.html>. [Accessed 17 January 2022].
- [19] “Time.timeScale,” Unity Technologies, 15 January 2022. [Online]. Available: <https://docs.unity3d.com/ScriptReference/Time-timeScale.html>. [Accessed 8 January 2022].
- [20] “Universal Render Pipeline overview,” Unity Technologies, 7 January 2022. [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@12.1/manual/index.html>. [Accessed 15 January 2022].
- [21] “Tonemapping,” Unity Technologies, 2020. [Online]. Available: <https://docs.unity3d.com/Packages/com.unity.render->

pipelines.universal@7.1/manual/post-processing-tonemapping.html.
[Accessed 14 January 2022].

- [22] “Motion Blur,” Unity Technologies, 4 December 2018. [Online]. Available: <https://docs.unity3d.com/2018.1/Documentation/Manual/PostProcessing-MotionBlur.html>. [Accessed 14 January 2022].
- [23] B. Helal, “How-can-a-quadcopter-yaw,” 2019. [Online]. Available: <https://www.quora.com/How-can-a-quadcopter-yaw>. [Accessed 29 December 2021].

List of Figures

Figure 2.1: 3 Axis of the drone [1]	2
Figure 2.2: How four rotors control the drone	3
Figure 2.3: Forces of a flying drone [3]	4
Figure 2.4: Tello Drone [5]	4
Figure 2.5: Tello Simulator screenshots	7
Figure 2.6: Screenshot of the DJI Flight Simulator [7]	9
Figure 2.7: Rigidbody component in Unity	11
Figure 2.8: How Quaternions interpret 270°	12
Figure 2.9: example of a state diagram [13]	13
Figure 3.1: different forces acting on the drone	17
Figure 3.2: Brief calculation of the forces in Figure 3.2	17
Figure 3.3: Different velocity require a different tilt angle	18
Figure 3.4: Visual display of the axis to rotate around	19
Figure 3.5: cut-out of the Rigidbody component on the drone	21
Figure 3.6: State machine of the drone	22
Figure 3.7: List of accepted commands	25
Figure 3.8: Default sky in Unity	27
Figure 4.1: Class Diagram	29
Figure 4.2: Example of a controller	30
Figure 4.3: Example of a controller	35
Figure 4.4: Code for flying in 3 dimensions by adding force	37
Figure 4.5: Code for rotating a set angle	38
Figure 4.6: Code which automatically balances the drone	39
Figure 4.7: Figure .: visualisation of “tilt” vector	39
Figure 4.8: function which returns the perpendicular vector from Figure 4.7	40
Figure 4.9: function which tilts 3° per 1m/s in the direction of flight	40
Figure 4.10: Tello Camp start screen	41
Figure 4.11: Pause menu	42
Figure 4.12: Heads-up Display	43
Figure 4.13: Connection UI	43
Figure 4.14: Debug UI	44
Figure 4.15: Instructions UI	44
Figure 4.16: Action UI	45
Figure 4.17: Speed UI	45
Figure 4.18: Drone model	46
Figure 4.19: Surroundings	47
Figure 4.20: Ring	48
Figure 4.21: Pillar	48
Figure 4.22: Stairs	49
Figure 4.23: Pipe	49

Figure 4.24: Left - after processing, right -before processing

50

I. Appendices

A. Git Repository

The whole Tello Camp including this documentation and builds for Windows, macOS and Linux can be found in the following git repository on the FHNW GitLab server:

<https://gitlab.fhnw.ch/iit-projektschiene/hs21/ip5-tello-drone>

A.1 Contents

A.1.1 Documentation

Subfolder: /documentation

Contains this documentation as .pdf file.

A.1.2 Builds

Subfolder: /tellocamp_builds

Contains the latest build for Windows, macOS (Intel & Apple Silicon) and Linux.

A.1.3 Unity Project

Subfolder: /tellocamp

Contains the whole unity project with all the assets.

Statement of Honesty

We hereby declare that we have written this document on our own, without the help of third parties and only in use of the sources indicated.

Brugg-Windisch, January 21, 2022

Maarten de Laat



Pascal Hostettler