

# UNIVERSITY OF TWENTE.

Digital Twins: simulation environment for smart applications  
and devices

Group 10 Project Report

April 19, 2024

STUDENT NAMES	STUDENT NUMBERS
David Hesthaven	2847825
Junjun Wu	2899108
Kevin Frohbös	2742608
Zhonghui Wen	2740613
Aleksii Kyryk	2488671
TCS	Module 11

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Domain Analysis</b>	<b>5</b>
2.1	Introduction to the Domain	5
2.2	General Knowledge of the Domain	5
2.3	Client, Users, and Interested Parties	5
2.4	Software Environment and Initial Situation	5
<b>3</b>	<b>System Proposal</b>	<b>6</b>
3.1	Planning	6
3.1.1	Initial Timeline	6
3.1.2	Actual timeline	6
3.1.3	Timeline reflection	6
3.1.4	Deliverables	6
3.1.5	Deliverables reflection	7
3.2	Responsibilities	7
3.2.1	3d modelling	7
3.2.2	Interconnection	7
3.2.3	Simulator building	7
3.2.4	Responsibilities reflection	7
3.3	Testing	7
3.3.1	Testing reflection	7
3.4	Reporting	8
3.4.1	Reporting reflection	8
3.5	Procedures	8
3.5.1	Communication	8
3.5.2	Task management	8
3.5.3	Procedures reflection	8
<b>4</b>	<b>Requirements</b>	<b>9</b>
4.1	Requirements Reflection	10
<b>5</b>	<b>Design</b>	<b>11</b>
5.1	Use Cases	11
5.2	Component Diagram	11
5.3	Operation sequences	12
5.4	Phone App Design	13
5.4.1	ConnectionActivity	13
5.4.2	Communication	14
5.4.3	DroneController	14
5.4.4	Message	15
5.4.5	Controls	15
5.5	Simulator Design	16
5.5.1	RigidBodyManager	16
5.5.2	DroneController	16
5.5.3	InputManager	16
5.5.4	IEngine	17
5.5.5	DroneEngine	17
5.5.6	Message	17
5.5.7	Controls	17
5.5.8	DroneControl	17
5.5.9	UIManager	18
5.5.10	DroneMover	18
<b>6</b>	<b>Test Planning</b>	<b>20</b>
6.1	Approach	20
6.1.1	Schedule	20

<b>7</b>	<b>Implementation</b>	<b>20</b>
7.1	Android App . . . . .	20
7.2	Desktop App . . . . .	21
7.2.1	Open-Source components . . . . .	22
7.2.2	Digital Twin component . . . . .	22
7.3	Azure WebPubSub . . . . .	23
7.4	Changes and Challenges during Development . . . . .	23
7.5	Notes on expansion . . . . .	24
<b>8</b>	<b>Tests</b>	<b>25</b>
8.1	Unit Tests . . . . .	25
8.2	Python Mocking . . . . .	25
8.3	System Tests . . . . .	25
8.4	Usability Tests . . . . .	28
<b>9</b>	<b>Evaluation</b>	<b>28</b>
9.1	Risks . . . . .	28
9.2	Ethics . . . . .	28
9.3	Future Development Paths . . . . .	29
9.3.1	Possible improvements . . . . .	29
9.3.2	Possible spin-off projects . . . . .	29
<b>10</b>	<b>Conclusion</b>	<b>30</b>

# 1 Introduction

As the development of smart devices advances further and further, the concept of the Internet of Things (IoT) becomes more and more important to everyday life. In order to research this field, the University of Twente has recently created the IoT CyberLab, whose goal it is to create an advanced Internet of Things utilizing several smart devices with the long term goal of assisting various projects to aid in the research of digital twins and cyber-security. As this is a new lab, foundations must be laid in order to have meaningful research results in the long term, and laying some of these foundations is the goal of this design project.

In particular, we sought to create a digital twin for a DJI Mini 2 SE drone, such that when the drone is connected to the digital twin, the twin will mimic the movements of the real device. While the actual use cases and objectives of the twin are quite limited, the true goal of the project is to lay a good foundation, with the code being thoroughly documented, to allow future research groups to easily extend the twin into becoming a more complex, and hence more viable to research piece of software.

In the following report we will be going through the process of designing this digital twin. Section 2 will revolve around discussing the definition and value of a digital twin and the Internet of Things. Section 3 will cover the initial proposal created for our design, alongside what adjustments we made, and how well we achieved the goals and plan set out in our proposal. Section 4 will look at the requirements our system while Section 5 will cover the system design. Section 6 will discuss our test plan, and how testing worked out over the course of the module. Section 7 will go into detail about how we implemented the various parts of the system laid out in Section 5. Section 8 goes in detail about the various tests that we did. Section 9 will be an evaluation and a reflection on where we succeeded and failed in our goals, and Section 10 will yield a final conclusion.

## 2 Domain Analysis

### 2.1 Introduction to the Domain

The domain in which the system is introduced involves the creation of a digital version of a drone, created through unity, that receives sensor data from the physical drone, including positional data, and the digital drone should reflect any movement made by the physical one. Thus, the system should act as a viable foundation for any future digital twins of the drone, as it should model as basic movement and sensor data.

### 2.2 General Knowledge of the Domain

The IoT-Cyberlab is a new lab created at the University of Twente. Its primary purpose is to aid in the research of digital twins. However, while being extremely useful digital twins are quite complex. This complexity obviously takes time to develop, and as a result, due to the shorter term project lengths that take place at the UT, it is difficult to create high quality specialized digital twins within the usual time-span. The IoT-Cyberlab hopes to resolve this issue by developing high quality digital twins that fulfill all basic requirements to be a digital twin, so that interested parties could take this foundation, and specialize their twin in such a way to provide useful research results even within a shorter time frame. Our project is a first step in developing such a twin, specifically for a drone.

### 2.3 Client, Users, and Interested Parties

Our Client is also our supervisor, João Moreira, who is the head of the IoT-Cyberlab, but this product is intended to be used by all members of the lab, whether that be other professors like him, or students interested in the topics being researched by the lab.

There also exist additional interested parties, in the form of any students or research groups that are interested in researching the usage of the digital twin of a drone, as our model would act as the initial foundation upon which more complex systems could be built. Thus, interested parties include future students, alongside the SCS research group as a whole, and the CAES research group, both of which are interested in researching the Internet of Things and cyber-physical systems, which is the topic our project falls into.

### 2.4 Software Environment and Initial Situation

In our initial meeting with our client, it was made clear that we could use any kind of language we wished, as our product is a foundational one that will be built upon, however, it was pointed out that they already possessed a basic gitlab for use in developing the drone, and most of the code in there was written in python, so we should keep that in mind. More details about our eventual decisions regarding design and languages can be found in section 6.

## 3 System Proposal

In the following section we will be going through the most critical aspects of our initial proposal and analyzing how well we managed to reach those goals and what changed throughout the project.

### 3.1 Planning

#### 3.1.1 Initial Timeline

- (26/2) Handing over the Project proposal to the Stakeholder.
- (1/3) Reflecting and improving on the proposal with the stakeholder. Deciding on the technologies to be used.
- (10/3) Specifying and implementing the test plan.
- (12/3) First MVP demo to stakeholder.
- (24/3) Finishing the first draft of the Design Report.
- (8/4) Presenting the final product.

#### 3.1.2 Actual timeline

- (26/2) Handing over the Project proposal to the Stakeholder.
- (1/3) Reflecting and improving on the proposal with the stakeholder. Deciding on the technologies to be used.
- (10/3) Specifying the test plan.
- (10/3 - 10/4) implementing the test plan.
- (4/4) First MVP demo to stakeholder
- (8/4) First draft of the Design Report.
- (10/4) Reflect on the feedback for the design report
- (15/4) Finalizing the poster.
- (16/4) Final Testing of product.
- (17/4) SCS Chair presentation.
- (18/4) Poster presentation.
- (19/4) Final hand-in

#### 3.1.3 Timeline reflection

Overall, while we were able to meet the initial deadlines we set, the later ones were consistently missed, showing that we were unable to properly predict the amount of time and work needed for those tasks. The primary reason for this was simply our lack of understanding the complexities in developing a digital twin and the fact that this project was on a scope that we were unfamiliar with, leading to improper estimations. Despite this, we still managed to complete all tasks with time to spare on an effective timescale. This implies that we were overly optimistic with our initial plan, but were still able to complete our tasks in an orderly and timely fashion.

#### 3.1.4 Deliverables

- A software application collaboration between a phone and a desktop app to connect and use a digital twin of the drone.
- A README explaining the setup and use of the system
- A Design Report detailing all functionalities and describing their implementation
- A set of tests documented by a test plan (part of the design report).
- A poster

### **3.1.5 Deliverables reflection**

We were able to finish all deliverables. The software application was further split into a mobile app to allow controlling the drone using a socket and a unity simulation that communicates with the app.

## **3.2 Responsibilities**

### **3.2.1 3d modelling**

- Design or find 3d-models of the drone.
- Design 3d-models of the laboratory space and of an open space (GPS must be usable there) on campus

### **3.2.2 Interconnection**

- Develop an Android application that uses the DJI mobile SDK to broker data between drone and PC application
- Develop interfaces and procedures to setup the connection of a drone to the mobile/Desktop application
- Define an interface for the simulator team to give instructions to the drone and read data from it when communicating with the mobile app

### **3.2.3 Simulator building**

- Develop a model of Drone Kinematics and Dynamics
- Design an interface to control the movement of the drone model inside the 3d-environments generated  
Note: This includes coding control software (PID, etc)
- Design procedures to adjust the drone status (position, power, connectivity) based on outside data.

### **3.2.4 Responsibilities reflection**

Almost all tasks were able to be completed, though not perfectly. For example, while the open space was developed, we did not have the time to test whether it was functional with the usage of GPS. Additionally, building a kinematics and dynamics model was not necessary since we adapted a previous drone simulation. Motion control was also adapted from this.

## **3.3 Testing**

- Develop automated tests for communication and authentication
- Manually test usability using test scenarios
- Test installability with the help of an unbiased subject
- Document testing results

### **3.3.1 Testing reflection**

Due to the nature of the application most automated testing could not be done as originally envisioned. The manual system testing and the usability test consequently had to be carried out in significant detail.

## 3.4 Reporting

- Identify the need for and ensure the use of authentication, encryption and other security measures
- Create a README containing the setup procedure as well as giving an overview of basic functionalities
- Document functionality and relevant construction details of the different subparts of the project

### 3.4.1 Reporting reflection

Working on the various reports of the report started later in the project's timeline than would've been optimal, however we still managed to complete all required reports on time. Other aspects of documentation, like documenting the code and the README, were completed significantly earlier, during the process of writing code. Security measures are proposed in the "Future Development Paths" but not implemented.

## 3.5 Procedures

### 3.5.1 Communication

Weekly meetings (physical) were held with the stakeholder to give progress updates and acquire feedback.

Bi-weekly meetings were held to address problems and keep everyone up to date with overall progress on different tasks. Team meetings take place via discord or physically.

Meeting-dates are planned at the latest at the preceding meeting.

### 3.5.2 Task management

- Tasks will be assigned to team members based on their expertise and preferences, these tasks will be managed using Trello, a project management tool.
- Progress on Trello will be tracked and discussed during the bi-weekly meetings to make sure everything is proceeding smoothly.
- Adjustments to task priorities will be made collaboratively, to ensure that the project stays on track and everyone knows what should be finished first.

### 3.5.3 Procedures reflection

Communication did not present a major issue. The meetings with the stakeholder as well as the bi-weekly meetings took place and all members were present physically or via discord.

The use of Trello was absent as on the initial day of setting up our Trello board, the website was having difficulties and thus we worked without a project management tool. We instead relied on several discord channels to organize our work.



## 4 Requirements

The following section has us going into depth on what our requirements for our final product would be. We utilized the MoSCoW system, splitting the requirements into Must, Should, Could, and Won't categories, alongside an additional category for quality.

Must:

- Teachers and students should be able to move around the drone model in a virtual version of an IoTCyberlab laboratory space.
- Teachers and student should be able to access sensor values (camera feed, accelerometer/GPS readouts).
- Teachers and students should be able to use a phone app to control the physical drone, at least moving up and down.
- Teachers and students should be able see movement from the physical drone be reflected in movement by the digital twin.

Should:

- Teachers and Students should be able to access a list of all sensors and actuators with their on/off-status.
- Teachers and students should be able to control the movement of the real-world drone by controlling the drone model.
- Teachers and students should be able to control the movement of a DJI mini 2 drone - Unity model using keyboard and mouse.
- Teachers and students should be able to view the battery status of devices in the digital twin interface.
- Teachers and students should be able to use a phone app to control the physical drone, including rotation.
- Teachers and students should be able to move around the drone model in a virtual version of a free space on campus (such that the GPS functionality of the drone can be used)

Could:

- Administrators should have access to log files of all data transmissions
- Teachers and Students should be able to write and run programs that control the vehicles of the digital twin environment using the communication interface employed in the digital twin software.
- Teachers and students should be able to simulate wind conditions in the digital twin environment.
- Teachers and students should be able to use the phone app to control the physical drone, including lateral movement.

Won't:

- Teachers and students should be able to carry out data visualization using sensor readings (ideally via Matlab integration).
- Administrators should be able to add custom models of other environments

Quality:

- The system should be modular and documented in sufficient detail to allow easily adding new devices or extra sensing capabilities for existing devices
- All functionalities should be tested
- Communication should follow standard security practices

- Any user accessing the vehicles must be authenticated
- The interfaces should be easy to understand for someone with a background in Engineering or Computer Science
- Latency of the communication between model and device should be small enough to not be noticed by a user.
- The used unity designs should mirror the look of the modelled objects
- The Behavior of virtual models should convincingly mirror that of real-world objects

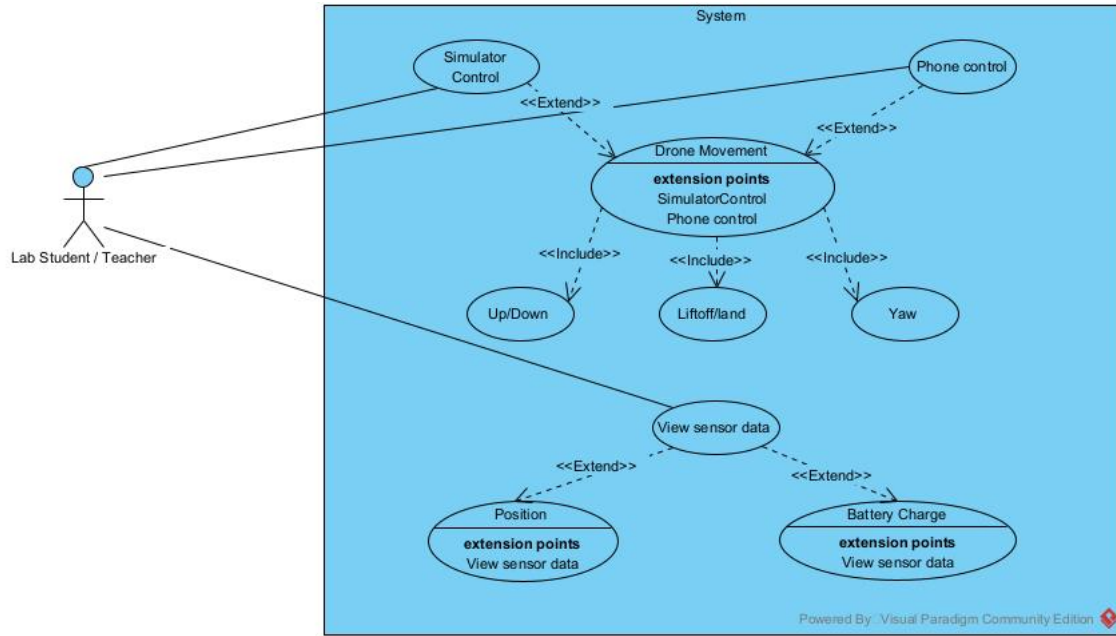
#### **4.1 Requirements Reflection**

We managed to accomplish a number of requirements, though not all. Within the "Must" category, we fulfilled all requirements with one small exception, the implementation of the camera sending data to the twin. This could not be implemented due to a firmware issue making it impossible to send a newly taken photograph, but all other requirements were fulfilled. Among the "Should" requirements, they too were almost completely fulfilled, barring the first, in which users could see a list of all available sensors. This was never implemented since we only had one device (with very basic sensing capabilities) to test on. Similarly, none of the "Could" requirements were fulfilled either due to a lack of time. With regards to the Quality requirements, we fulfilled all of them except those having to do with security and authentication, which similar to the above, were not done due to time constraints and low priority.

## 5 Design

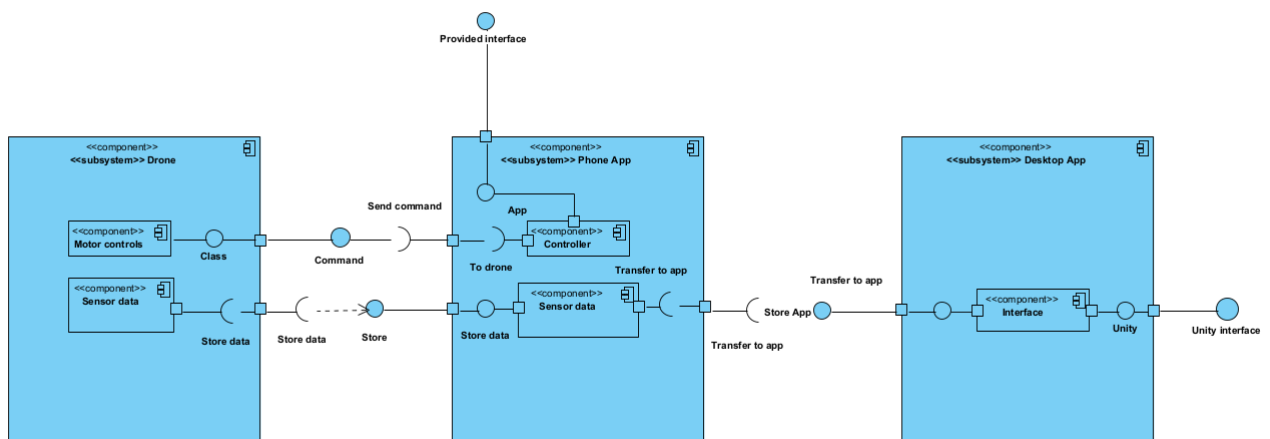
We chose to design our project such that it was split into two primary parts. On one side is the Unity component hosted on a desktop/laptop. Its code is written in C#. On the other side there is an android app, which controls the physical drone, and forwards data and instruction between drone and simulator. The communication between the two is handled through a message broker. The app is written in Kotlin. The following section will show the various diagrams developed for the system, alongside elaborations of those diagrams.

### 5.1 Use Cases



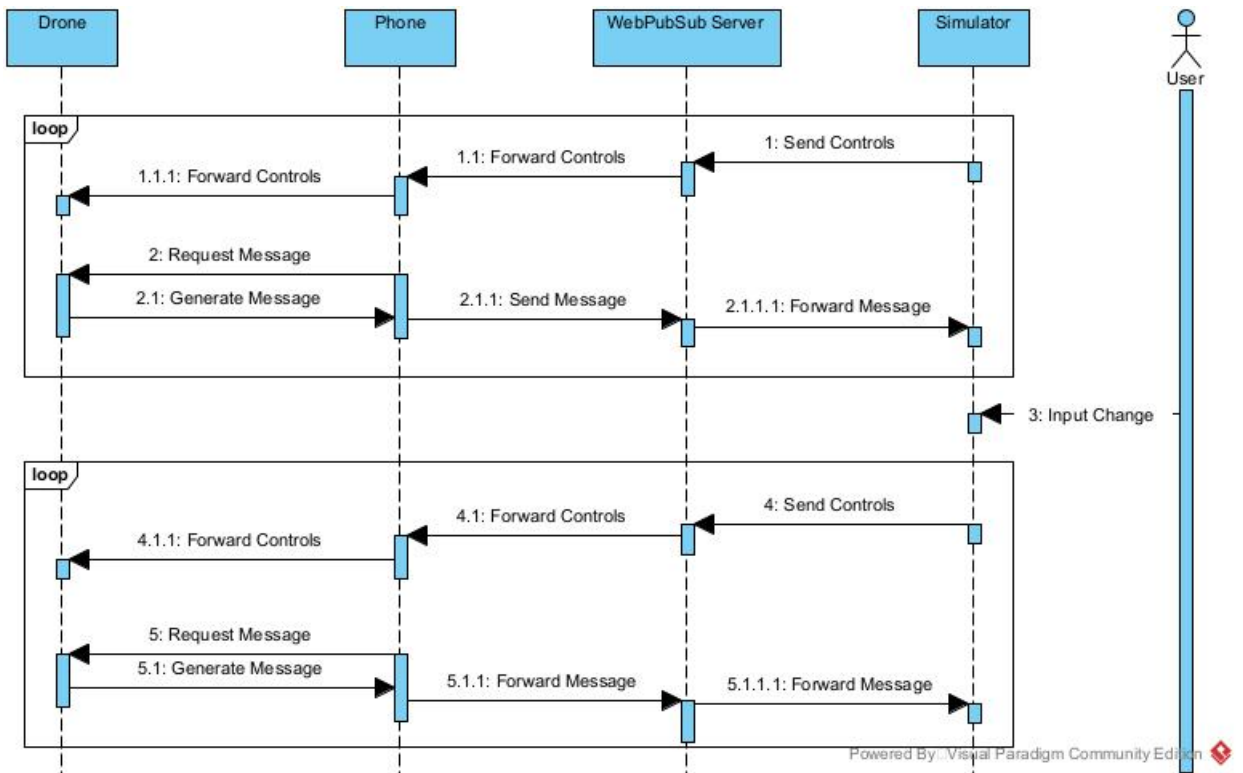
The main 2 use cases for the application are viewing information about the drone and controlling it. Control can be performed either via the simulator or the phone.

### 5.2 Component Diagram

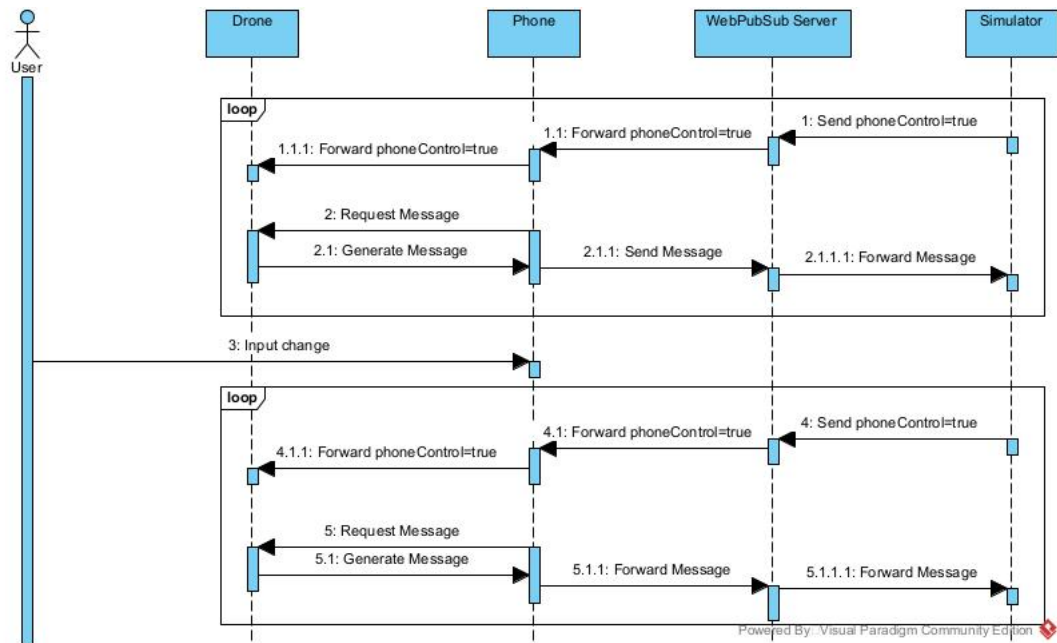


This diagram describes all the components which are accessed throughout the use of our app. There are 2 interfaces which the user can access. The provided interface is when the user uses the phone app and the sequence of actions is described as in the diagram, the unity interface is for when the user tries to control the drone through the desktop app.

### 5.3 Operation sequences

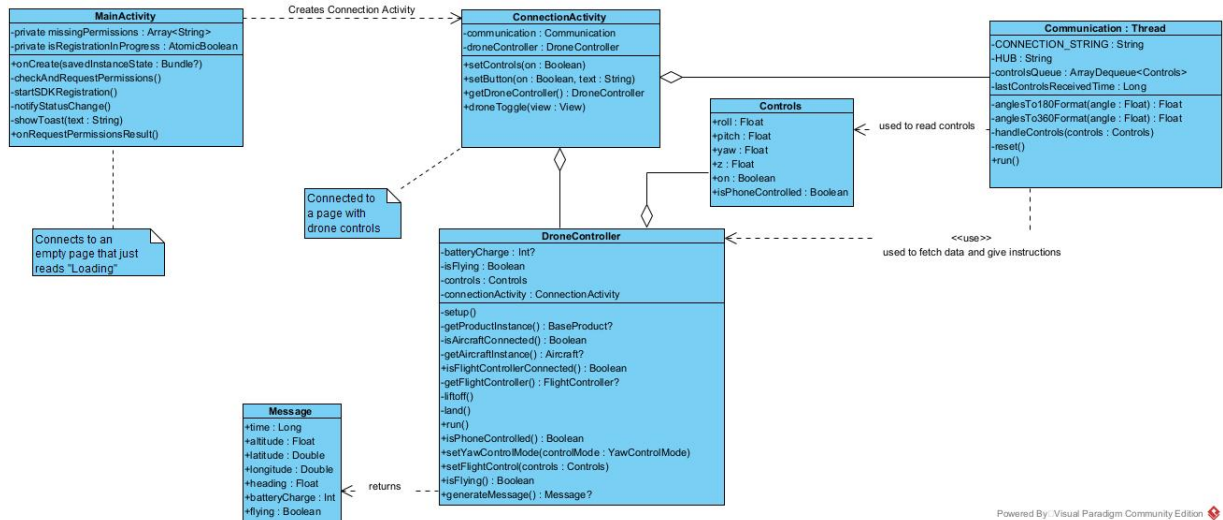


This diagram shows what the regular operations cycle when the simulator is in control of the drone.



This diagram shows what the regular operations cycle when the phone is in control of the drone.

## 5.4 Phone App Design



The app consists of 2 Views. There is a loading screen tied to the MainActivity class. The code in this class is taken from a sample project ([https://github.com/godfrey Nolan/DJITutorialsKotlin/blob/main/1-Registration/app/src/main/java/com/riis/kotlin\\_importandactivatesdkinandroidstudio/MainActivity.kt](https://github.com/godfrey Nolan/DJITutorialsKotlin/blob/main/1-Registration/app/src/main/java/com/riis/kotlin_importandactivatesdkinandroidstudio/MainActivity.kt)) and is responsible for connecting the App to the DJI mobile SDK. Once this is successful it redirects the user to a new View. The function `.onRegister()` inside of `.startSDKRegistration()` performs this redirect in case of success.

```

//checking registration
override fun onRegister(error: DJIError?) {
    error?.let { # DJIError
        //If registration is successful, start a connection to the DJI product.
        if(error == DJISDKError.REGISTRATION_SUCCESS){
            showToast( text: "Register Success")

            DJISDKManager.getInstance().startConnectionToProduct()
            //If registration is unsuccessful, prompt user and log the registration error

            // Redirect to Selection screen
            val intent = Intent(applicationContext, ConnectionActivity::class.java)
            startActivity(intent)
        }else{
            showToast( text: "Register sdk fails, please check the bundle id and network con
        }
        Log.v(TAG, error.description) *let
    }
}
    
```

### 5.4.1 ConnectionActivity

The view connected to this displays a Ltoff/Land-button and 2 seekbars for yaw and height control. This class contains an object for drone-control and one for communication. Its task is to pass on commands from the phone UI to those classes and vice-versa.

`.onCreate()` sets up the listeners for the 2 seekbars. It also ensures the UI is deactivated by default and initiates the communication and droneControl-threads.

`.setControls(on:Boolean)` turns the yaw and height seekbars on and off. when turned off seekbars are reset to the center position.

`.setButton(on:Boolean, text:String)` turns the takeoff/land button on and off, also allows setting the button text.

`.getDroneController()` is used by Communication to access the droneController property of the Connection Activity. This is needed for handling controls-messages.

`.droneToggle(view:View)` is attached to the liftoff/land-button. When phone control is enabled this button relays liftoff/land signals to the droneController. It further adjusts the UI.

### 5.4.2 Communication

This class is responsible for all communication with the Desktop app. It extends Thread so that it can run concurrently with the ConnectionActivity. The class takes the ConnectionActivity object as an argument in its constructor so it can access the DeviceController.

.run() is overridden from the Thread-class. It establishes a connection with the WebPubSub service and attaches a listener to enqueue incoming messages into controlsQueue. As long as the connection is up and a drone is connected, every 200ms, it passes on a controls-message to .handleControls and sends out messages constructed via droneController.generateMessage(). In case of disconnection (no controls-message in 5s or a closed websocket) the method runs .reset().

.handleControls(controls:Controls) is responsible for processing controls-messages. In case it registers a change in isPhoneControlled it adjusts the UI and the droneController. Notably we use velocity control for yaw during phoneControl and position control otherwise. If phoneControl is disabled it performs angle conversion using .anglesTo180Format() and passes the controls-object to the droneController.

.reset() this method resets the UI into its default state. It similarly instructs the droneController and drone to return to their original configurations.

.anglesTo360Format() this function converts angles of the format -180 to 180 to 0 to 360. (Note that 0 degrees correspond to the same position for both formats)

.anglesTo180Format() this function converts angles of the format 0 to 360 to -180 to 180. (Note that 0 degrees correspond to the same position for both formats)

### 5.4.3 DroneController

This class is responsible for setting up a connection with the drone and passing information to and from it.

.setup() first verifies whether a FlightController-object can be accessed. If so VirtualStickMode is enabled allowing remote control via the SDK. The coordinate system is set to body (see [https://developer.dji.com/mobile-sdk/documentation/introduction/flightController\\_concepts.html](https://developer.dji.com/mobile-sdk/documentation/introduction/flightController_concepts.html)). While roll, pitch and yaw are set in angle form, for vertical motion, velocity is controlled instead. By default the drone will not take-off without a GPS signal. This setting is changed. Likewise the drone asks confirmation if a landing area is not found to be clear. This setting too is disabled. A callback is configured to update the batteryCharge-property when the battery-state changes.

.getProductInstance() attempts to return the product the SDKManager has (any connected device). Taken from <https://github.com/dji-sdk/Mobile-SDK-Android/blob/master/Sample%20Code/app/src/main/java/com/dji/sdk/sample/internal/controller/DJISampleApplication.java>.

.isAircraftConnected() verifies .getProductInstance() returns an aircraft. Taken from <https://github.com/dji-sdk/Mobile-SDK-Android/blob/master/Sample%20Code/app/src/main/java/com/dji/sdk/sample/internal/controller/DJISampleApplication.java>.

.getAircraftInstance() attempts to return an instance of the connected aircraft. Taken from <https://github.com/dji-sdk/Mobile-SDK-Android/blob/master/Sample%20Code/app/src/main/java/com/dji/sdk/sample/internal/controller/DJISampleApplication.java>.

.getFlightController() attempts to return the flightcontroller of the connected aircraft.

.isFlightControllerAvailable() returns whether an Aircraft FlightController can be accessed.

.run() is overridden from Thread. As long as no FlightController is available this thread keeps the UI deactivated. When connected .setup() is run. The UI is enabled (in case of phone control). A secondary loop is initiated which stays active for as long as there is a FlightController. The loop

sends instruction to the virtualSticks of the FlightController as needed. The loop must iterate at a frequency between 5 and 25Hz to ensure instructions are regularly relayed to the drone. If this does not happen the drone stops manoeuvring and hovers in place. Position and height adjustments are only made if the controls-property requires it, otherwise no instructions are sent to use the default hover of the drone which is more stable than actively instructing it to stay in place. During simulator control the loop attempts to match the angle and height in controls for which it has tolerances of +-20cm and +-3 degrees respectively. During phone control the controls-property determines the yaw- and z-velocities instead of positions. An extra stable-variable was added for simulator control. It ensures that when the drone reaches a position within the height- and angle-tolerances it keeps manoeuvring for one more iteration. This way we avoid the drone perpetually flipping between a stable and unstable state. It must be noted that when the simulator is in charge we use velocities to change height but fixed angles to change yaw. This is because during experimentation we found that the drone sometimes struggles to achieve the correct height when given a fixed height values (likely due to fluctuations in height measurements) and therefore handles velocity control better. The drone handles being given a fixed target angle very well.

.isPhoneControlled() informs whether the phone is in charge of the drone or not.

.liftoff() initiates a take-off and sets isFlying to true if the flightController is available.

.land() instructs the aircraft to land and sets isFlying to false if the flightController is available.

.setFlightControl() is used to update the controls property. If controls.on and isFlying do not have the same truth-value .liftoff() or .land() is called to adjust this.

.setYawControl(controlMode:YawControlMode) is used to choose between angle and angular-velocity control. This is needed since while phone control uses angular velocity the simulator sends specific angles to match.

.isFlying() returns the value of the isFlying property.

.generateMessage() first verifies the flightController is set. If so it constructs a Message object using the properties of the flightController a string of the current time and the batteryCharge-property. Notably it does not allow negative height and sets all negative height values to 0.

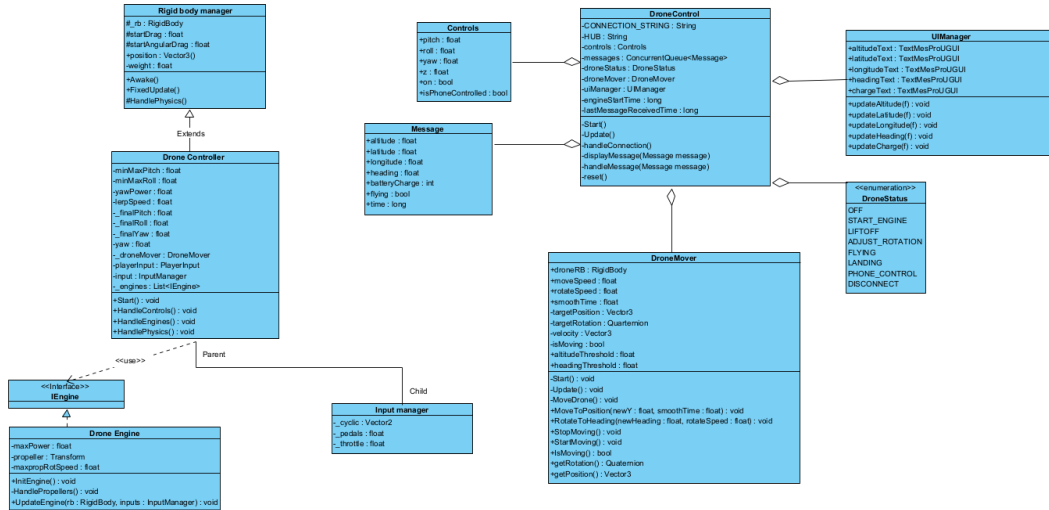
#### 5.4.4 Message

Contains data sent by the drone. The flying-property corresponds to flightController.state.isFlying not to the isFlying-property in DroneController.kt. Unlike the property state.isFlying it is not immediately true when initiating liftoff but only after the liftoff process has already completed.

#### 5.4.5 Controls

Contains instructions given to the droneController. Notably the z and yaw value correspond to target height/angle when phoneControl is off and target velocities when phone control is on.

## 5.5 Simulator Design



This is a diagram of the programming classes utilized in the desktop app of the system. It is split into two parts, with the left half handling the digital drone, and the right half handling phone and user interaction.

### 5.5.1 RigidBodyManager

An abstract class that manages the rigidbody object of a game object. It extends MonoBehaviour and implements the base methods.

.awake() loads the script instance, retrieves the rigidbody component and sets the mass, drag and angular drag.

.fixedUpdate() called every fixed framerate frame. If rigidbody has been set, call handlePhysics().

.handlePhysics() - virtual method implemented by the extension of the RigidBodyManager (see below).

### 5.5.2 DroneController

This class is an extension of RigidBodyManager and is responsible for controlling the movements of the drone within the digital environment based on the user input.

.start() is called before the first frame update. It initializes the input manager, gets the engines attached to the drone, and assigns references to the DroneMover and PlayerInput components.

.handlePhysics() handles the physics of the drone. It calls HandleEngines() and depending on whether the drone is moving also calls HandleControls()

.handleControls() handles the user input controls for the drone. It calculates the pitch, roll, and yaw based on the user input and applies the calculated values to the drone's rotation using interpolation.

.handleEngines() handles the engines attached to the drone. It iterates through each engine and calls the UpdateEngine method for each engine to update its behavior.

### 5.5.3 InputManager

This class is responsible for managing the input to control the drone. It updates its values whenever there is new input.

.onCyclic() is an event handler for the cyclic input. It receives the input value and assigns it to the cyclic variable.



.onPedals() is an event handler for the pedals input. It receives the input value and assigns it to the pedals variable.

.onThrottle() is an event handler for the throttles input. It receives the input value and assigns it to the throttle variable.

### 5.5.4 IEngine

This interface defines the engine functionality of the drone.

.initEngine() must initialize the engine.

.updateEngine() must update the engine based on the provided inputs.

.handlePropellers() must handle the rotation of the propellers depending on whether it is on the ground or not. If it's on the ground, speed goes to 0, otherwise it sets it to go to the max rotation speed. If propeller isn't initialized, we do nothing.

### 5.5.5 DroneEngine

This class implements the IEngine interface, and its defined methods.

### 5.5.6 Message

Contains data sent by the drone to the simulator.

### 5.5.7 Controls

Contains instructions from the Simulator to the drone.

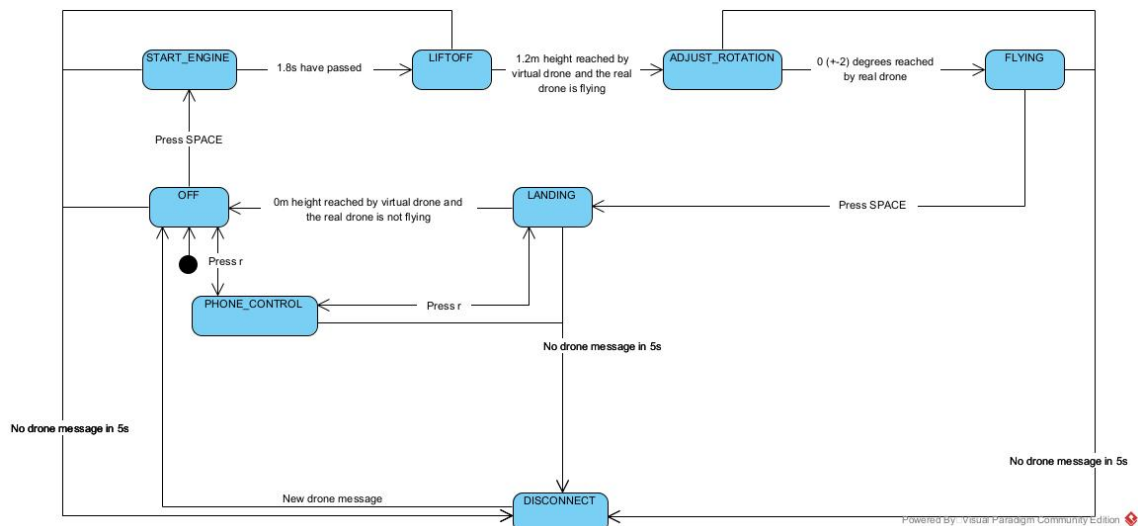
### 5.5.8 DroneControl

This class handles Communication with the drone. It also controls user inputs.

.Start() performs instantiation of relevant properties. This function starts the thread that runs .handleConnection() it further runs droneMover.StartMoving() which deactivates user steering of the drone.

.displayMessage(Message message) uses the uiManager to print altitude, latitude, longitude, heading and charge to the screen.

.Update() this method is run every frame. It is responsible for managing the droneStatus for which it processes user inputs and messages. The user inputs handled directly by this class are the SPACE- and "r"-key. The different states handled are explained in the below state-machine diagram.



When control is handed to the simulator we run droneMover.stopMoving(). This call lets the

droneMover hand over control to the keyboard. Conversely when switching to a state where the keyboard should not be in control droneMover.startMoving() is called. This allows controlling the virtual drone via calls to droneMover. To process messages .Update() runs .handleMessage(). If no message was queued instead a check is made whether 5s have passed since the last message and if so .reset() is called. While in the FLYING-state this function updates the controls-property with the virtual-drone height and heading. This property is used to instruct the physical drone in .handleConnection().

.handleMessage(Message message) runs .displayMessage() and processes message content in accordance with the above state machine. Notably if PHONE.CONTROL is active this uses droneMover to mirror the position of the drone described in message.

.displayMessage(Message message) adjust the text displaying: altitude, latitude, longitude, charge and heading.

.reset() sets the status to DISCONNECT and the controls sent and the values displayed to the defaults (drone off, phone-control off, altitude=0 etc). It puts the droneMover in charge of the virtual drone and moves it drone to its initial position.

.handleConnection() sets up a WebSocketClient to subscribe to the WebPubSubService. The client is configured with a callback that puts any received message in the messages-queue if the message contains the term "batteryCharge". This way we separate Message-objects intended for the Simulator from Controls-objects intended for the drone. The client will also attempt a reconnect every 5s if no message was received. It does so separately from the .reset() that is performed in .Update() after 5s of no message. After setting up the client the function enters a loop only broken if the running-property is false. In the loop it sends out the controls-property as a JSON string every 200ms.

### 5.5.9 UIManager

This class is responsible for updating the UI elements based on the inputs that are given.

.updateAltitude() changes the altitude text field with the given altitude value.

.updateLatitude() changes the latitude text field with the given latitude value.

.updateLongitude() changes the longitude text field with the given longitude value.

.updateHeading() changes the heading text field with the given heading value.

.updateCharge() changes the charge text field with the given charge value.

### 5.5.10 DroneMover

This class is used by the DroneControl to move the digital twin to a (new) received position whenever a new message is received from the drone. Important to note is that in the simulation adapted the y-axis corresponds to the vertical.

.isMoving() returns a boolean indicating if the drone is moved by the DroneMover. Used to disable user input while the twin is receiving movement instructions from DroneController.

.Start() is called before the first frame update. Initializes position and rotation values. By default the droneMover is off.

.Update() is called every frame. If the drone is expected to move, calls .moveDrone()

.MoveDrone() sets the drone's trajectory based on current and targeted position/rotation values as well as intended rotation velocity and movement time.

.MoveToPosition(newY, smoothTime) sets the intended height of the drone. Important to note is that in this simulation y corresponds to height. smoothTime defines the time the drone should

take to reach the target position. Its has a default value of 0.3s.

.RotateToHeading(newHeading, rotateSpeed) sets the intended heading of the drone. The default rotateSpeed is 180 degrees/second.

.getRotation() returns the rotation of the drone. The format is a Quaternion.

.getPosition() returns the position of the drone. The format is a Vector3. Again it can be noted that its y-property corresponds to height.

.StartMoving() enables control of this class over the drone. It also disables motion-control via the keyboard.

.StopMoving() disables control of this class over the drone. It also enables motion-control via the keyboard.

## 6 Test Planning

For testing we chose to split our tests into three types: Unit Tests for testing smaller things, System Tests for testing the system as a whole, and Usability Tests for testing how easy it is for others to use our system, since the whole point is to create something that can be expanded upon easily.

### 6.1 Approach

Since most functions written have no return types and rely on an interplay of several hardware and software components it was found that Unit testing was not suitable throughout most of development. Instead systems test of different functionalities were used while writing components. One exception is communication with the simulator for which an automated python client was written. To test the overall understandability of the system a usability test with an external party was performed.

#### 6.1.1 Schedule

Individual system tests were continually added and repeated during development. During the development of the simulator a python script was written to simulate sending client data. The usability tests were developed throughout the design process and carried out on the 18th of April.

## 7 Implementation

As previously stated, we wrote the phone app in Kotlin, and the desktop app in C#. We will now cover their implementation in detail.

### 7.1 Android App

The desktop app was originally built based on <https://github.com/godfreynolan/DJITutorialsKotlin/blob/main/1-Registration>. Only parts of the tutorial had to be adapted however to build an app.

First an empty views activity was built using API 26 (not 24 due to WebPubSub) and Kotlin. In build.gradle (Module: App) the target SDK was lowered to 30 due to an error originating with the DJI-imports. Further the contents of "dependencies" were changed:

```
dependencies {
    implementation("com.dji:dji-sdk:4.16")
    compileOnly("com.dji:dji-sdk-provided:4.16")

    implementation("androidx.multidex:multidex:2.0.1")
    implementation("com.squareup:otto:1.3.8")
    implementation("com.google.android.material:material:1.5.0")

    implementation("androidx.appcompat:appcompat:1.4.1")
    implementation("androidx.core:core-ktx:1.7.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.3")
    implementation("androidx.recyclerview:recyclerview:1.2.1")
    implementation("androidx.lifecycle:lifecycle-extensions:2.2.0")
    implementation("androidx.annotation:annotation:1.3.0")
    implementation("com.jakewharton:butterknife:10.2.3")
    annotationProcessor("com.jakewharton:butterknife-compiler:10.2.3")
    implementation("androidx.core:core-ktx:1.7.0")
    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.0-native-mt")
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.0-native-mt")
}
```

Note: One of the dependencies "implementation("androidx.core:core-ktx:1.7.0")" appears twice in the original, this does not matter.

In gradle.properties, "android.enableJetifier=true" was added.  
An xml file was added with path: "app/src/main/res/xml/accessory\_filter.xml". The contents of the file are:

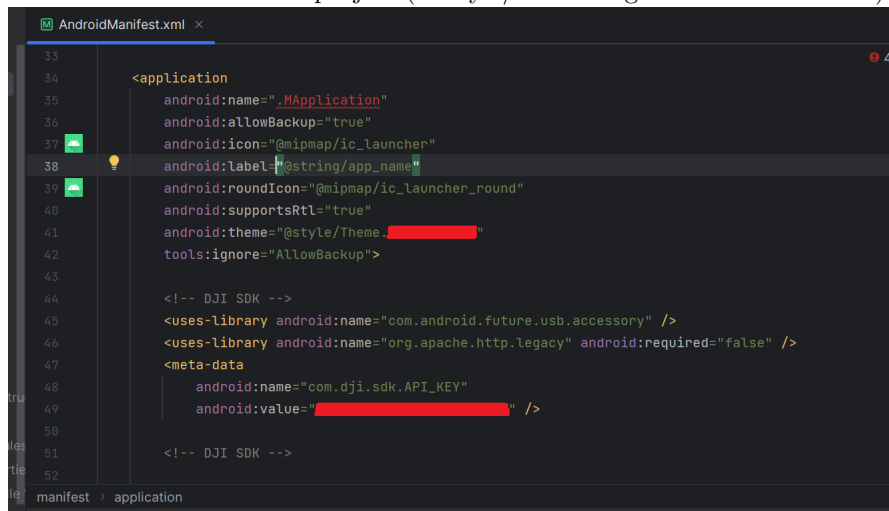
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <usb-accessory model="T600" manufacturer="DJI"/>
    <usb-accessory model="AG410" manufacturer="DJI"/>
    <usb-accessory model="com.dji.logiclink" manufacturer="DJI"/>
    <usb-accessory model="WM160" manufacturer="DJI"/>
</resources>
```

A file Kotlin class MApplication.kt was added in the same folder as MainActivity.kt. The contents of the 2 files were replaced with those of the files in [https://github.com/godfreynolan/DJITutorialsKotlin/blob/main/1-Registration/app/src/main/java/com/riis/kotlin\\_importandactivatesdk](https://github.com/godfreynolan/DJITutorialsKotlin/blob/main/1-Registration/app/src/main/java/com/riis/kotlin_importandactivatesdk) (except for the "using"-statements at the top). Inside .onRegister() inside .startSDKRegistration() in MainActivity.kt the following lines were added to redirect to the control-view:

```
// Redirect to Selection screen
val intent = Intent(applicationContext, ConnectionActivity::class.java)
startActivity(intent)
```

Further in the REQUIRED\_PERMISSIONS\_LIST at the top of the file the READ\_PHONE\_STATE and VIBRATE permissions should not be required.

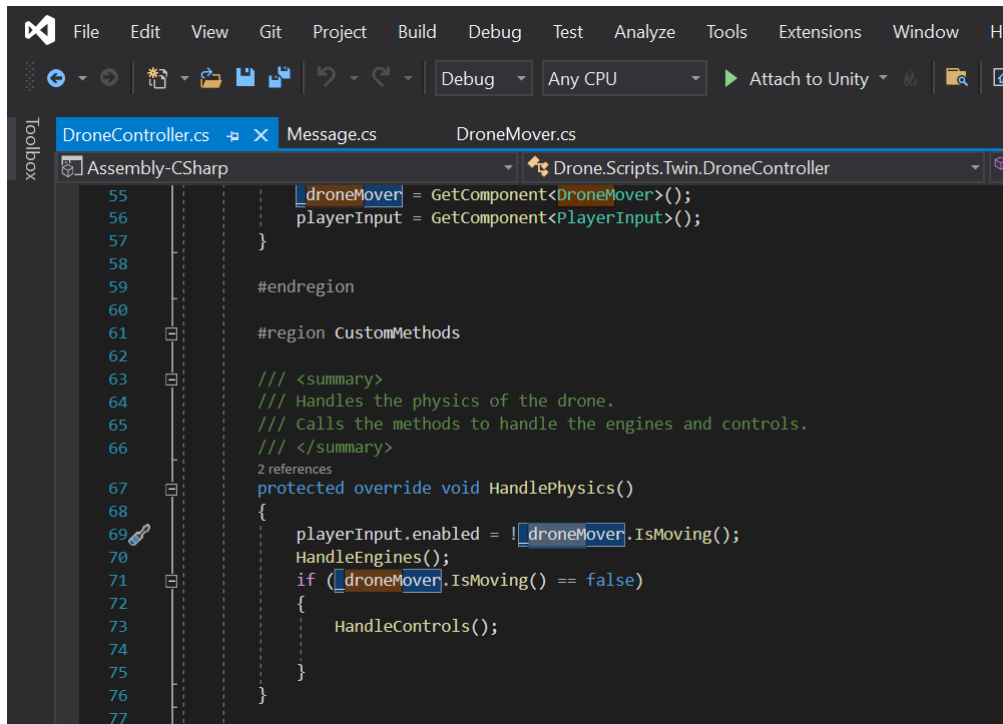
Inside AndroidManifest.xml the body of "manifest" was replaced with the contents of the "manifest"-tag in <https://github.com/godfreynolan/DJITutorialsKotlin/blob/main/1-Registration/app/src/main/AndroidManifest.xml>. An API-key was created for the application and used to replace the API key found in the file. Likewise the value under android:theme had to be replaced to match the name of the project ("@style/Theme.digitaltwin" in our case).



The READ\_PHONE\_STATE and VIBRATE permissions were removed.

## 7.2 Desktop App

The desktop app was originally built from <https://github.com/Shubhra22/DronePhysicsUnity/tree/master>. We left the code intact except for a modification of the DroneController class. A reference was added to the DroneMover class; we added a check whether DroneMover is currently moving the twin. If this is the case user inputs controlling drone movement are blocked. The check is made in the HandlePhysics-method of DroneController:



```
55         _droneMover = GetComponent<DroneMover>();
56         playerInput = GetComponent<PlayerInput>();
57     }
58
59     #endregion
60
61     #region CustomMethods
62
63     /// <summary>
64     /// Handles the physics of the drone.
65     /// Calls the methods to handle the engines and controls.
66     /// </summary>
67     2 references
68     protected override void HandlePhysics()
69     {
70         playerInput.enabled = !_droneMover.IsMoving();
71         HandleEngines();
72         if (_droneMover.IsMoving() == false)
73         {
74             HandleControls();
75         }
76     }
77
```

As previously stated, the desktop app, that implements the digital twin component of the project uses the Unity Engine. Within the Unity twin, as could be seen from the class diagram in the Design section, there are 2 parts of the system - the drone physics component with user input (implemented by DroneController, DroneEngine and InputManager classes) and the digital twin component (implemented by DroneControl, DroneMover, and UIManager).

### 7.2.1 Open-Source components

With advise and guidance of our supervisor, we were pointed to an open source implementation of the drone physics library for the Unity Engine. That was very useful as we therefore needed not to spend time on "re-inventing the wheel" and instead focus on the implementing the necessary components to achieve desired digital twin behaviours. We made very minor changes to the drone physics code-base as even though it serves as quite minimal foundation, it is enough to achieve the objectives of our project. Nevertheless, one of the important additions to the drone physics library was minor modification of the DroneController class. By adding a reference to the DroneMover, the DroneController can check whether the twin is being moved by instructions of the communicated drone's data. If so, we disable the user input in Unity to ensure that the user cannot "mess up" the representation of the drone's position in the Unity environment.

### 7.2.2 Digital Twin component

The centerpiece of the digital twin section is the DroneControl class. This class is responsible for handling the SPACE and "r"-key presses and handles all communication between phone and simulator. The details of how communication was implemented are elaborated on in the Azure WebPubSub section.

As described in the design, the DroneMover class is responsible for actually moving the rigid body of the drone in the Unity Engine. This class implements Unity's MonoBehaviour which means it has an Update() method that is called every frame. The Update() method is then essentially a loop that and the code within it runs many times a second. Therefore, the MoveToPosition() and RotateToHeading() don't actually move the rigid body but set the target values that are checked every frame. The implementation of MoveToPosition() method takes a newY value, and sets the target y value of the digital twin to the input newY value, again as only vertical movement was needed in our MVP. RotateToHeading() method takes a newHeading float value which is set to the target rotation y value. Lastly MoveDrone() uses the targetPosition and targetRotation and performs the movement on the rigid body using built in Unity Engine functionality. This method is called every frame by the Update() method. We use calibrated smoothTime and rotateSpeed values to move and rotate the twin smoothly, instead of it just suddenly "teleporting" to the target position and heading. A useful consideration for future work would be to extend the DroneMover

and DroneControl to enable smoother movement across a number of samples, which will potentially achieve a more synchronous perception between the real drone's movements and the twin.

Due to our solution being a proof of concept and a foundation for future projects, we did not focus on prettifying the UI. At the moment, every time the message is handled by DroneControl, the UI is updated by simply updating the UI text fields with the real drone data. That is the only functionality of the UIManager. Another extension suggestion would be to utilize open-source solutions as OpenStreetMap and extend UIManager to show more visual representation of data such as latitude and longitude.

### 7.3 Azure WebPubSub

To handle communication we used a Azure WebPubSub. Using a publish subscribe message broker makes it easier to communicate with multiple devices and generally simplified the communication setup and management a lot compared to initial designs which used Sockets. It also means that drone and simulator no longer need to be in the same network.

We elected to implement a cloud based message broker as that makes our application less complex than self-hosting. Since the university uses Microsoft services we opted to use Azure WebPubSub. A WebPubSub-service was chosen since it was relatively simple to use compared to "Event Grid" or "Service Bus".

The setup of the Azure service was done based on

<https://www.geeksforgeeks.org/microsoft-azure-messaging-with-azure-web-pubsub/>.

For the simulator our code was adapted from

<https://github.com/Azure/azure-webpubsub/tree/main/samples/csharp/pubsub>

while the App adapted from

<https://github.com/Azure/azure-webpubsub/tree/main/samples/java/pubsub>.

To install the simulator-dependencies needed for our project, we added Nuget for Unity to it. The installation was done using the .unitypackage as outlined in

<https://github.com/GlitchEnzo/NuGetForUnity>.

Using Nuget for unity "Azure.Messaging.Webpubsub v1.3.0" and "Websocket.client v4.185" were installed. The dependencies on the App side were "com.azure:azure-messaging-webpubsub:1.2.13" and "org.java-websocket:Java-WebSocket:1.5.6". To manage the errors resulting from the packages the following was added inside "android {...}" in the build.gradle.kts (Module:App)-file.

```
packagingOptions {
    resources.excludes.add("META-INF/INDEX.LIST")
    resources.excludes.add("META-INF/NOTICE.md")
    resources.excludes.add("META-INF/io.netty.versions.properties")
    resources.excludes.add("META-INF/LICENSE.md")
}
```

### 7.4 Changes and Challenges during Development

When we first started we had to decide how we wanted to implement the digital twin itself, with our supervisor recommending either Unity or Godot, however, due to Godot's limitations, we decided on utilizing Unity instead. Additionally, one of the big questions we had to answer was how to communicate from the phone to the desktop. The initial plan was to use Bluetooth, but after looking into how to do that, we rapidly decided that was an overly complicated option that wasn't worth the effort, so we switched to utilizing web-sockets instead. Later on this was replaced with Azure WebPubSub allowing further simplification.

setConnectionFailSafeBehavior(ConnectionFailSafeBehavior.LANDING) might not induce the landing behavior desired. There were problems with giving the drone permission to land in unsafe areas. To allow more stability when manoeuvring a slight pitch of 1 degree was added. This was found after extensive experimentation.

Originally we intended to only use position control to make the drone mirror the simulator by perpetually instructing it with the currently targeted position. Due to drift this was not possible and instead we rely on the drone's default stabilization as much as possible only giving instructions when it deviates sufficiently from the simulation. While using angle-control makes the drone very closely mirror the heading in the simulator, height control proved to be unreliable likely due to the

optical height sensor. We therefore give the drone a fixed up-/down-velocity when it deviates too strongly from the target height.

As mentioned in the Android App-section the target SDK is 30. This makes the project unsuitable for Google Play.

## 7.5 Notes on expansion

The documentation for the mobile SDK can be found under:

<https://developer.dji.com/api-reference/android-api/Components/FlightController/DJIFlightController.html>.

A list of compatible drones is available on

<https://developer.dji.com/products/#!/mobile>.

The message receiving code uses parsing to identify the type of message and therefore the intended receiver.

```
1 reference
private void handleConnection()
{
    var url = pubSubClient.GetClientAccessUri();
    using (var client = new WebSocketClient(url))
    {
        client.IsReconnectionEnabled = true;..
        client.ReconnectTimeout = System.TimeSpan.FromSeconds(5);

        client.MessageReceived.Subscribe(msg =>
        {
            if (msg.Text.Contains("batteryChange"))
            {
                try
                {
                    messages.Enqueue(JsonUtility.FromJson<Message>(msg.Text));
                } catch (Exception e)
                {
                    Debug.Log(e.Message);
                }
            }
        });
    }
}

Frohbö, K.C. (Kevin, Student B-TCS) *
override fun run() {

    val service: WebPubSubServiceClient = WebPubSubServiceClientBuilder()
        .connectionString("Endpoint=https://dronesim.webpubsub.azure.com;AccessKey=6z6QR0wNT")
        .hub("pubsub")
        .buildClient()

    val token: WebPubSubClientAccessToken =
        service.getClientAccessToken(GetClientAccessTokenOptions())

    while (true) {

        val websocketClient: WebSocketClient = object : WebSocketClient(URI(token.getUrl())) {
            override fun onMessage(message: String?) {
                if (message!!.contains("isPhoneControlled")) {
                    try {
                        val controls = Gson().fromJson(message, Controls::class.java)
                        controlsQueue.addFirst(controls)
                    } catch (e: Exception) {
                        // Do nothing
                    }
                }
            }
        }
    }
}
```

After making changes to Control or Message classes it must therefore be verified that these filters would still function. Likewise it must be noted that the Control class exists in simulator and phone. Changing it on one end therefore requires the same changes on the other end. The same applies for the Message class.



## 8 Tests

### 8.1 Unit Tests

We initially intended to utilize unit tests, particularly for the message handler and the communication protocol. However, as our project developed, we found that the overwhelming number of the methods in those classes were of type void, which were not viable to test utilizing unit tests. However, we did utilize a python mocker which we used to test these two aspects of the system.

### 8.2 Python Mocking

As Unit tests were not very suitable with our approach, and to ensure we had tested the software developed independently on the desktop and android sides, we created a very simple python mocker to test the handling of received drone data. First, we collected some real drone data with the android app and saved it as a data dump text file. Then we created a very simple python script that would read the data dump, connect to the Unity websocket and send the data at different sample rates. Thanks to this simple mocker, we could ensure that the Unity digital twin's behavior was correct without the use of the real drone. Besides using the real data dump, we also sent single messages that would result in a single simple movement (i.e. vertical up, vertical down, rotate 90 degrees), to test the DroneControl and DroneMover. Essentially, using this mocker "replaces" unit testing of the Unity component of the project.

### 8.3 System Tests

To perform these tests the simulator and App are set up as described in the README files.

1. Test: Turn the drone off using the button at the underside of the drone body. Turn the drone back on.  
Expected: The simulator should reset after 5s of the drone being disconnected. Once the drone is back online the simulator should pick it up again.  
Results:  
16/4/24 Success
2. Test: Try to move the drone using the arrow keys.  
Expected: Neither the simulated nor real drone should react.  
Results:  
16/4/24 Success
3. Test: Press SPACE in the simulator.  
Expected: The real world drone and the model should rise to 1.2m.  
Results:  
16/4/24 The real drone struggles to get to stability in a closed room and stabilizes around a value that is too low. Since this results from error in the height sensor (light sensitivity) there is no clear way to improve on this.
4. Test: Make the drone lift-off. Press SPACE again.  
Expected: The real and modeled drone should vertically descend until landed.  
Results:  
16/4/24 Success
5. Test: Attempt to move the simulated drone during takeoff using the arrow keys.  
Expected: Neither drone should react.  
Results:  
16/4/24 Success
6. Test: Attempt to move the simulated drone during landing using the arrow keys.  
Expected: Neither drone should react.

Results:

16/4/24 Success

7. Test: Make the drone lift-off. Use the sideways arrow-keys to make the drone spin.  
Expected: Real and simulated drone should rotate about their own axis in either direction. The displayed heading should change correspondingly.

Results:

16/4/24 Success, with the caveat of the real drone drifting during manoeuvring.

8. Test: Make the drone lift-off. Use the up-down arrow-keys to make the drone move vertically.  
Expected: Real and simulated drone should move vertically in either direction. The displayed height should change correspondingly.

Results:

16/4/24 Success, with the caveat of the real drone drifting during manoeuvring.

9. Test: Press "r".  
Expected: The lift-off button on the phone should be enabled.

Results:

16/4/24 Success

10. Test: Press "r". On the phone press "Liftoff", after liftoff press "Land".  
Expected: The lift-off button should be enabled. The drone should lift-off when "Liftoff" is pressed, the button text should change to "Land". The 2 seekbars should be enabled. The drone should land with the text changing to "Liftoff" when pressing "Land". This should disable the 2 seekbars.

Results:

15/4/24 Clicking "land" immediately after "liftoff" results in the button changing but no landing being initiated. Since pressing "Liftoff" and then "Land" again lands the drone and since this is a very unlikely thing to occur we avoided a solution attempt as this would add undue amounts of complexity.

11. Test: Press "r". Liftoff via the phone. Use the different settings of each seekbar.  
Expected: The "Yaw"-bar should rotate the drone left or right at fixed speeds. The "Vertical"-bar should move the velocity up-and down at fixed speed.

Results:

16/4/24 Success, with the caveat of the real drone drifting during manoeuvring.

12. Test: Press "r", then "Liftoff". Press "r" again. Control the drone via the simulator. Press "r" again. Move the drone using the phone.  
Expected: The drone should be able to switch control from phone to simulator seamlessly.

Results:

16/4/24 Success.

13. Test: Let the simulator be in control. Press "r" during liftoff.  
Expected: Pressing "r" does nothing.

Results:

16/4/24 Success.

14. Test: Let the simulator be in control. Lift-off. Press "r" during landing.  
Expected: Pressing "r" does nothing.

Results:

16/4/24 Success.

15. Test: Let the phone be in control. Press "r" during liftoff.

Expected: The handover of control proceeds. The simulator drone will be flying at the height the virtual drone was at when handover took place. So the real-drone might fly to default height (1.2m) and then rapidly descend to the handover height.

Results:

- i. 19.4 Success.

16. Test: Let the phone be in control. Lift-off. Press "r" during landing.

Expected: The handover of control proceeds. It is possible the simulator switches to FLYING-state. This would cause an immediate takeoff when landing has completed.

Results:

- i. 19.4 The virtual drone hovers in the air while its real counterpart stays on the ground. Landing it makes the virtual drone be in sync again. This issue arises since the `isFlying` property of `DroneController.kt` becomes true as soon as a `liftoff` or `land` command was sent. Unlike `FlightController.state.isFlying` this property is not tied to whether the drone is currently flying. It merely reflect whether most recently a landing or takeoff was ordered.

17. Test: Make the drone lift-off, disconnect the controller, reconnect the controller. Test this for drone- and simulator-control.

Expected: The simulator should reset to default after 5 seconds, once the drone is synced with the app again this should cause an immediate landing. If it was possible to reconnect quicker than that operations could resume normally.

Results:

16/4/24 The app crashed this was fixed by moving the if-statement in the inner loop of `run()` of `DroneController` above the `"Thread.sleep()"`-call.

16/4/24 After reconnection the droneController can no longer be identified. This relates to how the code from <https://github.com/godfreynolan/DJITutorialsKotlin/blob/main/1-Registration> sets up communication of controller and phone. Since it would be very difficult to fix this and it is unlikely someone would disconnect the controller during normal operation no solution for this was developed.

18. Test: Make the drone lift-off place a wrinkled blanket under the drone and press land.

Expected: The drone should always land when requested to. This is to verify that landing-confirmations which the drone requests for unsafe landing areas are actually being given.

Results:

- i. Using the DJI FLYApp which checks landing confirmation it was not possible to trigger a warning. Therefore the test failed to carry out successfully.

19. Test: Have the phone in charge. Liftoff. Disconnect the phone from the internet. Reconnect after 5s.

Expected: The phone and simulator should reset after 5s forcing a landing. Once the network is back on a reconnect should follow.

Results:

18/4/24 Success. Reconnect takes about 20s

Note: Important is that any network reset results in the phone and simulator returning to defaults. This therefore also covers simulator control.

20. Test: Have the phone in charge. Liftoff. Disconnect the PC from the internet. Reconnect after 5s.

Expected: The phone and simulator should reset after 5s forcing a landing. Once the network is back on a reconnect should follow.

Results:

18/4/24 Success. Testing this without liftoff we found Resynchronizing only happens after over 1 minute. Setting `client.ReconnectTimeout` in `handleConnection()` in `Dronecontrol.cs` to 5s did not alleviate this.

Note: Important is that any network reset results in the phone and simulator returning to defaults. This therefore also covers simulator control.

## 8.4 Usability Tests

Utilizing a 3rd Person, includes installing the application on phone, setup the drone, and performing a task.

A person external to the project with a Computer Science background is used to test the design and documentation. The following tasks are performed:

- Install Unity hub and Android Studio as instructed and import the projects.
- Setup the drone and controller, upload an instance of the app to a phone and open the simulator scene.
- Test lifting off and landing the drone via a phone. Test lifting off and landing via the simulator.
- Once it is clear how to liftoff and land safely, test steering the drone using the simulator and phone (switch control in the air).
- The DJI mobile sdk is capable of querying velocity data. Describe where in the code you would make changes to get and send this data to the simulator.

The test was carried out on 18/4/24 with another TCS student, Aren Merzoian, who consented to have his name listed in this report. After the test it was noted that the installation tutorial structure needed change to: Setup WebPubSubService, Setup App, Setup Unity, Connect Controller. Registering for the API key could be elaborated on specifically finding the developer center. It must be clarified where the battery holder is and which button turns the drone on. It should be clarified that connecting the controller requires permission. It must be clear what to expect when the drone is connected. It should be clear which axis is vertical and what yaw is (using images). The usage tutorial steps (initial liftoff and landing) should be denoted as such. An issue noted was that the App requests several unnecessary permissions such as being able to make phone calls. Further more subheadings could be used for navigation. It was noted that sometimes the app needs to close and open again to be able to grant the permission.

## 9 Evaluation

### 9.1 Risks

1. Loss of contact with the drone: Should the App or controller fail the drone will hover in place. The drone will still drift and could collide with something. As soon as the propellers make contact with an object they shut down and the drone crashes.

Mitigation: As a safety measure we strongly recommend installing and familiarizing oneself with the DJI Fly app. Even with the drone already in flight one can select "Go fly" to take control of the it and initiate a safe landing. Further the controller and drone should be charged at least to 50% when performing any flying and the network connection should be reliable. The App automatically lands the drone after 5s of no contact with the simulator.

2. Unsafe landing areas: The drone uses a sensor to determine whether the landing zone is clear. By default it is configured to request permission. In the .init() method of "DroneController.kt" this functionality was turned off using:

```
flightAssistant.setLandingProtectionEnabled(false, {})
```

This was done to ensure that in case of an emergency a quick landing could be enforced. Should the drone be moved to an unsafe landing area such as over water it would still attempt landing.

Mitigation: The digital twin application is intended for use in the IoT-Cyberlab (this is what was modelled) where such issues are unlikely to occur.

### 9.2 Ethics

Given the foundational nature of the project there are not many ethical issues that can be identified already. Since the environment of the digital twin is static, it does not properly model people in that environment. As a result, it is entirely possible to fly the twin, and hence the real drone, directly into a person, injuring them. As a result, some caution should be exercised when utilising the drone, and in general, it should not be utilized in its current state outside the lab, except under special conditions. The drone currently uses no encryption or authentication. If an application was developed from this such issues would need to be addressed in the interest of protecting privacy. Likewise the drone camera could raise privacy concerns if it was to be used in spin-off applications.

## 9.3 Future Development Paths

### 9.3.1 Possible improvements

We are overall quite satisfied with our progress during this module, however, there are a few ways that we envision our project could be further improved upon. A large issue remaining is drift during position changes. The drone is well-able to maintain its position while hovering as soon as the automatic-hover is turned off however even if no change in height or yaw is intended the drone will start drifting. Better algorithms for motion control (for instance using PID-control with feedback from the accelerometer) could alleviate this issue. A related problem is that no movement in the x,y-plane was implemented in the simulation. This is because there is no sensor for those values inside a room (no GPS). Modelling an outside area (which was originally planned) could help with this.

### 9.3.2 Possible spin-off projects

There are a number of paths that we could take to further develop the project in different directions, to create more specialized twins for the sake of future research.

The first of these ideas is to attempt to allow multiple digital twins, possibly of different types of machines, to interact with one another through the medium of the Internet of Things. One theorized concept was the idea of having the rover twin move around, and communicate its position with the drone twin, and have the drone navigate to and land on the rover.

Another idea would be to aid in the detection of objects using the camera function on the drone, perhaps extending to the point of being able to generate an environment for the twin utilising photos taken. This could perhaps be done at a basic level by attempting to generate a version of the lab utilizing a large number of photos, perhaps in the form of point-clouds, to allow the drone to detect when there's something or someone in the lab that isn't there normally, and then generating such a thing within the environment.

Particularly the mobile app and the drone-simulator-communication could be used to develop a line-tracking application. The camera might also be used to identify markers for positioning inside a room.

Another (smaller) note is that during flight the drone is unstable when actively controlled. The fact that by slightly adjusting the pitch we significantly stabilize it indicates that it should be possible to perform manoeuvres with much less drift given the right motion algorithms.

## 10 Conclusion

Overall, we as a team are very satisfied with the product we have made, and the things we have learned throughout this process. There were a number of problems that were discovered throughout the process that we think are worth considering for the future. As an example, we thoroughly underestimated the amount of time it would take to develop the twin, thereby proving the value of this project in the long term. This resulted in our true timeline being very different from our initial, predicted timeline. However, through solid teamwork, and high levels of communication, we still managed to keep to a very well structured timeline, and reached our goals at a pace that was satisfactory to us.

The goal of this project was to create a prototype digital twin of a drone, which could be utilised in the future by other research groups to create more complex twins, with the goal of advancing further research. In this regard, we believe we have succeeded, as we have developed a solid foundation upon which new twins can be built with minimal difficulty. We have been informed that our project is to be used in future by masters students for their projects in developing digital twins.

This project served to teach us working with unity and drones alongside further helping us learn how to work on a large scale project over a long period of time. It helped us understand how to better predict work requirements, and timescales, and how important communication is to a team and a viable product.