Intelligent Systems for Next Generation Drone Functionality

Thomas CARSTENS

In this thesis, I examine how smart systems benefit UAVs, and the tasks in which they operate. The problem I address is to understand the tradeoffs of certain smart systems over others in assisting UAV functionality.

The first approach explores the potential of distributed systems to assist task development upon UAVs. We document the design of a development and demonstration testbed conform to existing research. We describe a procedural task-based architecture to complement an existing swarm stack. We demonstrate that the runtime environment is capable of coordinating multiple robots. A custom high level interface wraps the testbed for more complex tasks, and this is demonstrated in a multi-drone choreography. We investigate a Mixed Reality Interface for the Testbed, as well as methods of drone Piloting using a Computer Vision algorithm. The utility of the framework is demonstrated with two different tasks: quadrotor piloting using computer vision and collision-free flight of multiple UAVs. Building on existing frameworks like MediaPipe Hands, and Unity3D, we create perception pipelines for semi-autonomous flight, and we proceed to evaluate the response latency of these pipelines.

The second approach explores the potential of onboard systems to assist the real-world deployment of UAVs. Applications are explored for UAVs as Mobile Sensing Platforms, with high-sampling and high-precision equipment. We design a carrier drone with Onboard Data Acquisition and we put it to practice along standards defined by industrial practitioners. Two payloads are tested in outdoor flight, for atmospheric data and vibration data.We characterise the sensors used on these payloads. A vibration probe is designed and our tests demonstrate its relevance in the field of mobile sensing.

Contents

Introduction 1

Drones, or unmanned aerial vehicles (UAVs), have been around since the early 1900s. Drones can range from the size of airplanes to the size of bumblebees (Figure 1.2). Originally used for military operations, they became more widely used after about 2010 when electronic technology got smaller, cheaper and more efficient, prices on cameras and sensors dropped, and battery power improved [339][340]. Where once scientists could only observe earth from above by using manned aircraft or satellites, today they are expanding, developing and refining their research thanks to drones [341] [342][339].

Depending on their mission, drones are equipped with different payloads or equipment (Figure 1.2). Digital cameras can identify plants and animals, and help create 3-D maps. Thermal cameras detect heat from living creatures like animals or stressed plants, as well as from water [342]. Hyperspectral imaging identifies features of plants and water through measuring reflected light and can interpret a wider range of wavelengths than the human eye can see. LiDAR, which measures how long it takes for an emitted pulse of light to reach a target and return to the sensor, can be used to calculate the distance to an object and its height, which is used for 3-D maps [342]. With this range of sensors, scientists and practitioners can choose from a range of options to expand their research.

Figure 1.1: Examples of mini-UAVs used for remote sensing, from [339].

We see great growth in mobile mapping research, that is "the acquisition of spatiotemporal phenomena by using a mobile multi-sensor platform" [343]. This field of research includes remote sensing [344] the acquisition of information about an object or phenomenon without making physical contact with the object, as well as various contact-based techniques [345] for the acquisition of information in direct contact with the object.

Figure 1.2: DJI Matrice 200 with onboard gas detector (from DJI's website, 2021)

Inspection and Data Gathering is an area of scientific study that is gradually adopting drones. For instance, the Sea Level and Coastal Changes group at MARUM (University of Bremen) [346] studies coastal erosion, mangrove communities as well as the distribution of corals and the death of shallow corals. Prior to drones, the typical thing to do is to put a GPS on your backpack and walk along the beach to actually measure points on the beach. Coastal areas change rapidly for example, before and after a storm (Figure 1.3). Instead, The drones take many pictures at short intervals, and repeat flights at short intervals can show differences in conditions. A drone can cover the same area in less time and get much higher resolution pictures. This demonstrates that these tools are an ideal integration in a data gathering toolkits of researchers and practitioners. As a result, the industrial usage of drones is growing steadily.

Figure 1.3: Drone picture of coastline after a storm.

All in all, applications for drones are crossing boundaries of science and industry, with everything from aerial photography to package delivery to disaster management benefiting from the technology. But before they become commonplace, there are challenges to be solved to make them reliable and safe.

Drones are usually flown with a controller on the ground, and some form of wireless communication (usually radio signals) between the operator and the drone [339]. Since the remote pilot focuses on navigational aspects, it restricts the range of flights of a single operator. A challenge to drone flight is the amount of human involvement. [347] looks at the capacity of the research team collecting data. Five provided information regarding the human involvement in their experiments; four of which are damage assessment. Pratt et al. [348] (2008) included four members in their UAS operation crew: a pilot, a safety manager, a mission specialist and a tether manager. Kruijff et al. (2012) [349] applied the same team structure as Pratt et al. (2008), less the tether manager. Murphy et al. (2008) [350] discussed extensively the responsibility of each team member in a UAS flight and recommended a crew of three: a pilot, a mission specialist and a flight director.

In developing complex functionalities, the operator is focused on remote piloting, or on manning the ground station. In so doing, the operation requires further automation in order to give more flexibility to the operator, and in their amount of involvement.

A window of opportunity for the quality and application possibilities of industrial drones is the built-in sensor technology [346]. Sometimes referred to as smart drones, additional monitoring systems can be installed between a drone's flight control and smart sensors [351]. Ideally, this allows for more efficient motors, better on board processors and software, more accurate sensors, as seen in Figure 1.4, built-in safeguards, networked together to enable coordination, collaboration and real time data delivery, etc.

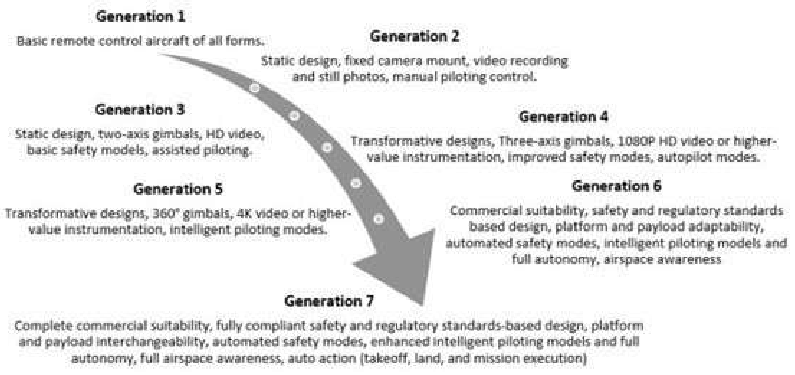

Generation 1
Basic remote control aircraft of all forms.
Generation 2
Static design, fixed camera mount, video recording and still photos, manual piloting control.
Generation 3
Static design, two-axis gimbals, HD video, basic safety models, assisted piloting.
Generation 4
Transformative designs, Three-axis gimbals, 1080P HD video or higher-value instrumentation, improved safety modes, autopilot modes.
Generation 5
Transformative designs, 360° gimbals, 4K video or higher-value instrumentation, intelligent piloting modes.
Generation 6
Commercial suitability, safety and regulatory standards based design, platform and payload adaptability, automated safety modes, intelligent piloting models and full autonomy, airspace awareness
Generation 7
Complete commercial suitability, fully compliant safety and regulatory standards-based design, platform and payload interchangeability, automated safety modes, enhanced intelligent piloting models and full autonomy, full airspace awareness, auto action (takeoff, land, and mission execution)

Figure 1.4: Generations of drone technology.

There are challenges to implementing smart sensors, which is the length and difficulty of the development process. For instance, collision avoidance is generally recognised as a complex process [352][353]. This complexity can be attributed to the many different variables that factor in based on the applications in which this technology is used. Collision detection requires regular input of altitude, which is usually transmitted by an onboard range sensor. Additionally, with a view of obstacle, the drone can estimate. This is delegated to a camera or radio frequency sensor. Furthermore, algorithms would be required to monitor a obstacles relative to the moving drone, or detect if noise is an obstacle, which is solved with finer state estimation techniques [354].

1.1 Problem Statement

In this thesis, I examine how smart systems benefit UAVs, and the tasks in which they operate. The problem I address is to understand the tradeoffs of certain smart systems over others in assisting UAV functionality.

What environments, and tools, can we put at disposition to accelerate the development of specialised functionalities using drone equipment? What functionalities are ideally developed in a research setting, as opposed to an industrial one? Are there any functionalities which require specific technological integrations? Should these functionalities be developed before flight, or during flight? This last distinction is the basis for two separate approaches: a dedicated test-and-demonstrate environment as opposed to a ground station.

The first approach explores the potential of distributed systems to assist task development upon UAVs.

In our first approach, we examine distributed systems, whereas multiple systems coordinate to assist, and simplify tasks during development and demonstration. This is a systemic approach that aims to interconnect the available systems. For instance, smart sensors can control a drone's navigation path and monitor its flight. The operator's interactions can be guided by computer vision systems along with object detection and collision avoidance programs. New forms of artificial intelligence or algorithms can make them even more adaptable. We explore the ways in which distributed systems can spare the practitioner from repetitive and time-consuming tasks.

Figure 1.5: A dedicated test-and-demonstrate environment.

The second approach explores the potential of onboard systems to assist the real-world deployment of UAVs.

In a second approach, we look more closely at the interplay between smart systems, and the real-world deployment of a UAV. We investigate how a drone's design aids to gather multi-faceted data. Data acquisition systems offer an opportunity to manage the data collection process. Specialised navigation tasks offer other means of automating the data collection process. The drone operator can examine the data gathered in real time, and becomes involved as a data interpreter. We explore the ways in which onboard systems can aid with the practitioner's task.

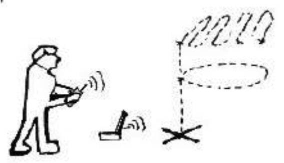


Figure 1.6: An environment for outdoor deployments.

1.2 Research Domains

This thesis is associated with three major fields of work: swarm engineering, human drone interfaces and mobile sensing. These fields take inspiration from a variety of other fields. Figure 1.7 shows several domains that are explored in this thesis.

According to *An Introduction to Swarm Robotics* [355], swarm robotics is an approach to collective robotics that takes inspiration from the self-organized behaviors of social animals. Through simple rules and local interactions, swarm robotics aims for robust, scalable and flexible collective behaviors for the coordination of large numbers of robots. In contrast, the term swarm engineering [356] describes the design of predictable, controllable robot swarms with well-defined goals and the ability to function under certain conditions. Swarm engineering focuses

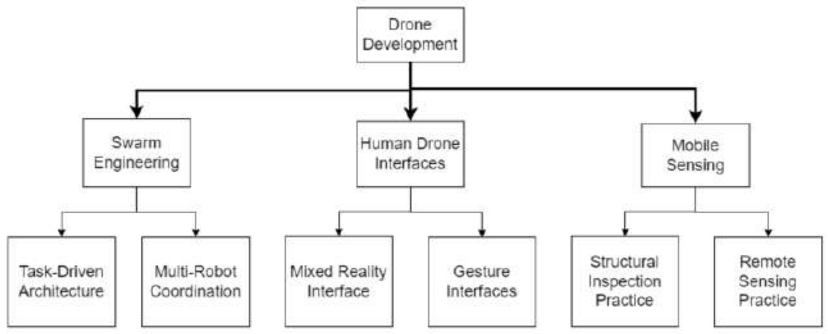


Figure 1.7: Research domains explored in this thesis.

mainly on concepts that could be relevant for real-world applications, therefore shifting swarm robotics to engineering applications.

In *The state-of-the-art of Human–drone interaction: A survey* (2019), Dante Tezza and Marvin Andujar define Human-Drone Interaction (HDI) as a field of research that consists of understanding, designing and evaluating drone systems for use by humans, and in contact with humans. This field is similar to human-robot interaction (HRI), however, a drone's unique characteristic to freely fly in a 3D space, and unprecedented shape makes human-drone interaction a research topic of its own. Researchers develop control modalities and better understand means of communicating with a drone.

We see great growth in mobile mapping research, that is "the acquisition of spatiotemporal phenomena by using a mobile multi-sensor platform" [343]. The UAV is a platform that greatly simplifies research. One such field remote sensing, the acquisition of information about an object or phenomenon without making physical contact with the object [344]. Recently, UAVs have enabled research towards contact sensing, for the acquisition of information in direct contact with the object [345], on a platform that serves for optimal sensor placement [345] [358].

1.3 Contributions

This list of contributions outlines each project in its respective field and an overview of the approach used to evaluate it.

A Testbed Environment for Task Development

A Multi-Robot Management Layer

We motivate a smarter ecosystem for task development upon drones by beginning with the infrastructure for new technologies and for prototyping functionalities. A centralised swarm framework serves to set up flight performance monitoring systems, a fundamental asset to the development of robots and multi-robot groups [359]. A hover stability test is a good measure of system performance since it requires quick readjustments of the drone to counter natural disturbances during hovering. In [360], Michał Waliszkiewicz et al. determine the performance of their flight controller by comparing the attitude of the drone in relation to the demanded null value of angular rotations. In this experiment, two drones are required to hover at an input setpoint with minimal error. The error over time is compared for the drones to better understand flight stability.

A High Level Interface

A high level interface is an abstraction layer for development activities. In order to simplify task development, and align with the thesis goals, we develop a framework for high level interaction between the operator and the functionalities of the testbed. A drone choreography is designed as a live demonstration of the Testbed's functionality. The experiment data is accessible publicly [361].

Experimentations for Human-Drone Interfaces

A Gesture Controller for UAV Piloting

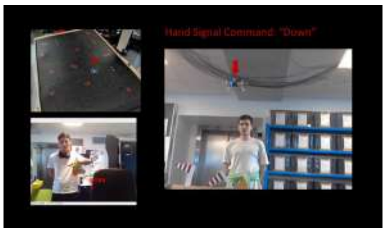


Figure 1.8: Presentation video [362] of the Gesture Recognition Pipeline

As of Nov. 2019, multiple gesture interfaces have been developed for UAVs [363] [364], but are lacking in drone piloting. Realtime interfaces for drone piloting are discouraged [357] due to high latency and low control precision compared to other drone control modalities. As of Sept. 2021, the literature utilizing the Crazyflie nanodrone does not include realtime streaming commands [365].

We put in place a demonstration for flight piloting in real-time using the developed gesture interface. We present the workflow of real-time gesture piloting pipeline and we evaluate it in terms of:

- System response time
- Accuracy of gesture recognition

Effective Gesture Recognition In order to evaluate gesture recognition performance, [366] evaluates the false positive and negative rates of the pose detection by manually identifying both the incorrectly recognised gestures, and the unrecognised gestures. Similarly, we identify the false positive and negative rates of the pose detection.

A Pipeline with minimal Response Time The system response time was verified by applying a series of rapid maneuvers to register any significant delays between the pilot's commands and their execution by the flight control system. Michał Waliszkiewicz et al. [360] choose to modify the drone's angle in a specified direction. This choice is arbitrary and the changes in velocity are used in this case.The input was a demanded velocity in a specified direction. The input was changed randomly by the operator with hand movements. The output was a delay of the velocity change in the drone. Finally, a system response time is determined by averaging the response delays over the experiment.

A Mixed Reality Setup for Drone Development

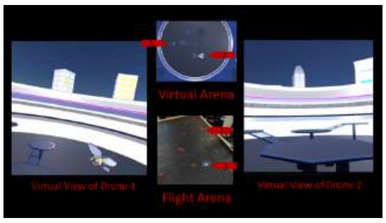


Figure 1.9: Video feeds of Mixed Reality Setup. The setup is maintained as a Github repository [367] and a presentation video is available [368]

The first objective of the simulated environment is to serve as a graphical interface in order to develop tasks otherwise too difficult to deploy. The priority of the virtual reality is therefore set on rendering capabilities, and the ability to obtain camera streams from this environment. We set up a virtual interface between real and virtual objects in real time. This MR simulation consists of a network interface between a robotics backend (ROS) and virtual environments (Unity3D). Similarly to [369], the pipeline is then evaluated in terms of communication latency for two separate scenarios.

- when transmitting parameters into the simulated environment

- when transmitting parameters to the robotics backend.

The latency of data transmission into the virtual setup

This latency is evaluated by determining the time-delay between the detection of drone poses from the robotics backend and the time they are received in the virtual setup. This approach is also taken in [369], who observe that on average a 400 ms time delay occurred in their MR simulation.

The latency of data transmission out of the virtual setup

This latency is evaluated by determining the time-delay between the detection of an event from the virtual setup, and the time that they cause a state change in the drone's task manager. This approach deviates slightly from [369], who measure the moment the information is displayed on their graphical interface. Both approaches measure the response time before the event-data has its intended effect.

In Vivo Deployment for Industrial Environments

Optimal Payload Shock Absorption for Optimised Payload Transport

We test the vibration sensitivity of the payload to remove any parasitic vibrations. We compare the shock absorption of two payloads subject to different amounts of damping material. This is done by developing a vibration profile as direct input from two accelerometers during drone flight and increasing the volume of damping material.

.

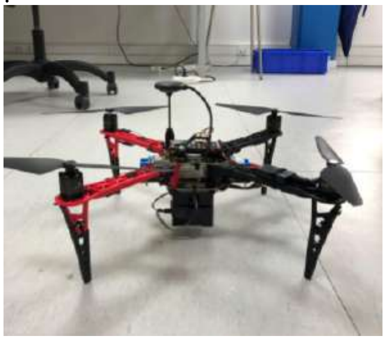


Figure 1.10: Data Acquisition System setup.

Monitoring Environment Conditions with a Drone Fieldscan Solution

We develop a drone solution for sampling environment conditions during a drone flight (Figure 1.11). Sunlit and shaded regions of an open field were scanned for relative humidity, luminosity and ambient temperature. The accuracy of the data setup was verified by examining the contrasts in atmospheric conditions between sunlit and shaded regions of an open field. The dataset [370] and presentation video [371] are available.

Figure 1.11: Field Scans

Detection of Footsteps with a Drone Vibration Solution

Structural inspections include seismic equipment upon UAVs [358], yet there lacks any mention of vibration probes on UAVs. We develop a drone prototype for acquiring vibration data after flying, landing, and recording under various scenarios (Figure 1.12).

We test the validity of a drone vibration solution by using real-world human walking experiments on a concrete floor structure. This approach is also taken in Jonathon Fagert et al. [372] (2021), who test an onboard UAV sensing module consisting of a series of geophones.

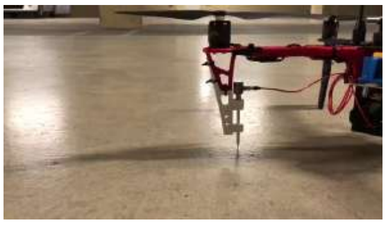


Figure 1.12: Footstep Detection with an Onboard Accelerometer

Determining the Sensitivity of the Drone Vibration Solution

We characterise the sensitivity of the drone probe solution with a sinusoidal sweep. A vibration shaker performs a sinusoidal sweep in a controlled setting.

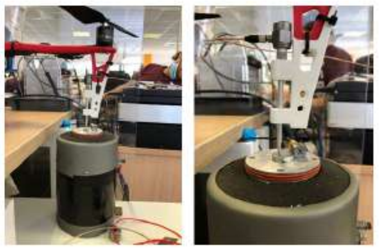


Figure 1.13: Sensitivity Test: Experimental Setup

1.4 Overview of the Thesis

Chapter 2: A Testbed Environment for Task Development

In line with the thesis goal, we seek to better understand how distributed systems can offer smart assistance to multi-robot development. We explore research and techniques designed to coordinate multiple robots. We document the design of a development and demonstration testbed conform to existing research. We describe a procedural task-based architecture to complement an existing swarm stack. We demonstrate that the runtime environment is capable of coordinating multiple robots. A custom high level interface wraps the testbed towards more complex tasks, and it is demonstrated in a multi-drone choreography.

Chapter 3: Experimentations for Human-Drone Interfaces

In line with the thesis goal, we look at two types of smart systems that enhance the interactions between humans and drones. The first allows the human to pilot a drone through a gesture interface. The second looks to a virtual interface as a training ground for a real drone to avoid a virtual object. A distributed system is used to to communicate and coordinate these pipelines by passing messages to one another from any system. To better understand their tradeoffs, the distributed systems are explored and evaluated in this chapter. We investigate a Mixed Reality Interface for the Testbed, as well as methods of drone Piloting using a Computer Vision algorithm. The utility of the framework is demonstrated by using it for two different tasks: quadrotor piloting using computer vision and collision-free flight of multiple UAVs. Building on existing frameworks like MediaPipe Hands, and Unity3D, we create perception pipelines for semi-autonomous flight, and we proceed to evaluate the response latency of these pipelines.

Chapter 4: In Vivo Deployment for Industrial Environments

In line with the thesis goal, we pay attention to the interplay between smart systems, and the real-world deployment of a UAV. The carrier drone aids in streamlining the data collection process, by automating different fly-by procedures, safeguards, and scheduling the data collection. Two payloads are tested in outdoor flight, for atmospheric data and vibration data, and the sensors used in these tests are evaluated. In this way, onboard systems can aid with the practitioner's task.

Applications are explored for UAVs as Mobile Sensing Platforms, with high-sampling and high-precision equipment. We design a carrier drone and Onboard Data Acquisition systems and we put them to practice along standards defined by industrial practitioners. Two payloads are tested in outdoor flight, for atmospheric data and vibration data, and we characterise the sensors used for these tests. A vibration probe is designed and our tests demonstrate its relevance in the field of mobile sensing.

A Testbed Environment for Task Development 2

2.1 Introduction

According to *An Introduction to Swarm Robotics* [355], swarm robotics is an approach to collective robotics that takes inspiration from the self-organized behaviors of social animals. Through simple rules and local interactions, swarm robotics aims for robust, scalable and flexible collective behaviors for the coordination of large numbers of robots. In contrast, the term swarm engineering [356] describes the design of predictable, controllable robot swarms with well-defined goals and the ability to function under certain conditions. Swarm engineering focuses mainly on concepts that could be relevant for real-world applications, therefore shifting swarm robotics to engineering applications. We motivate a smarter ecosystem for task development upon drones by beginning with the infrastructure for new technologies and for prototyping functionalities. A centralised swarm framework serves to set up flight performance monitoring systems, a fundamental asset to the development of robots and multi-robot groups [359].

Multi-robot systems and swarms of unmanned aerial vehicles (UAVs) in particular have many practical applications. *Sensors and measurements for Unmanned Systems: An overview* [340] from March 2021 shows applications as diverse as surveillance and monitoring, inventory management, search and rescue, or in the entertainment industry. Swarm intelligence has, by definition, a distributed nature. Yet performing experiments in truly distributed systems is not always possible, as much of the underlying ecosystem employed requires some sort of central control. Indeed, in experimental proofs of concept, most research relies on more traditional connectivity solutions and centralized approaches [359][373].

In recent years there have been significant advancements in this research field. However, very rarely do UAV swarms leave the controlled and safe environment of laboratories, and when they do it is for short-duration experiments. The current use of swarms is generally based on custom, centralized solutions, in environments with reliable communication [373]. There remains large challenges to reliable flight of UAVs: the reliability of software, the limits on the hardware, test and validation of new elements on pre-existing systems [359]. A further challenge concerns multi-robot functionality. Current UAVs have very limited functionality for multi-robot coordination. Market drones are too limited for swarm research, as they only supports a single point-to-point link between a program and the drone, thus, a program can only communicate with a single drone.

In line with the thesis goal, we seek to better understand how distributed systems can offer smart assistance to multi-robot development. We explore research and techniques designed to coordinate multiple robots.

2.2 Related Work

In this section, we examine how researchers tackle the challenge of swarm engineering in the past. Prior work exists in a variety of drone laboratories [374] [375]. Drones Spatial Localization, UAV Architectures and UAV Swarm Frameworks require tradeoffs.

2.2.1 UAV Spatial Localization

The Flight of UAVs, as with any robotic system, requires accurate positioning. However, a drone suffers from cumulative drift in position data [351]. In order to achieve autonomous flight, a drone will need to know if it is following a trajectory correctly. A localization technology allows for this centimeter level accuracy.

Figure 2.1: An example of drones tracked by two motion capture cameras.

It is by using highly precise equipment, that UAVs can have highly precise state estimation. This setup includes the selection of onboard positioning systems as well as external positioning solutions. We focus on optical motion capture coupled with algorithms for state estimation primarily a set of technologies commonly used by UAV laboratories[376] [377] [375]:.

The high level of precision from a motion capture system allows us to synchronously hover multiple UAVs. The tracker precision can reach sub-millimeter accuracy, and drones are hovered at a precision of a few centimeters.

2.2.2 UAV Architectures

UAVs such as AscTec Pelican, Parrot AR.Drone, and Erle-Copter are other examples of UAVs commonly used in the literature. These MAVs have Software Development Kits (SDK) that enable applications from third-party developers to communicate with the drones. However, both SDKs are too limited for swarm research, as they only supports a single point-to-point link between a program and the drone, thus, a program can only communicate with a single drone.

In comparison, the Crazyflie packages the full robotic stack. This robotic stack includes its own state estimator, control architecture and trajectory follower, which work out of the box. FreeRTOS handles the scheduling of processes and control the flight calculations. The Crazyflie contains a 32-bit, 168MHz ARM microcontroller with floating-point unit that is capable of significant onboard computation. The FreeRTOS firmware is opensource and modifiable.

Figure 2.2: The Crazyflie 2.1 miniature quadcopter.

The Crazyflie's small size makes it suitable for indoor flight in dense formations. As a result, it has been used widely in research. As of 2021, this drone is used to validate research: from new algorithms for agile flight [377] to drone swarm research [376].

2.2.3 UAV Swarm Frameworks

Unmanned Aerial Vehicle (UAV) swarms have been used indoors for formation flight and collaborative behaviors, outdoors to demonstrate swarming algorithms, and in the media for artistic shows. According to *Inaki Navarro and Fernando Matia* [355], The group of robots has some special characteristics, which are found in swarms of insects, that is, decentralised control, lack of synchronisation, simple and (quasi) identical members. In this section, we explore the requirements placed on the software framework for interacting with a robot swarm.

[378] *Carlo Pinciroli, Adam Lee-Brown, and Giovanni Beltrame* discuss the key requirements a successful programming language for swarm robotics must meet. According to them, the level of abstraction need be adapted to the task at hand. The complexity of concentrating on individual robots and their interactions, i.e., a bottom-up approach, increases steeply with the size of the swarm. Conversely, a purely top-down approach, i.e., focused on the behaviour of the swarm as a whole, might lack expressive power to fine-tune specific robot behaviors. The runtime platform of the language must ensure acceptable levels of **scalability** (for increasing swarm sizes) and **robustness** (in case of temporary communication issues).

[375] *Sergei Lupashin et al.* present The Flying Machine Arena. This was put in place in 2014 with the goal of becoming a "demo-and-development" arena. They include both single and multi-robot experiments. One key element of their work is that the UAV swarm can be heterogeneous. Additionally, the position controller runs offboard, that is, the UAVs all rely on a companion computer to position themselves. The additional computational power is used for a latency compensation algorithm to improve accuracy for high-speed flights. Despite this, the framework remains robust: swarms of up to 5 UAVs are flown on a regular basis.

[376] *Thai Phan, Wolfgang Honig, and Nora Ayanian* define a system architecture for a large swarm of miniature quadcopters flying in dense formation indoors. The main challenges in swarm robotics are addressed in this framework, namely by reducing communication latency to 26ms. This is done in major part via the structure of messages broadcasted to the UAV. Preiss et al. [376] use a programmable UAV with an onboard position controller, making the system more robust to communication packet drops. With this method, a swarm of 49 Crazyflies have been flown using 3 radios. As a result, the drone swarm framework allows for robotics developers to send commands to drones in a fleet. A scalable and robust run-time platform is, in this way, a key element for real-world deployment of swarm behaviors.

2.2.4 UAV Software and Middleware

UAVs have a long tradition of being controlled with the Robotic Operating System. ROS is a meta-operating system designed for the construction of distributed systems. It provides a set of extensible tools for managing distributed robotic applications. The main goals of ROS are package management, hardware abstraction, low-level device control, message exchange between processes, and implementation of several functionalities. As a result, there are many ROS packages devoted to controlling such UAVs as individuals.

However, using multiple UAVs creates entirely new challenges that such packages cannot address. These new challenges include, but are not limited to, the physical space required to operate the robots, the interference of sensors and network communication, and safety requirements. In [376], Thai Phan, Wolfgang Honig, and Nora Ayanian thus motivates the use of a hardware abstraction layer on top of the Crazyflie. This abstraction layer, in the form of a ROS layer, is only used on the PC controlling one or more Crazyflies. The ROS driver sends the data to the different quadcopters using the protocol defined in the Crazyflie firmware.

[376] *Thai Phan, Wolfgang Honig, and Nora Ayanian* demonstrates interoperability between the PC and UAV components. The *crazyflie_ros* framework helps wrap CRTP within a ROS framework, which is useful for scenarios of hovering and waypoint following from a single robot to the more complex multi-UAV case. It provides not only standard operating system services (hardware abstraction, contention management, process management), but also high-level functionalities (asynchronous and synchronous calls, centralised database, a robot configuration system, etc.). Additionally, this includes command-line tools and a GUI for mass rebooting, firmware updates, firmware version query, and battery voltage checks over the radio.

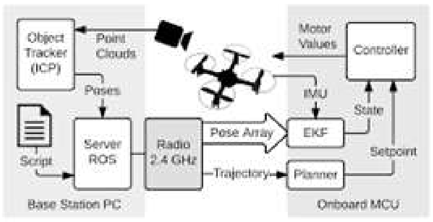


Figure 2.3: Overview of the Crazyswarm Control Loop, as per the Crazyswarm official documentation (August 2021) [379]

[380] *James A. Preiss* et al.* furthers this work by offering all the necessary components for controlling multiple drones remotely, by relating the drone flight controller of the Crazyflie to a set of controllers on the PC, but also by offering ways to send trajectories to the drones in realtime. Crazyswarm attempts to couple an external motion capture technology like Optitrack with the rest of a drone's control loop: knowing its position, the drone will be able to generate and follow a trajectory more precisely. When viewing a single body, motion capture certainly has sub-millimeter accuracy. However, as the number of drones increases, there are two limiting factors to the reliability of the control loop: the first is recognition of the drones by the optical capture system, and the second is low communication bandwidth. Multiple algorithms are therefore incorporated into this framework to mitigate the effects of these processes.

[381] *Rihab Chaari et al.* design a distributed cloud robotic architecture for computation offloading based on Kafka middleware as messaging broker. Empowering robots with cloud computing comes with a fundamental tradeoff. Offloading the execution of a computationally intensive algorithm to the cloud can reduce resource utilization, including CPU, memory, and the battery. However, this comes with a cost: communicating with cloud resources over a congested network increases latency and can lead to delay for real-time applications. *Rihab Chaari et al.* showcase that an offloading decision need not reduce the overall execution time of the application.

2.3 System Overview

2.3.1 Functionality

The testbed is designed according to four functional requirements.

1. Managing the interface with drone firmware.
2. Localizing the drones in a Flight Arena.
3. Rendering the drones in a simulated environment.
4. Managing the flow of offboard code for each drone.

These elements occur separately and simultaneously. They manage individual drones asynchronously from one another, a key element in swarm engineering. Each of these requirements is fulfilled respectively by Crazyswarm, Optitrack, Unity and the Task Manager. Each of these are explored in turn in this chaper.

2.3.2 Network Architecture

Figure 2.4 gives a brief overview of the data interfaces between the four main components of this architecture.

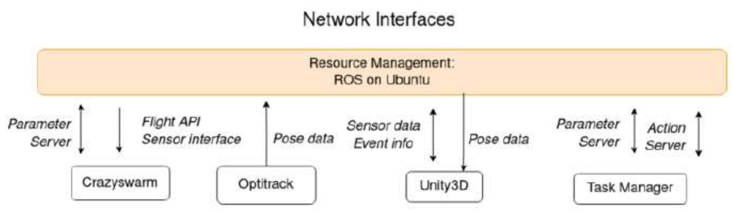


Figure 2.4: Overview of Network Interfaces.

The data flow in the Flight Arena is as follows: vehicle/object pose measurements are provided by a motion capture system to software modules running on companion computers running consumer operating systems. Within task-specific modules ("user code") and the Crazyflie communication channels, estimation and control pipelines produce vehicle motion commands. The appropriate commands are transmitted to the vehicles. Onboard the vehicles, high-frequency controllers track these commands using on-board inertial sensors in feedback. All intermodule communication is via multicast UDP and the vehicles commands are sent over a dedicated wireless channel.

2.3.3 Chapter Structure

The drone testbed is comprised of a hardware and a software environment. First, Section 2.3.2 presents the Network Interfaces. The Hardware Environment consists of the physical Flight Arena. Section 2.4 presents this Arena, the drone model and the localization system. Section 2.5 then touches on swarm management followed by task management.

2.4 Hardware Environment

2.4.1 Drone Selection Process

Several drones were compared using custom criteria for drone development. These custom criteria are based on ease of use and programmability. The Dimensions criterion aims to minimize the drone size and weight. The Reconfigurable criterion investigates the modularity of the hardware layout. The Programmable criterion looks at the available interfaces for communicating with the firmware. The Autonomous Flight criterion looks at the compatibility of state estimation and trajectory planning algorithms.

Table 2.1: Drone Selection Matrix.

Criteria	Snapdragon Flight Pro	Bitcraze Crazyflie	Tello Drone	Custom Flight Controller
Dimensions	●○○○○	●●●●●	●●●●○	●●○○○
Reconfigurable	●●●●○	●●●●○	●○○○○	●●●●○
Programmable	●●●○○	●●●●○	●●●○○	●●○○○
Autonomous Flight	●●●○○	●●●●○	●●●●○	●○○○○
Selection	✗	✓	✗	✗

The Crazyflie Drone has several advantages over other drones.

- **Autonomous Flight**. State estimation and trajectory planning are managed by the Crazyflie firmware. The operating procedure is simplified to sending setpoint commands from a remote PC. [382]
- **Programmability**. At the moment of writing, there are two APIs known to send high-level commands to the drone. [382]
- **Dimensions**. Due to our space constraints, a small, light drone is preferable. Our payload of motion-capture markers brings the Crazyflie's mass to 33 grams.
- **Reconfigurability**. The Crazyflie is easily assembled and maintainable. It is compatible with a range of sensor modules for different activities. [382]

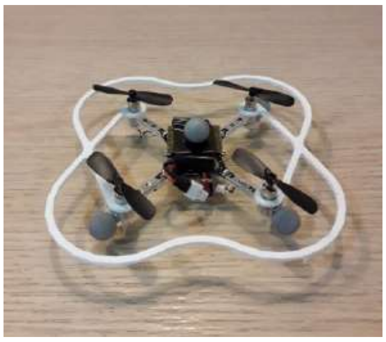


Figure 2.5: The Crazyflie 2.1 miniature quadcopter with four motion-capture markers, expansion boards (not visible), and battery.

2.4.2 Flight Arena and Spatial Localization

The Flight Area (Figure 2.6) measures 3 x 2 meters, with a table-to-ceiling distance of 1.3m. In order to dampen the impact of falling drones, the table is layered with anti-vibration cork material (Figure 2.7).

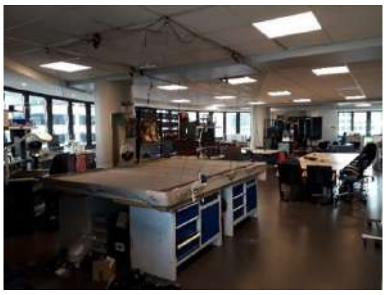


Figure 2.6: The Motion Capture Table, net and Flex 13 cameras positioned above the platform.

Optitrack [383] was adopted as the Motion Capture since the equipment was available in the laboratory. It is compatible with the swarm management solution of Section 2.5.5. Optitrack uses a Point Cloud reconstruction engine [384]. That is, it triangulates two-dimensional points from camera images into coordinates in a three-dimensional space. For this purpose, four Flex 13 cameras are set up on the Flight Arena (as seen in Figure 2.6).

The Flex 13 cameras [384] are infrared cameras, and so they must have an unobstructed view of any tracked object.

Figure 2.7: Protective cork layer.

The exact positions of the cameras give a certain coverage of the Flight Arena. The next section determines how much of the Flight Arena is localized by the cameras.

2.4.3 Lightray Coverage Study

We investigate how much of the flight arena is localized by the motion capture. The drones can only be flown in a space covered by the infrared cameras, therefore we perform a design study to maximize this flight space.

Lightray Simulation

A model is designed in Solidworks [385] to simulate the coverage of our cameras. Figure 2.8 simulates the camera coverage on a table the size of the Flight Arena. A key factor is the camera's pitch angle down from the horizontal. Figure 2.8 compares an orientation at 45° from the horizontal to one at 30 degrees.

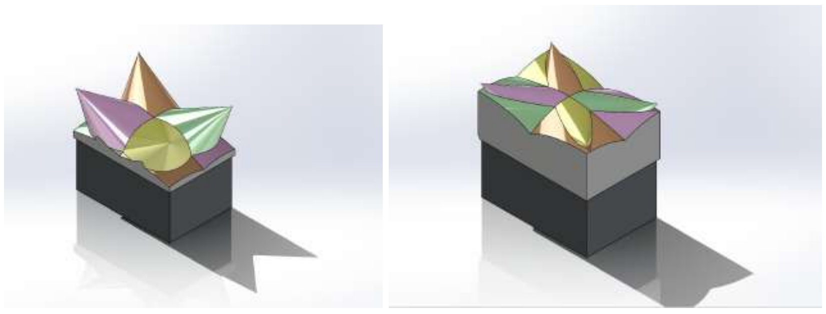
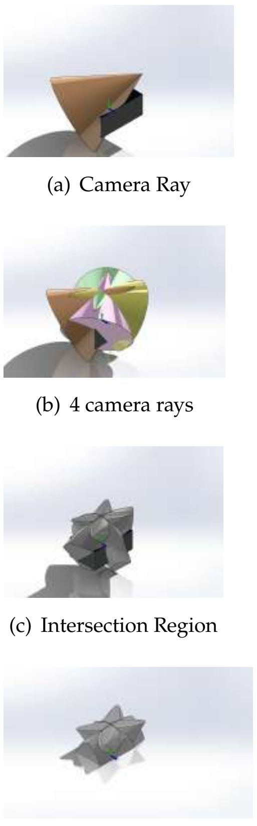

(a) Isometric view of 45deg rays (b) Isometric view of 30deg rays

Figure 2.8: Lightray simulation on the Table for 45° angle from the horizonta

The coverage percentage is determined as the volume of space localized by the flight cameras over the total usable volume above the Flight Arena. Figure 2.9 shows the modelling process of ray coverage volumes. The design requirements are as follow:

- The cameras are placed above the table corners. within the net region so as to have a clear view of the drones when the net is lowered
- There are a total of 4 cameras available during motion capture installation. The flight space measures 3×2×1.3 m.
- Flex 13 cameras have a 56° field of view, and this is replicated in simulation (Figure 2.9).
- In order to triangulate a position, the motion capture requires a minimum of 2 rays to intersect [384].

These volumes can then be determined in Solidworks using its Volumetric Tool [386]. For a pitch angle of 30^o, The volumes of the flight space and of the intersection area above, are respectively of 7.8 m^2 and 6.134 m^2. As a result, we determine that the usable region for flight is 78.64% of the 3×2×1.3 m flight space. This demonstrates that 20% of the flight space is unusable. This is not surprising, considering that 4 cameras are directly above the table and constrained by the netting.


(a) Camera Ray

(b) 4 camera rays

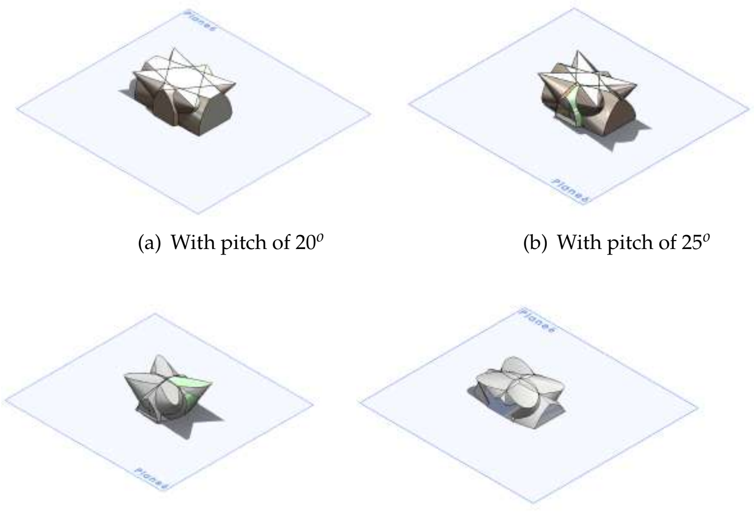
(c) Intersection Region

(d) Covered Volume

Figure 2.9: Modelling the Coverage Volume

Coverage Optimisation Study

The camera's pitch angle is varied to determine the point of optimal coverage. Figure 2.10 shows the volumes generated during the study.



(a) With pitch of 20^o

(b) With pitch of 25^o

(c) With pitch of 30^o

(d) With pitch of 35^o

Figure 2.10: Lightray simulation for Intersection of any 2 lightrays.

The study has two parts. The first study varies by increments of 5^o, in order to determine the range of volume maxima. The second study finetunes by increments of 1^o, with the help of a Solidworks Design Study [386]. This simulation tool is used to generate volumes automatically. It has trouble generating volumes if the angle increments are too large, therefore the first study is done manually.

Table 2.2: Results of Coverage Optimisation Study.

(a) Finding Local Maxima with Manual Study

Pitch (deg)	Volume (mm^3)	% Covered	Max Range
10	6 607 805 800.35	84.7154	○
15	6 970 142 481.03	89.3608	●
20	7 014 173 181.97	89.9253	●
25	6 804 720 310.30	87.2400	●
30	4 855 500 995.33	62.2500	○
35	5 006 693 602.87	64.1884	○
40	3 747 076 773.66	48.0394	○

(b) Finetuning with Automatic Design Study

Pitch (deg)	Volume (mm^3)	% Covered	Max Range
16	7007549683.27	89.8404	○
17	7034381844.70	90.1844	○
18	7050625519.44	90.3926	●
19	7056144612.83	90.4634	●
20	7014172714.71	89.9253	●
21	6999358686.15	89.7354	○
22	6972124192.42	89.3862	○

Through these studies, the coverage volume was increased from 78.64% by 11.31% up to 89.95%, and by 0.51% to 90.46%. This demonstrates that about 10% of the flight space is still out of reach. While the netting constraint forces the cameras to have this inconvenience, the study could be further optimised by varying the yaw angle of the cameras and moving them away from the corners.

2.4.4 Flight Stability Tests

Tests of the stability of robotic systems are routinely performed to measure their robustness to external forces. This is a key challenge in drone development [382], where a drone maintains dynamic stability by counterbalancing six directions of freedom, as opposed to two for wheeled systems in static stability.

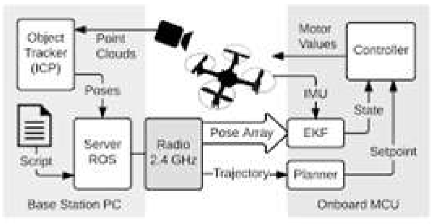


Figure 2.11: Overview of the Crazyswarm Control Loop, as per the Crazyswarm official documentation (August 2021) [379]

The Crazyswarm ecosystem [382] makes use of a position and rate controller for each drone in its swarm. This means that a setpoint is sent to each drone separately, each correcting their current pose towards the setpoint. The streaming setpoints are broadcasted from one or more antennas. As the number of drones in the swarm increases, they receive less frequent broadcasts from the antenna [380].

The purpose of this test is to determine the response of the drone's position and angle controllers to the natural disturbance during hovering. This experiment investigates the effect of antenna distance and interference on drone flight. With multiple drones to a single antenna, we evaluate if the system demonstrates any performance limits.

Hypothesis

The hypothesis is as such: the error in drone pose will correlate with the distance of the drones from the antenna.

Prediction

A hover stability test is a good measure of system performance since it requires quick readjustments of the drone to counter natural disturbances during hovering. In [360], Michał Waliszkiewicz et al. determine the performance of their flight controller by comparing the attitude of the drone in relation to the demanded null value of angular rotations. In contrast, our input is a setpoint. The output is a set of translation and rotational angles relative to a demanded null value for translation and rotation. This output is graphed as a deviation over time. The shape of the response charts are associated with flight stability over time.

Experiment Methodology

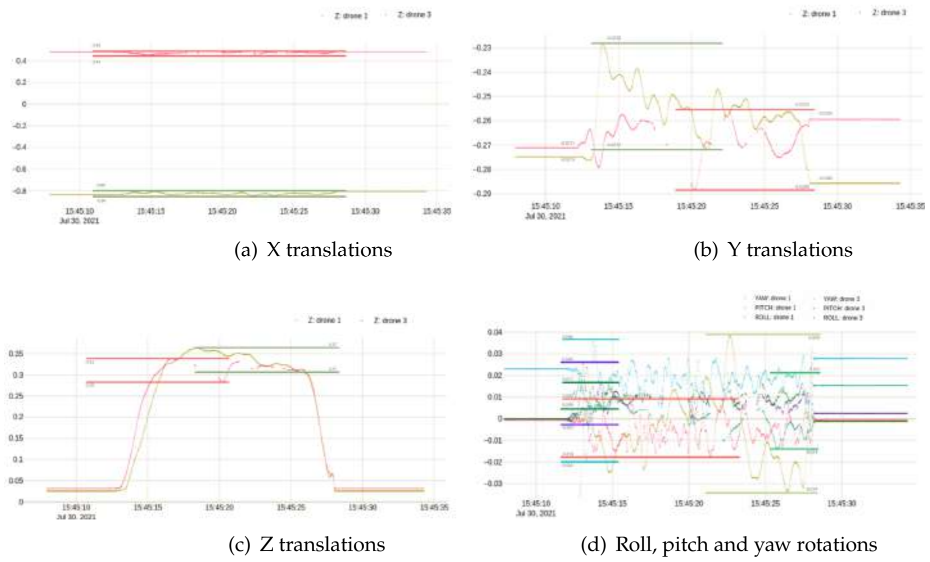
Hover stability is examined on the Flight Arena. The telemetry recording and external video cameras are programmed to launch with the swarm control interface. Three drones are hovered in the Flight Arena at an altitude of around 1 m. It was possible to record a 20-s long autonomous flight during which the flight controller attempted to stabilize the quadcopter. During that time, the quadcopter remained within a radius of two meters from its takeoff location.

Figure 2.12: View of the Flight Arena during the 2 Drone Hover experiment.

Constraints: In preparation for the flight, each of the three drones are inspected for minimal positional displacement of less than (1cm + 0.01 rad). This ensures fully functional position controllers for the drones.

Results

The flightpaths of the three drones are plotted alongside . The topview and the sideview are featured below.



(a) X translations (b) Y translations

(c) Z translations (d) Roll, pitch and yaw rotations

Figure 2.13: Hover Experiment: Stability Tests on Flight Arena.

The flightpaths are smooth and generally show very minimal jerking. There are very little discontinuities, attesting to a continuous localization process.

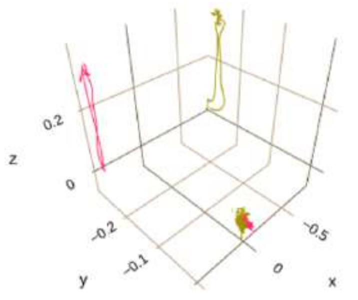


Figure 2.14: 3D Plot of 2 Drone Hover.

Table 2.3: Stability Comparison of two Drones.

Criteria	**Drone 1** Min	Max	Range	**Drone 2** Min	Max	Range
X	-0.84	-0.80	0.04	0.44	0.48	0.04
y	-0.0272	-0.0228	0.0044	-0.0289	-0.0255	0.0034
Z	0.31	0.37	0.06	0.28	0.34	0.06
Roll	-0.018	0.009	0.027	0.009	0.017	0.008
Pitch	-0.020	0.036	0.056	-0.003	0.026	0.029
Yaw	-0.034	0.039	0.073	-0.014	0.021	0.035
Selection			✗			✓

Drone 2 has less variation in both translations and rotations than Drone 1. This is confirmed in Table 2.3. Drone 2 is more stable in this test than Drone 1. The sample hover error is ±72.24 mm ±0.096 rad.

The discrepancy between the two drones could be attributed to a number of factors. This work may be improved with a second test, where the two drones' positions are inversed. All in all, the flight is substantially accurate, with a peak translation of 6cm.

Conclusion of Test

Drone 2 is further from the arena and exhibits more stability. The drone that is furthest from the antenna does not have more pose error, and the hypothesis is rejected.

2.5 Software Environment

2.5.1 Modules involved in Software Environment

This section is a brief mention of all the platforms, systems, services, and processes the software environment would depend on.

- Motive [383] processes OptiTrack camera data to deliver global 3D positions, marker IDs and rotational data.
- Crazyswarm [380] is an swarm management layer that allows multi-drone flight of Bitcraze Crazyflie drones in tight, synchronized formations,
- ROS [387] is a set of software libraries and tools that assist in building robot applications.
- SMACH [388] is a task-level architecture for rapidly creating complex robot behavior and integrating ROS utilities,
- Unity [389] is a cross-platform game engine used in a range of mixed reality research [390][376][391].

For the sake of replicability, the version of each module is documented in the references.

2.5.2 Operating Systems

We adopt a distributed systems approach, whereas various components are spread across multiple computers on a network. These devices split up the work, coordinating their efforts to complete the job more efficiently than if a single device had been responsible for the task. This section is a brief description of relationships between the modules and system features. Figure 2.15 encapsulates the software modules into their respective operating systems, Ubuntu and Windows.

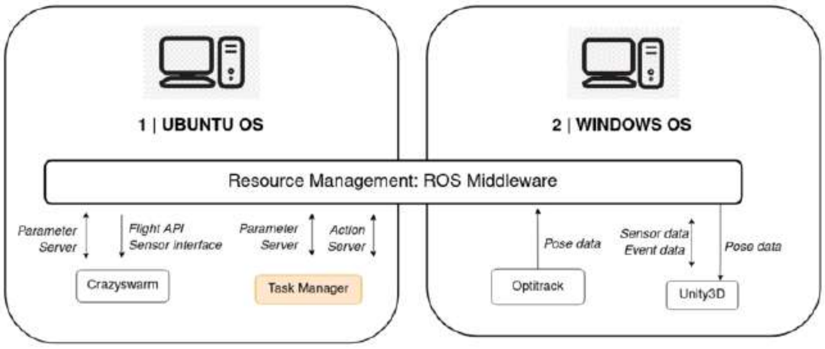


Figure 2.15: Network Interfaces Encapsulated in Operating Systems.

Each OS accommodates compatible software technologies used in this architecture. Optitrack and Unity have been developed for Windows systems. A Windows 10 OS is loaded on a standalone PC. On the other hand, ROS have been developed for Ubuntu systems. An Ubuntu 18.04 OS is loaded on a standalone PC. The interface between the two is managed by the ROS middleware, which is expanded upon in Section 2.5.3.

2.5.3 Middleware Solution

In a middleware [392], modules do not need to be linked within a single process, and this instead can be separated into the following elements.

- **Package management**: drivers and other algorithms can be contained in standalone executables,
- **Hardware abstraction**: in software, this refers to a sets of routines that provide programs with access to hardware resources through programming interfaces. This is explored in Section 2.6: Swarm Programming Interface.
- **Low-level device control**: the ROS interface serves as a communication layer with onboard devices such as motors and the battery sensor,
- **Message exchange between processes**: inter-process communications allows to pass data between modules, such as data from drone poses shown in Figure 2.16.
- **Managing robotics-related functionalities**: handling the concurrent activity of multiple robots via a global parameter manager and a global task manager.

The main objective for this system's Middleware Solution is a more flexible, more reconfigurable and generally modular layout. This proves useful in a development and demonstration environment that requires many critical moving parts. This system's network interface is shown in Figure 2.16.

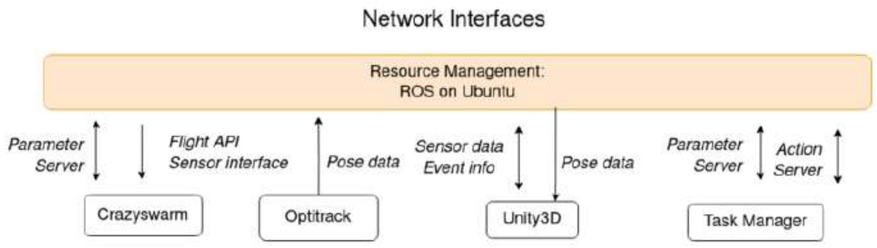


Figure 2.16: Network interfaces with ROS.

ROS provides a central role of **resource management**, from managing various interfaces in the system implementation to further hardware abstractions.

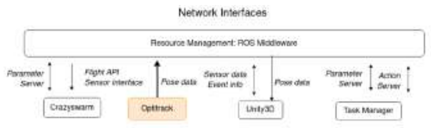


Figure 2.17: Swarm solution interactions with System Architecture

2.5.4 Virtualisation of Physical Objects

Once localized by the motion capture setup, pose data is transferred to the middleware layer. The pose data of physical objects, including the drones, becomes available in real-time to a range of companion software, via this ROS middleware layer.

2.5.5 Swarm Management Layer

The Crazyswarm framework [379] is adopted as an control layer for the Crazyflie drone. The main advantages of the Crazyswarm over other frameworks are:

- **Motion capture integration**. Crazyswarm contains drivers for the Optitrack System. In contrast, the Crazyflie proprietary API can send position measurements to the Crazyflie, but does not know how to get position measurements from mocap hardware.
- **Python firmware bindings**. Crazyswarm's simulator is built upon automatically generated Python bindings for certain modules in the Crazyflie firmware. The binding system can be helpful when developing new firmware modules, especially when they are mathematically complex and hard to debug.
- **ROS foundation**. The Crazyswarm server program is a ROS node. The Python API Reference is a thin wrapper around the ROS interface. The ROS interface is explored in this section.

2.5.6 Simulation Environment Layer

The first objective of the simulated environment is to serve as a graphical interface in order to develop tasks otherwise too difficult to deploy. The priority of the virtual reality is therefore set on rendering capabilities, and the ability to obtain camera streams from this environment. The robotics backend, described in the previous elements, can interact with the Unity3D game engine.

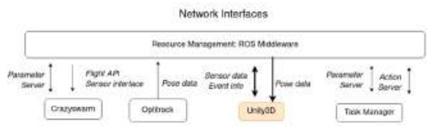


Figure 2.18: Simulation environment interactions with System Architecture

As shown in Figure 2.18, ROS has a steady stream of poses from the physical drones, allowing for virtual visualisation. Key events and data can be exchanged between ROS and Unity3D. The way this is achieved is examined in Section 3.4.

2.5.7 Task Management Layer

A Task Manager assists in the scheduling of flight tasks relative to one another. The task manager has multiple responsibilities in this framework.

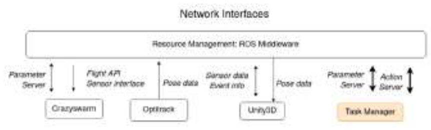


Figure 2.19: Task Manager interactions with System Architecture

- First, it loads the description of all tasks.
- It then provides a service to start or stop a given task,
- It keeps track of the status of all tasks currently running or recently terminated.
- It is also responsible for instantiating the task scheduler that manages the threads in which tasks actually run.

This manager is implemented with a Client-Server communication as seen in Figure 2.20.

The Client directly tracks the state of each process in a larger decision process. The Action Server interacts with automated functionality, and the Flight Server interacts with the robot instruction stream. The building blocks of this approach are:

1. A Client State Machine
2. Client-Server Messages
3. Server Handling of Actions
4. A Distributed Parameter Handler
5. Scaling to Multiple Drones

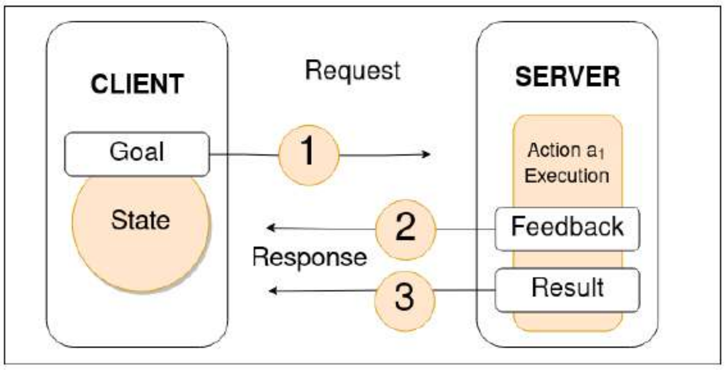


Figure 2.20: The client-server interaction.

1 | A Client State Machine

The client requires a decision-maker between each state and a set of possible future states. A state machine is chosen to coordinate the transition between different usecases. For this, the SMACH library is used [388]. Task handling is implemented with several scheduling elements.

- **Concurrency**: the ability for a program to be decomposed into parts that can run independently from each other. This means that tasks can be executed out of order and the result would still be the same as if they are executed in order.
- **Preemption**: the act of temporarily interrupting an executing task, with the intention of resuming it at a later time. This interrupt is done by an external scheduler with no assistance or cooperation from the task.
- **Interruption**: a process tells the task manager to stop running the current program so that a new one can be started.

2 | Client-Server Messages

A message transmits data values during client-server communication. ROS uses a simplified messages description language [392] for describing the data values (aka messages) that ROS nodes publish. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages.

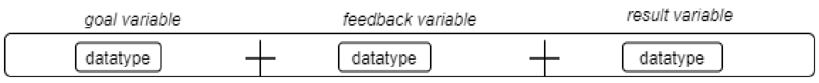


Figure 2.21: Overview of the ROS Message structure.

In an Action Client-Server interaction, communication is ensured ROS Messages with three distinct roles: the goal, the feedback and the result [392]. An action is executed when the goal requests an action with a set of parameters to the server. Feedback parameters can selected for monitoring during the action's execution. The result informs any concurrent threads of the final state of the action.

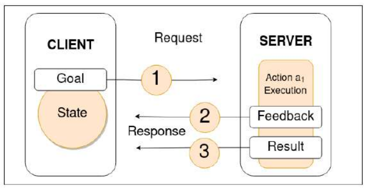


Figure 2.22: The client-server interaction.

3 | Server Handling of Actions

The Action Server executes an action in the form of a callback functions. A task completes when a particular condition is met. In order to manage this, an action callback can incorporate a condition in its execution. The next section examines this in more depth.

Each of these pre-loaded behaviours needs to be scheduled. For this, an open-source Function Handler is used, referred to as the ROS Action Server [392].As a result, each function for a specific task server will be launched from a central **launchfile**:

```
    <launch>
    <group>
        <remap from='_goTo' to='drone1_goTo'/>
        <node name='drone1' pkg='crazyswarm' type='ros_action_server.py'>
        </node>
    </group>

    <group>
        <remap from='_goTo' to='drone2_goTo'/>
        <node name='drone2' pkg='crazyswarm' type='ros_action_server.py'>
        </node>
    </group>
    </launch>


```

Figure 2.23: Example of Server Launchfile with multiple concurrent uses of the same program.

This ROS launchfile loads the functions declared in the Action Server, and remaps them to each drone in the choreography. This allocates a thread under the form of a ROS node. These ROS nodes act as separate Request/Response instances.

4 | ROS Parameter Handler

The ROS main thread includes a commonly-used component called the Parameter Server, implemented in the form of XMLRPC, and which is, as the name implies, a centralised database within which nodes can store data and, in so doing, share system-wide parameters.

```
        crazyflies:
        - channel: 35
          id: 1
          initialPosition: [0.0, 0.0, 0.0]
          type: default
        - channel: 27
          id: 2
          initialPosition: [1.0, 0.0, 0.0]
          type: default
        - channel: 27
          id: 5
          initialPosition: [4.0, 0.0, 0.0]
          type: default
```

Figure 2.24: Parameter File (.yaml)

Multiple programs query this file upon initialization of the swarm management layer. Each robot is distinguished by their channel, id and initialPosition. This allows for identifying drone ids and other unique information.

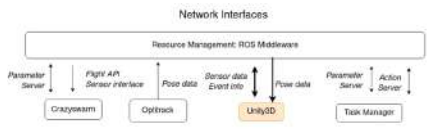


Figure 2.25: Task Manager interactions with System Architecture

5 | Scaling to Multiple Drones

Similarly to Cedric Pradalier[393], a procedural task-based programming approach is adopted. This can be likened to a centralized server that services multiple drones. Figure 2.26 shows the full Task Management Layer Architecture.

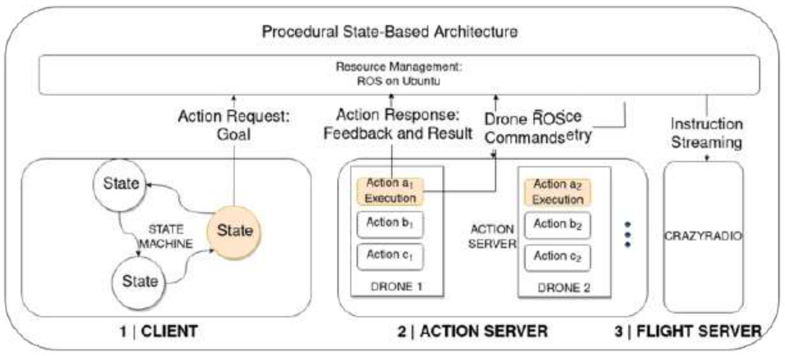


Figure 2.26: Task Management Layer Architecture.

This approach values granularity, being lightweight, and the ability to share similar processes across multiple apps. As a result, it is therefore highly reusable for new tasks.

2.6 High Level Interface

A high level interface is an abstraction layer for development activities. In order to simplify task development, and align with the thesis goals, we develop a framework for high level interaction between the operator and the functionalities of the testbed.

2.6.1 Motivation

The Testbed, as described in Section. For such purposes, it is required to test user code. Therefore an interface is a key element for the user. There are several advantages to specialised tasks for the testbed.

- **Handling sub-tasks** to various levels of depth: microservices help automate sub-tasks at a desired complexity. When encapsulated in this way, are separate modules fit for demonstration, that can later be optimised and refined during development.
- **Monitoring the swarm**: a central monitoring system can run in parallel with the particular algorithms that are tested and validated. For instance, a battery voltage threshold helps to monitor a correct running of the hardware.This allows for preventive maintenance during demonstrations but also during development.

In summary, more 'complex tasks' will allow for the automation of separate subtasks in a controlled manner.

2.6.2 Conceptual Overview

The high-level interface combines three major elements:

- the management of low-level devices upon each robot,
- communication with the swarm, and
- scheduling of instructions.

In order to achieve this, the intermediary structures for tasks are laid out here. Figure 2.27 labels a hierarchy of tasks. The drone is instructed to alternate between two waypoints until a software condition is triggered.

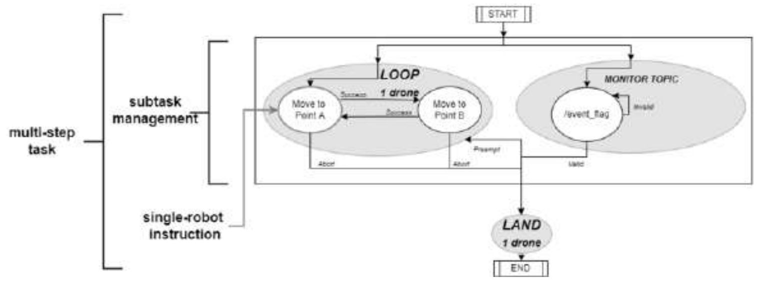


Figure 2.27: Example state machine to implement individual tasks.

This example serves to illustrate the conceptualisation of a subtask and a multi-step task. The concurrence of waypoints and the software trigger is consider a sub task, and within a state machine, which is referred to as a multi-step task. This framework offers a definition for complex tasks, as tasks that coordinate the scheduling of instructions, with multi-robot instructions.

2.6.3 Architectural Approach

To create these complex tasks, this high level interface has the following architectural choices:

- **Encapsulating robot instructions** Robot commands are assimilated into this programming interface as individual tasks.
- **Encapsulating swarm instructions** Robot instructions are included in generic functions as swarm instructions.
- **Encapsulating sub tasks** Scheduling processes such as a concurrence runs in a function encapsulating it.

This architecture is written in Python, known for its ease of use and flexibility. In this way, multi-step tasks manage the swarm stack, from executing single-robot commands to ensuring the dynamic management of swarms.

Robot instructions

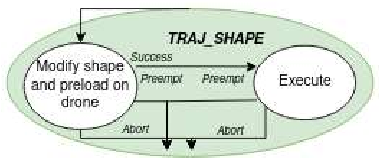


Figure 2.28: Example of a robot Instruction.

The Crazyswarm API from Section 2.5.5 interfaces with low-level hardware for landing, takeoff and further behaviours that can be coded remotely. However, for the purpose of **centralised task management**, the execution of each instruction should be monitored accordingly. Thai Phan, Wolfgang Honig, and Nora Ayanian [376] offer ROS telemetry tools, such as battery monitoring and a reset utility. These can be used as conditions in the execution process. Ultimately, an additional layer of abstraction is required for multi-drone instructions.

Multi-robot instructions

This section outlines the functions developed **for group behaviours**: concurrent takeoffs and landings, querying multiple drones for low battery level, etc. Fly-Octogon and Land-all are examples of multi-robot instructions.

This microservice model is a major component of optimizing swarm programming towards the multipurpose task model outlined in the objectives.

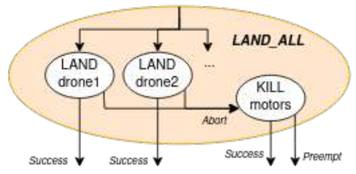
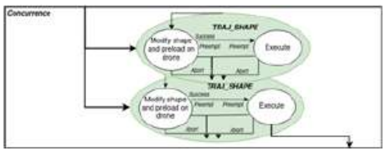


Figure 2.29: Example of a multi-robot instruction.

Sub task management

The objective for sub tasks is to assist in creating, and coordinating, **higher-level behaviours**. An example of this is the concurrent_traj module, whereas two drones are told to fly simultaneous trajectories. This is of particular interest as two drones will take **indeterminate amounts of time** to respond to commands. **Concurrency** — in the context of programming — is the ability for a program to be decomposed into parts that can run independently of each other.


Concurrence
TRAJ_SHAPE
Modify shape and preload on drone
Execute
TRAJ_SHAPE
Modify shape and preload on drone
Execute
Figure 2.30: Example of a sub task, that includes individual robot instructions, but can also include multi-robot instructions.

Multi-step tasks

A decision process combines the various modules developed above into a sequence of tasks. This is achieved with a Finite State Machine, which is implemented programmatically with the SMACH python library [388]. A choreographic state machine is implemented in section 2.7.

2.7 Testbed Demonstration

A drone choreography is designed as a live demonstration of the Testbed's functionality. The experiment data is accessible publicly [361].

2.7.1 Choreography Design

The State Machine for the full choreography is available in Figure 2.31. This demonstration includes:

- Takeoff and landing, separately and concurrently.
- A pre-loaded trajectory, from a Bezier curve: concurrently.
- A polygonial shape flown by two drones, demonstrating simultaneous movement through a set of waypoints.
- Autonomous state changes.

This state machine functions any number of drones: using the swarm building blocks developed in Section 2.6, the dronesexecute trajectories simultaneously; it then moves to certain waypoints indefinitely. In this

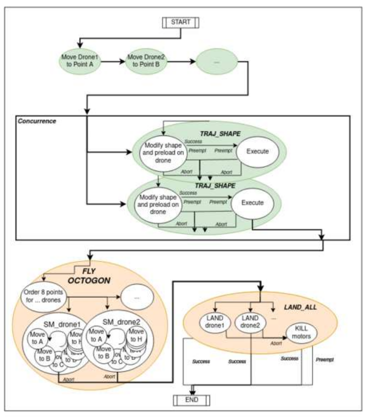


Figure 2.31: Choreography Design: State Machine

case a figure of 8 is executed on both drones followed by an octogon. Finally, upon an operator signal, the drones land. The state machine is such that the drones also land if one does not reach its corresponding waypoint in time.

1 | Pre-loaded Trajectory

Two Figures of 8 are flown simultaneously. The figures of 8 are concurrent Bezier shapes, pre-loaded onboard each drone's trajectory follower [379]. This is coded using the high level interface as in Fig 2.33.

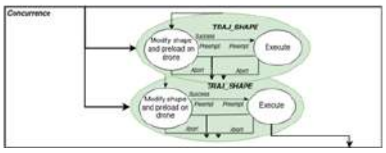


Figure 2.32: Simultaneous pre-programmed trajectories.

```
        fig8_sm = concurrent_trajs(selected_drones = ids, traj_id = 8)
        StateMachine.add('FIG8_EXECUTE', fig8_sm,
                        transitions={'succeeded' : 'NEXT_STATE',
                                     'aborted' : 'land_all',
                                     'preempted' : 'land_all'})
```

Figure 2.33: Integrating a pre-loaded trajectory in the State Machine

The Figure of 8 is assigned an id of 8. Other trajectories are assigned other ids. The concurrent_trajs function is thus called upon with the required drones and their required ids.

2 | Multi-point Trajectory.

This state loads a custom trajectory on the drone, which is executed, before moving to an indefinite octogonal trajectory. The use of **waypoint following** is an automation of the motion to specific points.

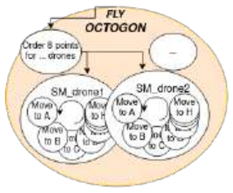


Figure 2.34: Multi-drone and multi-point loop.

3 | Topic Monitor

The use of a **Topic Monitor** is useful to interface with active topics. For instance, at any one moment that a drone gets too close to a particular point, it initiates a landing. The intended behaviour is represented visually alongside.

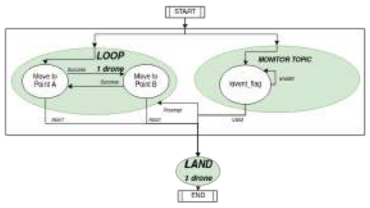


Figure 2.35: Sub task: monitoring an active topic.

This is another such subtask that fulfils the initial goal: monitoring the swarm with preventive measures during demonstration as well as training phases. This is performed programmatically with a concurrence between a drone and the /collision topic.

4 | Choreography State Machine

The previous sections are integrated into a State Machine. The configuration of the state machine is displayed in Figure 2.36. Individual tasks are coloured in green and swarm tasks in orange.

Three drones are positioned about the Flight Arena as in Figure 2.37.

5 | Choreography Execution

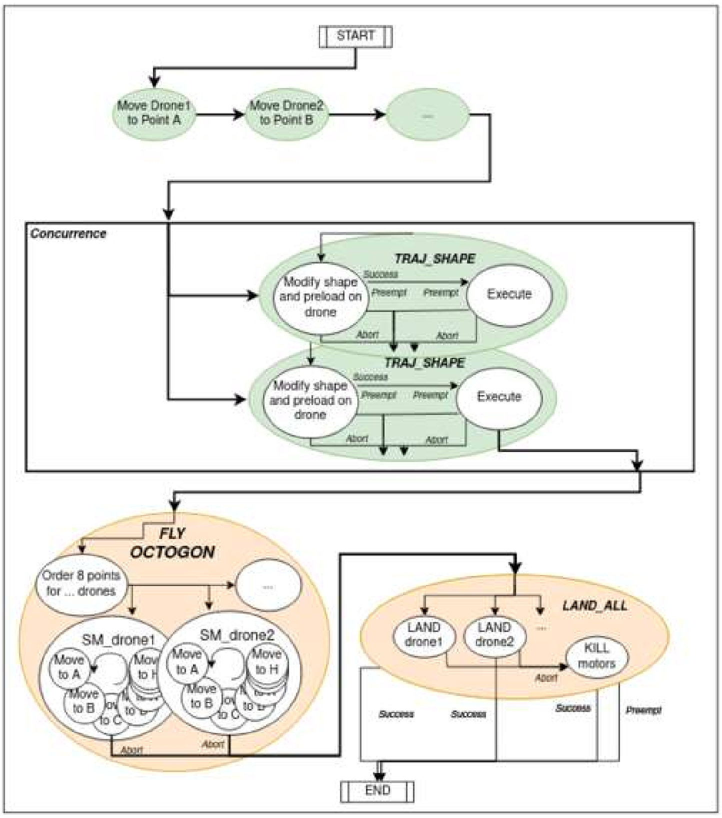


Figure 2.36: State Machine Visualisation

Figure 2.37: View of the Flight Arena during Experiment.

The state machine is run on a separate thread as in Figure 2.38.

```
myswarm = swarmInterface(all_ids = [1,3,5])
sm0 = myswarm.execTrajandOctogon (ids = [1,3], traj_shape = 8)
myswarm.start_sm_on_thread(sm0)
```

Figure 2.38: Execution of Multi-step tasks

This invocation of the state machine clearly shows drone ids [1,3,5] as extracted from the Parameter Server, in order to act as the drones 1,2,3. The trajectory shape 8 refers to the Figure of 8.

2.7.2 Results

We proceed with an inspection of the demonstration. The flightpaths of all three drones are plotted together.

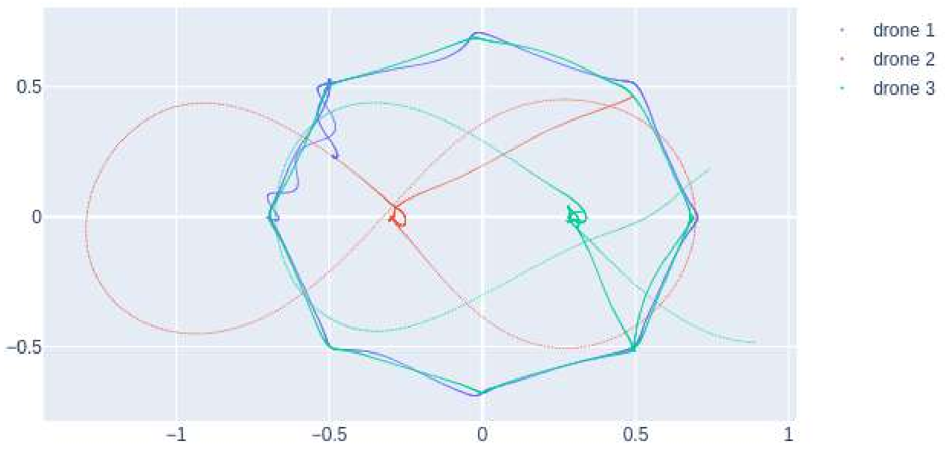


Figure 2.39: Full Flightpath of all three drones during the Choreography.

Overall, the flightpaths are smooth. The Figures of 8 are traced distinctly, as well as the two octogons. The figure of 8 of drone 3 is discontinuous, and yet there is no apparent effect on the shape. This suggests that the drone moved beyond the area localized by motion capture. When examining the octogons, the top view shows a near perfect superposition: showing small differences in position of less than 2cm. Finally, a line connects the two shapes. This shows that these figures did actually occur in sequence.

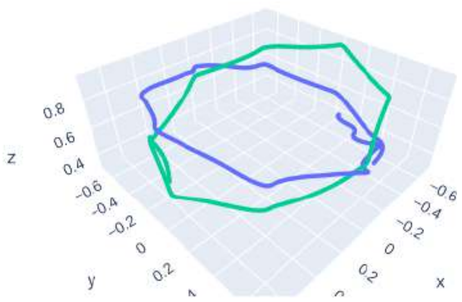


Figure 2.40: 3D Plot of Octogon Figure.

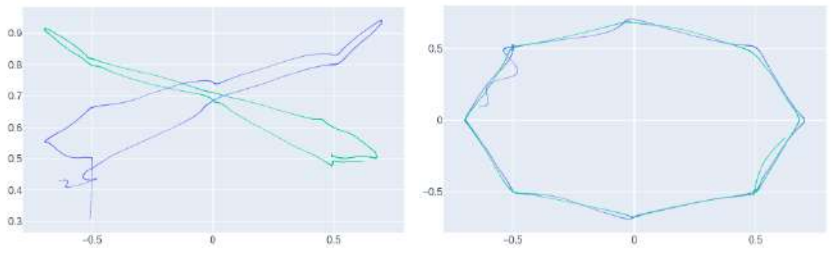


Figure 2.41: Hover Experiment: Stability Tests on Flight Arena.

Further inspection of the octogons requires a topview and a sideview. The Octogon is traced very clearly. Near the end of the experiment, there is a noticeable wobble in the blue line. This behaviour is due to a low battery level. A notable difference is the wobble in the xz plane, which demonstrates a loss of precision as the drones get closer to the ground. There is a symmetrical behaviour. Further tests can determine what this is attributed to.

Table 2.4: Key findings in Chapter 2.

Test Description	Value
Volume of Flight Arena Localized by Motion Capture	90.46%
Maximum Flight Error Recorded in Hover Test	±72.24 mm ±0.096 rad

2.7.3 Discussion

In this chapter, a swarm programming approach is developed along similar lines to [378], the swarm API that was developed for swarms of nanodrones. Both frameworks load a set of functions, they allow the user to select which drones perform a certain task, and group drones according to the task at hand. While Buzz manages membership with a dedicated hash table, our interface makes use of a global parameter handler [373]. Both architecture allows for the development different modules can be developed independently and related dynamically.

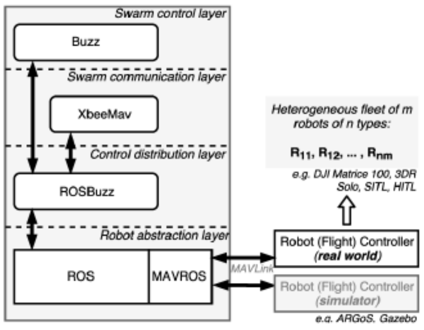

Swarm control layer
Buzz
Swarm communication layer
XbeeMav
Control distribution layer
ROSBuzz
Robot abstraction layer
ROS
MAVROS
MAVLink
Heterogeneous fleet of m robots of n types:
$R_{11}, R_{12}, \ldots, R_{nm}$
e.g. DJI Matrice 100, 3DR Solo, SITL, HITL
Robot (Flight) Controller (real world)
Robot (Flight) Controller (simulator)
e.g. ARGoS, Gazebo

Figure 2.42: Swarm stack in the Rosbuzz system [373]

With a high-level interface, this work concerns itself with a swarm-specific language that is not "too top-down" or "too bottom-up". This distinction is seen with increasing swarm sizes, and for creating user tasks more focused on development, or on improving safety and phone interference for demonstrations.

Figure 2.42 demonstrates that Buzz uses comparable structures for swarm engineering.). ROS communicates with device hardware via MAVROS, a ROS library for compatible Micro Aerial Vehicules. It then has a control distribution layer that is comparable to our Task Manager, a swarm communication layer like Crazyswarm and a swarm control layer like our high level interface. The swarm stack in other research may be composed of other technologies, but retains this structure.

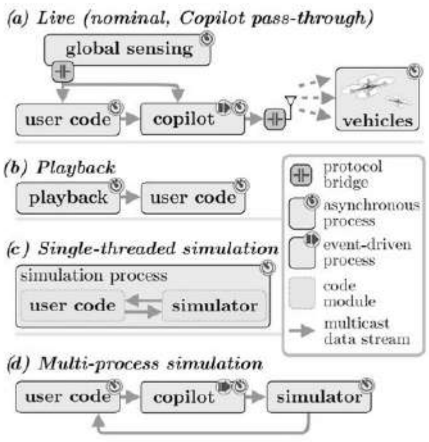


Figure 2.43: Copilot on the Flying Machine Arena [375]

The Flying Machine Arena [375] is an active area of research for drone development and demonstration, and their 'Copilot' is described as a flight monitoring solution. Figure 2.43 demonstrates the types of activities achieved via the copilot: updating drone poses during code execution (a), executing playback on recorded poses (b), and executing procedures in simulation (c). These elements are handled by the Task Manager described in this chapter, demonstrating the pertinence of a flight management solution.

The procedural task-based approach of this chapter is not unique: it figures in [393] who develops a generic pythonic form that need not depend on middleware for task management. All in all, this approach can aid in development, in ways that can be outlined here.

- preventing mechanical failure upon software failure.
- assist in creating, and coordinating, higher-level behaviours.
- monitor the state of every UAV asynchronously.
- Assist in troubleshooting with a modular layout.

2.7.4 Summary

This flight has demonstrated multiple working functionalities. The first is the use of the Swarm Programming Interface. Using the building blocks developed in this chapter, it is possible to develop a multi-stage process, one that includes preloaded trajectories as well as waypoint trajectories, choreographic positioning, and escape cases upon a system abort. With such tools for assistance during development, this set of functionalities pushes beyond previous work, as it offers a layer beyond the crazyswarm's robot instruction set.

2.8 Chapter Summary

The proposed state-based architecture is a first step towards creating UAV operations to perform complex tasks, collaboratively or otherwise. After all, this framework has put in place the monitoring tools and the task-based framework to execute complex behaviours; and beyond that, putting in place a Flight Arena has already helped to validated these tools. Such services can easily be tested and deployed from a framework like this one.

Certain elements in this framework are taken a step further in Chapter 3: the ability to send streaming setpoints to a drone opens the possibility of flight piloting through other means. This would not be possible without the foundation established in this chapter: the drone architecture, the motion capture setup and the swarm framework.

The tools that were established here may need to be challenged by further research. One direction is the decentralisation of agents with respect to the platform: where this framework has a central role in allocating behaviours, one would opt for a framework that gives each agent the ability to act independently. However, the central monitoring can remain a major asset when developing such a swarm, as it serves as a safety recourse to prevent any hardware damage.

The testbed makes great use of distributed networking, and it aligns with the first approach of the thesis for task creation. From handling specific parameters, to managing the task execution and scheduling in a centralised manner, the middleware monitors the different agents in an asynchronous manner. As opposed to non-distributed systems, such as direct one-to-one links to onboard devices, it allows the developer to divert their focus from system communication to performance-critical applications.

Experimentations for Human-Drone Interfaces 3

3.1 Introduction

In *The state-of-the-art of Human–drone interaction: A survey* (2019), Dante Tezza and Marvin Andujar define Human-Drone Interaction (HDI) as a field of research that consists of understanding, designing and evaluating drone systems for use by humans, and in contact with humans. This field is similar to human-robot interaction (HRI), however, a drone's unique characteristic to freely fly in a 3D space, and unprecedented shape makes human-drone interaction a research topic of its own. Researchers develop control modalities and better understand means of communicating with a drone.

Human-drone interaction is a broad research field, for instance, a researcher can design new drones' shapes with friendly-like appearance, while another researcher can focus on designing new user interfaces that allow non-skilled pilots to accurately operate drones without extensive training.

In line with the thesis goal, we look at two types of smart systems that enhance the interactions between humans and drones. The first allows the human to pilot a drone through a gesture interface. The second looks to a virtual interface as a training ground for a real drone to avoid a virtual object. A distributed system is used to to communicate and coordinate these pipelines by passing messages to one another from any system. To better understand their tradeoffs, the distributed systems are explored and evaluated in this chapter.

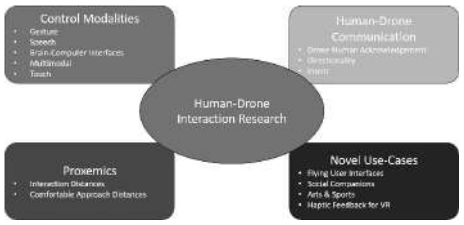


Figure 3.1: Major fields that constitute HDI [357].

Number	Name	Gesture
1	Help	
2	Ok	
3	Nothing	
4	Peace	
5	Punch	

Table 3.1: UAV rescue hand gesture dataset, from [363]

3.2 Related Work

3.2.1 Exploring more Intuitive Gesture Control

[394] *Jessica R. Cauchard et al.* focalise on innovative methods to interact with drones, including gesture, speech, brain-computer interfaces, and others (Figure 3.1). As drones have different characteristics than ground robots, such as not allowing touch interaction, it is unclear whether existing techniques can be adapted to flying robots. Their user-centric design strategy seeks to understand how users naturally interact with drones.

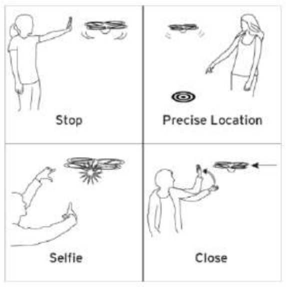


Figure 3.2: Usecase of HDI that have been incorporated in commercial drones [394]

3.2.2 Computer Vision for UAV Research

With state-of-art computer vision technology, gesture-based interaction is growing and several publications are identified.

[363] *Chang Liu and Tamas Sziranyi* contribute to an opensource database of body gestures which they test in practice with a drone (Figure 3.1). This paper contributes with an outdoor recorded drone video dataset for action recognition, an outdoor dataset for UAV control and gesture

recognition, and a dataset for object detection and tracking. These datasets are developed for emergency rescue services, which reveals how critical these applications can be.

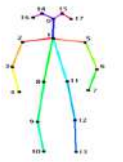
[364] *Valiallah Mani Monajjemi et al.* explores real-time vision-based Human Drone Interaction with multi-robot systems. To create a team the user focuses attention on an individual robot by simply looking at it, then adds or removes it from the current team with a motion-based hand gesture. Another gesture commands the entire team to begin task execution.

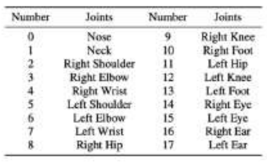
Compared to wearable sensor-based approaches, automated methods for video analysis based on computer vision technology are almost non-invasive. This is beneficial, and even critical, for applications in emergency rescue services.

3.2.3 Pose Recognition Algorithms

As a result, performance becomes application-critical for automated methods for video analysis. This, however, remains a technical challenge. According to [395], "Robust real-time hand perception is a decidedly challenging computer vision task, as body parts often occlude themselves or each other (e.g. finger/palm occlusions and handshakes) and lack high contrast patterns (e.g. between fingers)." To respond to this challenge, the Mediapipe framework [395] bases itself on a Machine Learning model, and on techniques for efficient resource management for low latency performance on CPU and GPU.

In contrast, OpenPose [363] employs a convolutional neural network to produce two heap-maps, one for predicting joint positions, and the other for partnering the joints into human skeletons. In brief, the input to OpenPose is an image and the output is the skeletons of all the people this algorithm detects. Each skeleton has 18 joints, counting head, neck, arms, and legs. Each joint position is spoken to within the image arranged with coordinate values of x and y, so there's an add up to 36 values of each skeleton.





Number	Joints	Number	Joints
0	Nose	9	Right Knee
1	Neck	10	Right Foot
2	Right Shoulder	11	Left Hip
3	Right Elbow	12	Left Knee
4	Right Wrist	13	Left Foot
5	Left Shoulder	14	Right Eye
6	Left Elbow	15	Left Eye
7	Left Wrist	16	Right Ear
8	Right Hip	17	Left Ear

Figure 3.3: OpenPose joint data and skeleton information

3.2.4 Mixed Reality for UAV Research

Simulation systems have long been an integral part of the development of robotic vehicles. They allow engineers to identify errors early on in the development process, and allow researchers to rapidly prototype and demonstrate their idea.

One of the first simulators that could recreate complex worlds in 3D is Gazebo, circa 2004 [396]. The difference between Gazebo and different 3D simulation software of that time is that Gazebo was one of the first to focus on resembling the world as realistic as possible for the robot instead of for the human. Immersive robotic simulations can be used to judge the performance of the robot and/or its concept [359]. In this way, simulators can increase the efficiency and decrease the costs of the development [396].

The first published definition of Mixed Reality (MR) was given by Milgram and Kishino [397] as the merging of physical and virtual worlds. In their definition, Augmented Reality (AR) and Augmented Virtuality (AV) are seen as special instances of MR. In Augmented Reality, virtual objects are projected onto the physical environment, while in Augmented Virtuality, physical objects are incorporated into a virtual environment.

In 'Mixed reality for robotics' [390], the definition of Mixed Reality is expanded to robotics by accommodating seamless interaction between physical and virtual objects in any number of physical or virtual environments. It is further demonstrated in [376] that Mixed Reality can reduce the gap between simulation and implementation by enabling the prototyping of algorithms on a combination of physical and virtual objects within a single virtual environment.

In drone research, immersive simulators have various applications [396], of which two are explored here:

- **Generating exteroceptive sensor data**: capturing sensor feeds of the environment for one or more drones simultaneously.
- **Testing navigation behaviour**: Testing flight patterns subject to simulated environment stimuli, prior to real-world deployment.

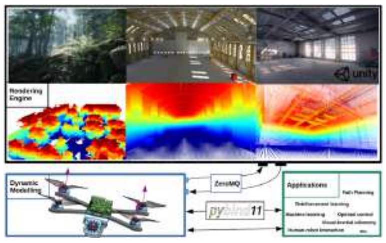


Figure 3.4: Flightmare Simulator [374]: photorealistic views (top) and spectral views (middle), followed by physics engine.

3.2.5 Generating exteroceptive sensor data

Simulation can be a huge advantage when real robot prototypes or products are not available or cannot be used due to other circumstances. During the development, simulation can be used to assess the basic hardware functionality.

For instance, FlightGoggles is capable of high-fidelity simulation of various types of exteroceptive sensors, such as RGB-D cameras, time-of-flight distance sensors, and infrared radiation (IR) beacon sensors. This example can be extended to multiple sensors simultaneously, leading the way to richer distributed swarm systems.

However, older simulators don't provide an efficient API to access 3D information of the environment [374]. To foster research in this direction, Flightmare provides an interface to export the 3D information of the full environment (or a region of it) as point cloud with any desired resolution.

3.2.6 Testing navigation behaviour

Controllers evolved in simulation can be found to be inefficient once transferred onto the physical robot, remains a critical issue in robotics, referred to as the reality gap. the most efficient solutions in simulation often exploit badly modeled phenomena to achieve high fitness values with unrealistic behaviors. This gap highlights a conflict between the efficiency of the solutions in simulation and their transferability from simulation to reality. When deploying to real-life scenarios, there are several challenges [398]:

- Optimising the flight control of a UAV. This is relevant with changing payloads, unexpected weather conditions (dust, rain, changing wind), as well as preventive maintenance (motor degradation, battery damage).

- Optimising the flight path of a UAV. Several sensor inputs can inform the drone's flight path and flight speed. This gives several ways to optimise the data acquisition process, from more complex data intakes and various activation/triggering optimisations.

Prior to real-world deployments, different functional elements on a robot can be tested in parallel and reduce development time. For instance, the algorithms for localization, motion planning or control can be tested, improved, and integrated continuously. There are various artificial intelligence algorithms concerned with the thematic of guidance, navigation and control (GNC). A subset of these algorithms is explored in, pertaining to Deep Reinforcement Learning (DRL). These techniques can improve the drone operation as shown in Table 3.2.

Table 3.2: Instances of RL tasks taken from [398]

Task	**Input Observations**	**Output Actions**
1. Quadrotor control	$[p, \theta, v]$, d=10	$[c, \omega_x, \omega_y, \omega_z]]$, d=4
2. Control (motor failure)	$[p, \theta, v, \omega]$, d=12	$[f_1, f_2, f_3]$, d=3
3. Flying (in gate)	$[p, \theta, v, \omega, p_{gate}, \theta_{gate}]$, d=18	$[f_1, f_2, f_3, f_4]$, d=4

Current research of drone quadrotor control employs newly architected neural networks and learning time-optimal controllers for drone racing [398]. This element is largely figured in Flightmare's [374] simulation usecases, and echoes the state of the art research in Reinforcement Learning for UAVs [398], suggesting that new RL implementations can optimising the flight stability of a UAV as well as new perception pipelines for the navigation of a UAV. Flightmare offers convenient wrappers for reinforcement learning. Those gym wrappers give researchers a user-friendly interface for the interaction between Flightmare and existing RL baselines designed around the gym interface.

3.3 Drone Piloting With Gesture

Gestures are the most natural way for people to express information in a non-verbal way. Users can simply control devices or interact without physically touching them. Nowadays, such types of control can be found from smart TV to surgery robots, and UAVs are not the exception.

Drone piloting and other control modalities [357] make use of various inputs to assist in flight. Perception modules for drone flight usually consist of data-driven models based on multiple sensor modalities. These inputs can be sensor modalities, such as camera, lidar, and radar, published in autonomous-driving related datasets, but also human commands, in the case on drone piloting. In this way, perception pipelines are routinely developed as a realtime interface for sensor data from multiple perception configurations.

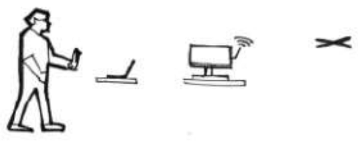


Figure 3.5: A dedicated test-and-demonstrate environment.

As of Nov. 2019, multiple gesture interfaces have been developed for UAVs [363] [364], but are lacking in drone piloting. Realtime interfaces for drone piloting are discouraged [357] due to high latency and low control precision compared to other drone control modalities. As of Sept. 2021, the literature utilizing the Crazyflie nanodrone does not include realtime streaming commands [365].

3.3.1 Overview of Pipeline

A pipeline is implemented to ensure that the right gesture is associated to the right drone command, in real-time and continuously. This pipeline is conceptualised as in Figure 3.12.

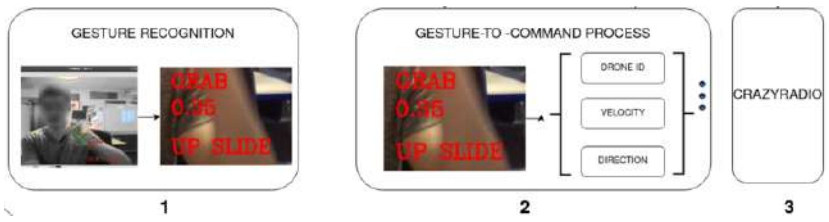


Figure 3.6: Gesture recognition workflow

The Testbed of Chapter 2 offers a working environment, as well as a command streaming interface between a drone and a companion computer. The focus is therefore on the two first elements: a gesture recognition workflow, followed by a drone control workflow.

3.3.2 Gesture Recognition Layer

In this project, we employ a **3D Pose Estimator**, described in the following section, followed by a **Gesture Classification** Script.

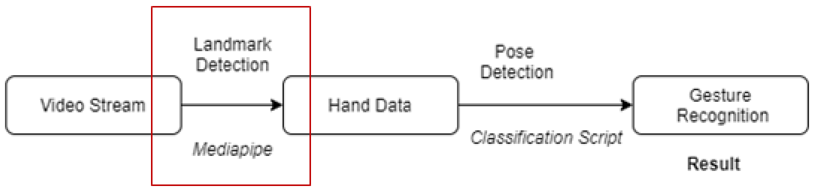
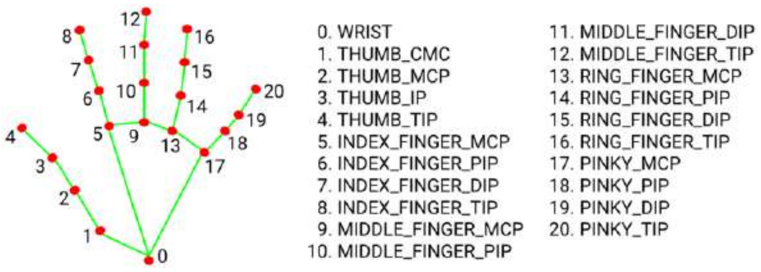


Figure 3.7: Mediapipe algorithm in gesture recognition workflow

Machine Learning 3D Pose Estimation

MediaPipe Hands [395] is a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 landmarks of a hand from just a single video frame.



0. WRIST	11. MIDDLE_FINGER_DIP
1. THUMB_CMC	12. MIDDLE_FINGER_TIP
2. THUMB_MCP	13. RING_FINGER_MCP
3. THUMB_IP	14. RING_FINGER_PIP
4. THUMB_TIP	15. RING_FINGER_DIP
5. INDEX_FINGER_MCP	16. RING_FINGER_TIP
6. INDEX_FINGER_PIP	17. PINKY_MCP
7. INDEX_FINGER_DIP	18. PINKY_PIP
8. INDEX_FINGER_TIP	19. PINKY_DIP
9. MIDDLE_FINGER_MCP	20. PINKY_TIP
10. MIDDLE_FINGER_PIP	

Figure 3.9: MediaPipe Hands data and joints information.

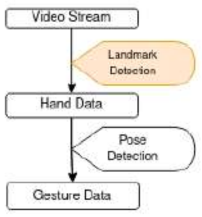


Figure 3.8: Pose detection in gesture recognition workflow

Precise key-point localization of 21 3D hand-knuckle coordinates remain inside the detected hand regions. This allows us to have the spatial position of each of the joints of a hand using only a normal camera.

Whereas current state-of-the-art approaches rely primarily on powerful desktop environments for inference [363], MediaPipe Hands achieves real-time performance on a mobile phone, and even scales to multiple hands, making it an ideal solution for real-time pose tracking.

The desired gesture is hardcoded by its absolute position. For example, if Figure 1's landmark 8 is below the landmark 5, it can be interpreted as closing your index.

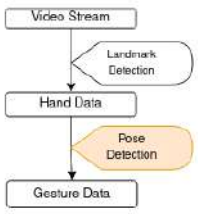


Figure 3.10: Classification Script in gesture recognition workflow

Gesture Classification Script

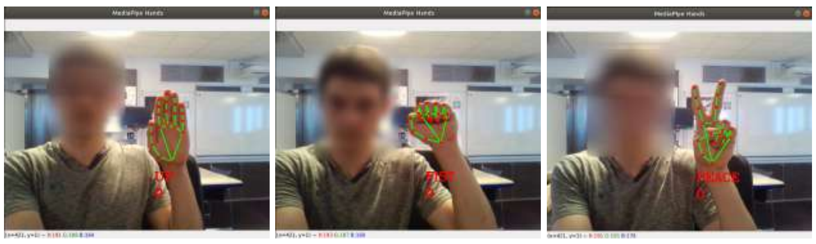


Figure 3.11: Static Hand Signals: Right, Left, Up and Down

While the model is precise in landmark detection, there are false detections of gestures due to marginal cases. A special buffer is used to filter the most frequent gesture on a sliding window (for every 10 detections). This helps to remove glitches or inconsistent recognition.

3.3.3 Drone Control Layer

Using a live gesture recognition module, a system is designed for streaming commands to be sent to the drone.

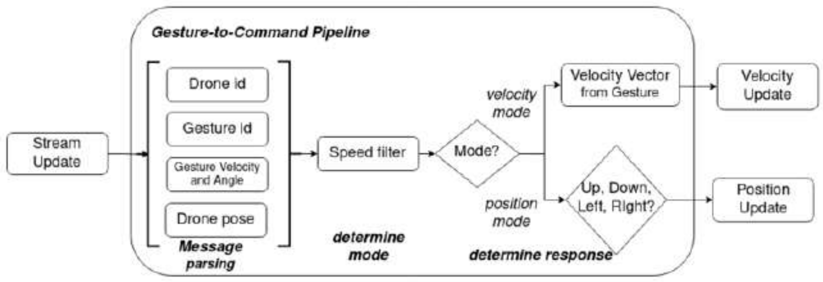


Figure 3.12: Gesture recognition workflow

Note that the critical information flow between the components of the system is unidirectional. Bidirectional communication, e.g. telemetry from the vehicles, is supported, but is not required for controlled operation. All communication is done in a distributed, one-way manner, such that the gesture recognition workflow is not affected by the drone listener and there is no reliance on high-level code to keep track of the various components, preventing unnecessary interdependence. The Gesture-to-Command script decrypts the messages encoded into a custom ROS message. This workflow serves as a fallback during experiments and demonstrations.

Message passing between applications

The following message is passed via a ROS topic (TCP/IP message). It is then depacketized upon arrival.

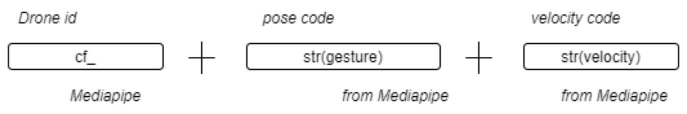


Figure 3.13: Message structure adopted for command streaming

Mode Selection

The following element in the drone control layer is the selection of an adequate mode.

- Flight Mode 1: Position Update Mode. This mode moves the drone translationally in three axes based on an absolute position.
- Flight Mode 2: Velocity Update Mode. This mode moves the drone according to inputted velocities.

In each mode, the drone can move up, down, left or right. The following hand gestures are associated with these directions.

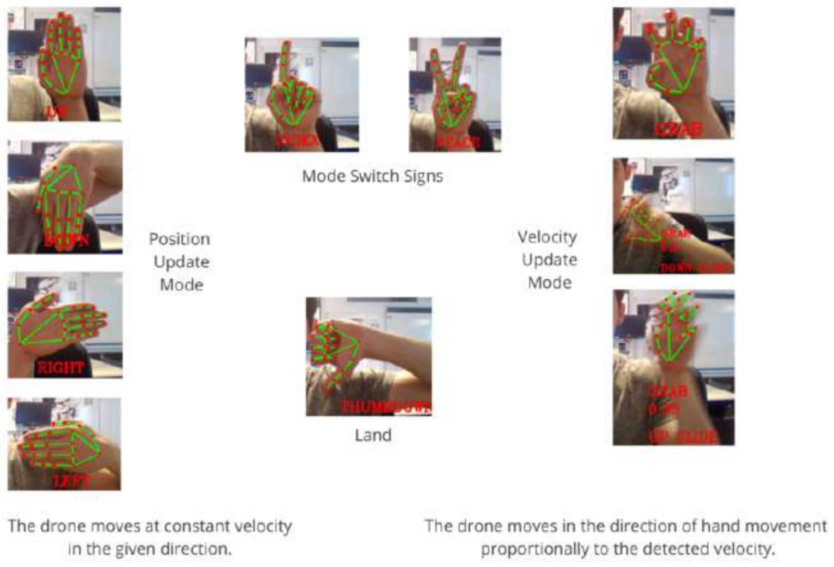

UP
DOWN
RIGHT
LEFT
Position Update Mode
Mode Switch Signs
THUMBDOWN
Land
Velocity Update Mode
GRAB
The drone moves at constant velocity in the given direction.
The drone moves in the direction of hand movement proportionally to the detected velocity.

Figure 3.14: Associating Gestures to Commands for the Gesture Piloting Pipeline

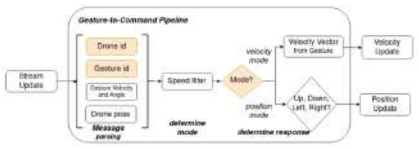


Figure 3.15: Velocity Filter in Pipeline.

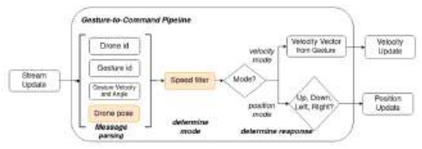


Figure 3.16: Velocity Filter in Pipeline.

Velocity Filter

The velocity filter serves as a safety measure during development and demonstrations. Given a drone's position in the Flight Arena, it attributes a particular value between 0 and 1. This value is then multiplied to the velocity value determined from the gesture movement.

The shape of this Velocity Filter was defined using an ellipsoid, with a near constant velocity within the ellipsoid and a sigmoid on the boundaries of this shape. An initial volume is designed on 3.1.

$$A_1 = \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} \text{ with } (a, b, c) = (1.35, 0.85, 1.1) \tag{3.1}$$

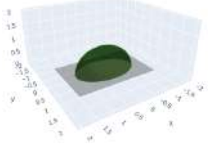


Figure 3.17: Initial volume A_1 based on Equation 3.1

This figure is scaled according to constants (a, b, c). They are determined empirically by measuring the furthest distance measured by the motion capture. This ellipsoid is then transformed to better approximate the required filter.

$$A_2 = \alpha * ((1 - A_1) - \beta) \text{ with } (\alpha, \beta) = (18, 0.3) \tag{3.2}$$

The volume is scaled up by α and shifted down by β. Determining the α and β scaling parameters, leading to the shape in Figure 3.18.

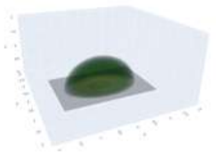


Figure 3.18: Volume A_2 after rescaling with Equation 3.2

$$A_3 = \frac{1}{1 + e^{-A_2}} \tag{3.3}$$

A logistic regression allows for a smooth speed transition at the limits of this volume.

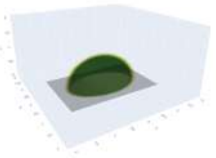


Figure 3.19: Volume A_3 after integrating the logistic regression of Equation 3.3.

Gesture Speed and Angle

Hand movement is separated into angle and speed. As the hand moves in a specific direction on the screen, the components of that vector can be used to calculated the speed and angle of the drone's movement. To help smoothen the output velocity, the mean pixel distance is taken over a rolling window.

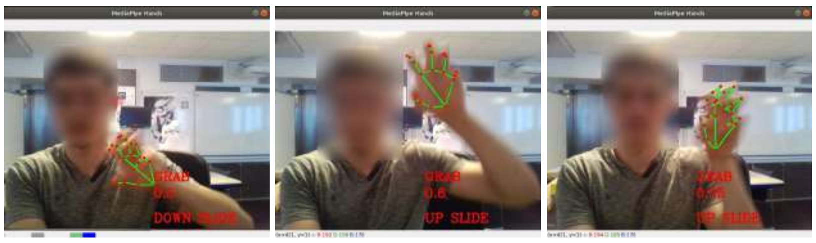


Figure 3.21: Dynamic Hand Signals: Right, Left, Up and Down

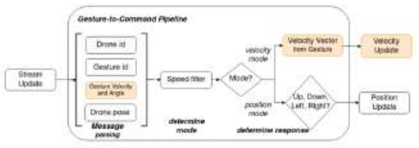


Figure 3.20: Gesture Speed in Message Streaming Workflow

Using these key landmarks, it is possible to discern hand poses and develop a library of drone-piloting hand signals. These are programmed accordingly in the Experimentation Section.

3.3.4 Performance Analysis

Aim

We put in place a demonstration for flight piloting in real-time using the developed gesture interface. We present the workflow of real-time gesture piloting pipeline and we evaluate it in terms of:

- ▶ System response time
- ▶ Accuracy of gesture recognition

Evaluation Techniques

System Response Time
The system response time is verified by applying a series of rapid maneuvers to register any significant delays between the pilot's commands

Figure 3.22: Components involved in experimentation.

and their execution by the flight control system. Similarly, Michał Waliszkiewicz et al. [360] choose to modify the drone's angle in a specified direction. This choice is arbitrary and the changes in velocity are used in this case.The input was a demanded velocity in a specified direction. The input was changed randomly by the operator with hand movements, using the workflow described in this chapter. The output was a delay of the velocity change in the drone. Finally, a system response time is determined by averaging the response delays over the experiment.

Gesture Recognition Effectiveness
In order to evaluate gesture recognition performance, we identify the false positive and negative rates of the pose detection, to compare with existing research in real-time gesture detection. In comparison, Jonathon Bolin et al. [366] evaluates the false positive and negative rates of the pose detection by manually identifying both the incorrectly recognised gestures, and the unrecognised gestures. Similarly, we identify the false positive and negative rates of the pose detection, but instead of doing it manually, we examine any discrepancies in the UAV's trajectory flight. Any inconsistencies in recognition are considered false positives.

Methodology for Piloting Operation

We design our experiments for an operator to guide the drone in an intuitive way through hand commands.

Dataset The experiment was filmed from three angles, and a presentation video is uploaded on Youtube [362]. The results were saved in a rosbag format on Google Drive [399]. The data that is examined extends from 11:19:20 to 11:20:30 on July 29, 2021.

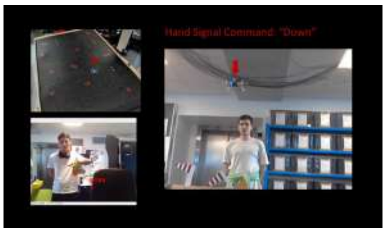


Figure 3.23: Camera recording setup and demonstration of a "DOWN" command

Throughout this procedure, data is collected as a rosbag, a self-contained file for recording ROS nodes and topics. In post-processing, we timestamp the hand signal stream. This file is available at [399] and contains:

- The poses of the drone, ordered by timestamp: /tf topic.
- The hand gesture message contents: /hand_signal topic.

Results

Overview of Results The two flight regions were plotted separately. The trajectory is plotted on 3 planes. The drone's trajectory is first plotted on the X-Z plane (as per Figure 3.24). The two flight regions are separated by locating the transition gesture's timestamp.

Gesture Recognition Effectiveness In preparation for this evaluation, we label the drone positions where each hand signal is detected. Figure 3.27 superposes the drone's trajectory and the hand gestures identified at that particular point on the drone's path.

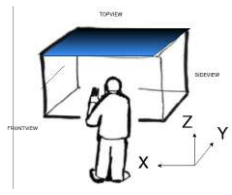


Figure 3.24: The frontview, sideview and topview are taken with respect to this frame of reference.

The detection of hand gestures is found to be mostly continuous. Using the methodology outlined earlier, the false positive and false negative rates can be determined.

Looking at the graphs, it seems that left and down gestures are quite regularly mistaken for one another. In contrast, gestures are different enough (such as right and index) are recognised at 90%.This demonstrates

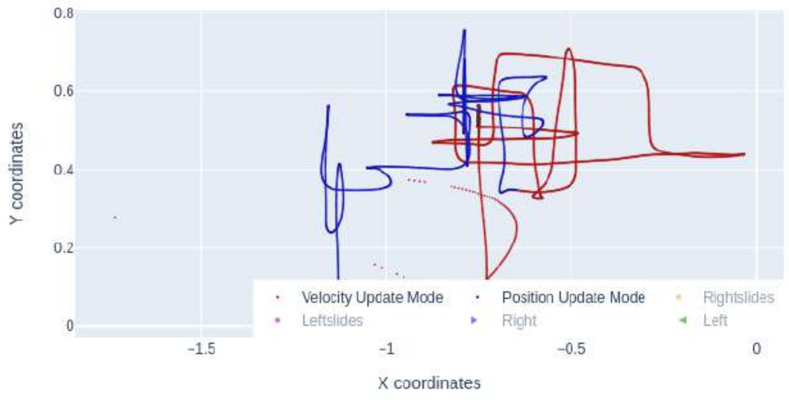


Figure 3.25: Frontview Trajectory Graph with Gesture Piloting.

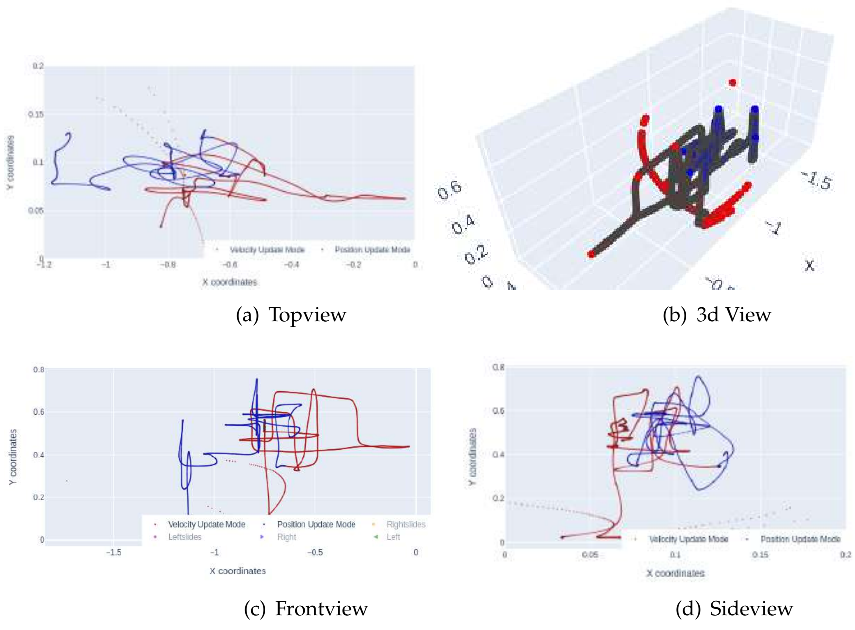


Figure 3.26: Flight Trajectory Graphs with Gesture Piloting.

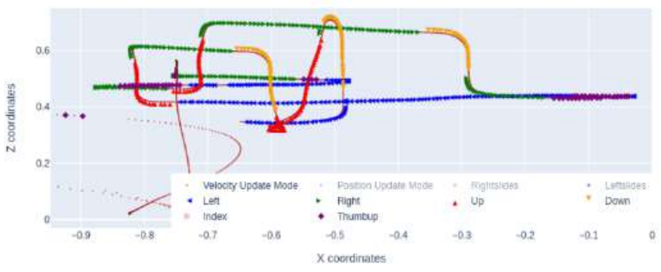
Table 3.3: Gesture Recognition Effectiveness.

Criteria	RIGHT	LEFT	UP	DOWN	THUMBUP
len	342	280	150	87	201
False Positives	5	33	3	6	201
False Negatives	30	175	10	5	0
Accuracy (% correct)	89.7	25.7	93.33	87.35	0

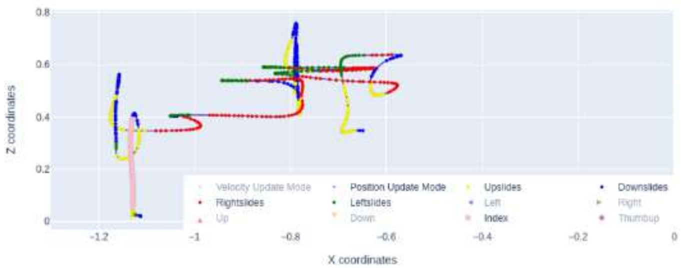
Criteria	INDEX	PEACE	THUMBDOWN	TOTAL
len	192	131	100	1483
False Positives	15	6	100	369
False Negatives	23	36	0	299
Accuracy (% correct)	80.2	67.94	0	56.3

an interesting limitation in the pipeline's design: the recognition seems stumble between two similar gestures.

The effectiveness is averaged as the total percentage of correct gestures



(a) Position Update Mode: Up (red), down (yellow), right (green) and left (blue)



(b) Velocity Mode: Rightslides (red), leftslides(green), upslides (yellow), downslides (blue)

Figure 3.27: Flight Timeline with Annotated Hand Signs.

over the full dataset. The accuracy is determined as 56.3% of the full gesture dataset.

System Response Time In preparation for this evaluation, we to plot the speeds at which the poses are streamed, as well as the desired speed transmitted from the gesture script. The velocities of the actual drone are calculated as per Equation 3.5 from successive pose data points over the period of interest.

$$\Delta_x = x_f - x_i \text{ and similarly for } \Delta_y, \Delta_z \tag{3.4}$$

$$v^* = \frac{\delta u}{\delta t} = \frac{\sqrt{\Delta_x^2 + \Delta_y^2 + \Delta_z^2}}{t_f - t_i} \tag{3.5}$$

This calculation is a simplification given the pose data has a stable 120 ± 0.4 Hz transmission frequency, which is ascertained during the experiment (Figure 3.28).

Figure 3.29 plots the drone's position and its associated velocity in Position Update Mode (**blue**) followed by Velocity Update Mode (**red**).

The red graph is significantly more jaggered. This is expected since the velocity updates depend on fast moving hand movements. In Figure 3.30, we take a closer look at the interactions between a drone's trajectory and the signs identified at that particular instance. The velocity commands (in **green**) are plotted alongside the drone responses (**red**).

The system response time is determined from Figure 3.30 by locating specific spikes of velocity change, in the velocity command stream, and locating spikes of velocity change in the drone flight stream. Their

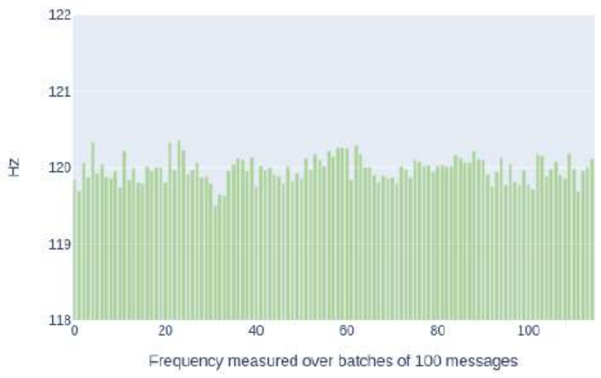


Figure 3.28: Transmission Frequencies throughout Response Time Experiment.

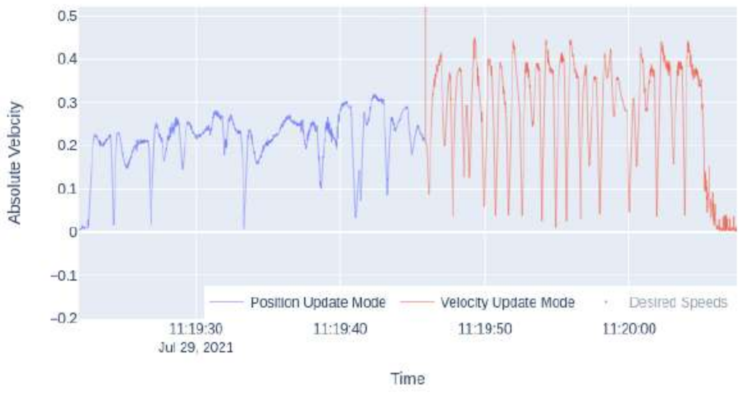


Figure 3.29: Flight Timeline with Gesture Piloting.

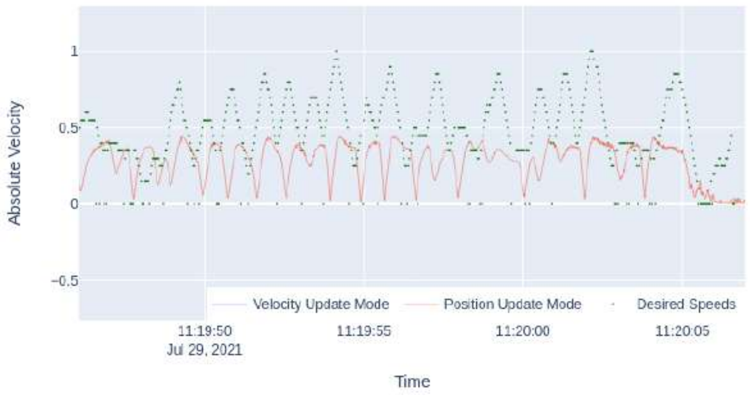


Figure 3.30: Flight Timeline with Velocity Updates.

timestamps are recorded in Table 3.4.

Table 3.4: Latency Calculation from Velocity Stream.

Points	1	2	3	4	5
Velocity Command	19.49.636	19.50.2053	50.4723	51.3419	52.2437
Velocity Response	19.9789	19.568	50.728	51.6482	52.5579
Duration of Latency	342.9	362.7	255.7	306.3	314.2

The average latency is found to be 271.0 ms from the average latency of 10 points. This latency remains rather consistently between 200 and

Points	6	7	8	9	10
Velocity Command	53.753	54.7583	55.3996	56.4005	57.6692
Velocity Response	53.9478	54.898	55.6579	56.6375	57.9679
Duration of Latency	194.8	139.7	258.3	237.0	298.7

300 ms, which demonstrates stability over time. We can perform a cross-validation this result with a visual check: we superpose the graphs by eye to determine an approximate value (Figure 3.31).

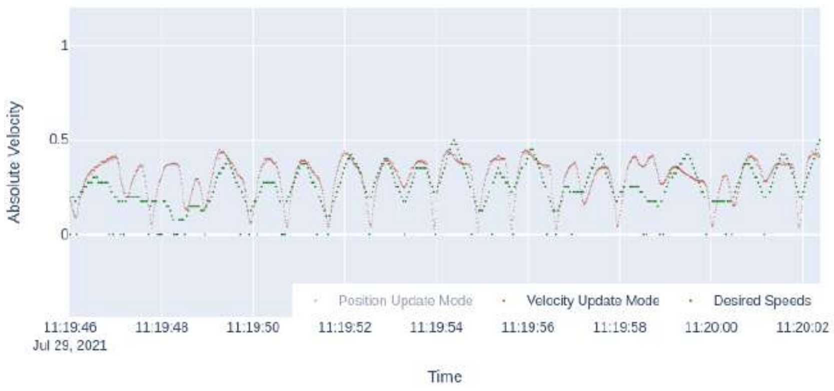


Figure 3.31: Cross-verification by fitting the commanded velocities to the actual velocities.

To create this superposition, we shift the second graph by a difference of 250ms. This agrees with the experimental latency of 200-300ms.

3.3.5 Discussion

While Chang Liu et al. focus on outdoor datasets for single large drones, this work looks towards interacting specifically on the drone's position. Such a specific usecase of hand-following seems to be relatively rare in the literature. In fact, Tezza et al. [357], despite their survey of the research field, remain sceptical as to whether this method might be the best approach to applications that require fine and precise control, as they pose the problems of higher latency and lower accuracy than other methods such as a remote controller. This vision is coerced with other members of the HDI community, and most datasets focus rather on signaling events to the drone, instead of direct piloting (Figure 3.1 from [363]). .

Figure 3.32: Crazyflie in the Flight Arena.

This performance analysis has measured the pipeline latency is evaluated at 270ms. This is in large part thanks to MediaPipe Hands algorithm [395]. This algorithm is relatively new, and demonstrates real-time inference capabilities, with a maximum inference of 36ms for hand landmarks (as shown in Figure 3.5). This performance is evidently far different from that of the perception pipeline developed here around the Mediapipe framework, with different equipment.

Model	Pixel 3	Samsung S20	iPhone11
Light	6.6	5.6	1.1
Full	16.1	11.1	5.3
Heavy	36.9	25.8	7.5

Table 3.5: On-device inference speeds (in ms) for MediaPipe's hand landmark model, adapted from [395]

This experiment has offered a way to approach hand-drone interaction. Other approaches can offer a fuller exploration of the operator's ease in controlling the drone, by examining the frequency at which different hand signals are used. As the operator controls the drone by sight, it is possible for them to make minute readjustments. As a result, further research could examine the role of intuition within this gesture loop. It might also be possible to explore instances where the operator does not look at the drone. Without visual feedback, this could give better hints as to the controller's effectiveness subject to clear hand commands.

Regarding the Thesis

In creating better service drones, one might wonder if a piloting system is an effective means to research and development. It could easily help manage swarms of drones, but is drone development the type of research that requires the operator to make split-second decisions? After all, certain tasks require split-second reactions: drones doing free-fall recovery for instance. Perhaps it could be the beginning of an era of drone real-time learning, where drones can develop functionalities more rapidly than before, through kinetic means. Figure 3.33 is taken from the DJI website, and shows a gesture instruction for a drone to take a picture. Perhaps functionalities like this can become more natural, more closely coupled with human behaviour.

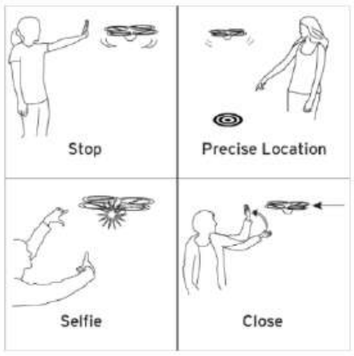


Figure 3.33: Usecase of HDI that have been incorporated in commercial drones [394]

3.3.6 Summary

The gesture interface used to pilot the drones is given 56% accuracy. While the pipeline is based on MediaPipe Hands, the pose classification was hardcoded, and the software can then be improved with a neural classifier or an ML pipeline. In practice, the errors were filtered out by the drone control pipeline.

3.4 Mixed Reality Interface

Drone piloting and other control modalities [357] make use of various inputs to assist in flight. Perception modules for drone flight usually consist of data-driven models based on multiple sensor modalities. These inputs can be sensor modalities, such as camera, lidar, and radar, published in autonomous-driving related datasets, but also human commands, in the case on drone piloting. In this way, perception pipelines are routinely developed as a realtime interface for sensor data from multiple perception configurations.

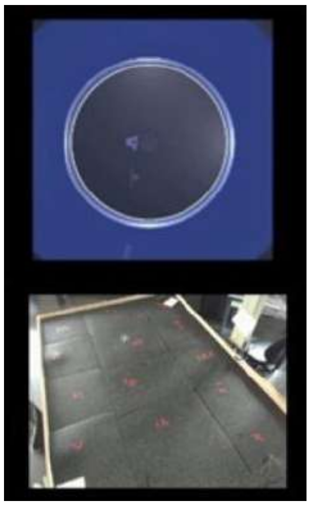


Figure 3.34: Virtual and real environments linked by this interface.

A mixed reality interface serves to enable data transmission between a physical drone and its virtual equivalent. This section documents the design of a mixed reality environment (Figure 3.34). The first objective of the simulated environment is to serve as a graphical interface in order to develop tasks otherwise too difficult to deploy. The priority of the virtual reality is therefore set on rendering capabilities, and the ability to obtain camera streams from this environment.

3.4.1 Selected modules and technologies

This section is a brief mention of all the platforms, systems, services, and processes this interface depends on.

- Unity3D [389] is a popular game engine which offers a simulated environment. It is set up as the virtual companion to the Flight

Arena. Unity is well suited since it enables high-fidelity graphical rendering, including realistic pre-baked or real-time lighting, flexible combinations of different meshes, materials, shaders, and textures for 3D objects, skyboxes for generating realistic ambient lighting in the scene, and camera post-processing [374].

- ROS Sharp [391] is a set of open source software libraries and tools in C# for communicating with ROS from .NET applications, in particular Unity.
- ROS [387] is a set of software libraries and tools that assist in building robot applications.

For the sake of replicability, the version of each module is documented in the references.

3.4.2 Conceptual Overview

The link between the real and the mixed reality is designed with the following core capabilities:

- Transmitting the **pose of a real drone** into a virtual environment.
- Transmitting **an event** between the physical and the virtual environment.

For instance, a drone collision with a virtual object would have the following workflow (Figure 3.35).

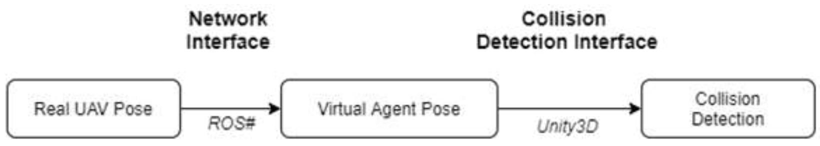


Figure 3.35: Proposed Workflow for Mixed Reality Collisions

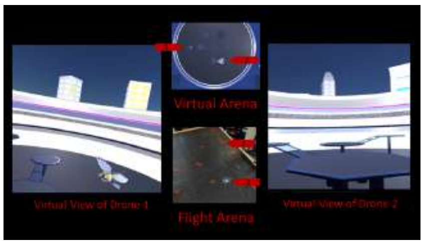


Figure 3.36: Video feeds of the test environment

The process of transmitting the pose of the drone to the simulator is referred to as **pose injection**. This is done via the Network interface, from ROS to the simulator. The process of collision occurs in the simulator, between the injected pose, and a virtual body. This is done via a collision interface within Unity.

These two elements can be readapted to a variety of event-driven scenarios. For this reason, a mixed reality setup offers inexhaustive resources to drone development.

3.4.3 Overview of System Network Interfaces

In order to establish a two-way mixed reality interface, the simulator and the robotics backend are configured to communicate to each other.

To return to the System Network Layout (Chapter 2, Figure 3.37), the Mixed Reality Interface involves Unity3D as well as the Task Manager.

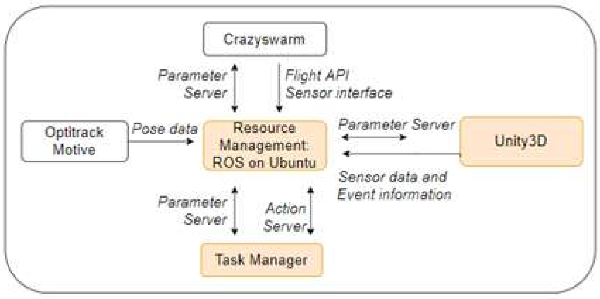


Figure 3.37: Network Interfaces involved in MR flight.

3.4.4 Event Sharing Workflow

A Network Interface for Mixed Reality Event Sharing

A Network Interface is used for the two objectives: injecting pose messages into the game engine and retrieving event data to be sent to the ROS Task Manager.

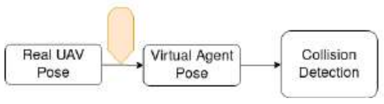


Figure 3.38: Design of the network interface.

Collision Detection in the Virtual Environment

The virtual environment is a Virtual Arena. A particularity of this arena is that it is 3 times larger than the actual flight arena. In other words, the drone's recorded position differs from the real arena by a factor of 3, and the drone's velocity also differs by 3. The two agents are also embodied by virtual characters, annotated as 1 and 2 in the visual below. Video feeds (Figure 3.36) show the perspectives of both agents, and these are recorded as part of the experiment.

An Event Stream Using this Network Interface

An Event Detection is triggered within the game engine when a particular condition is met, and it then publishes the corresponding message.

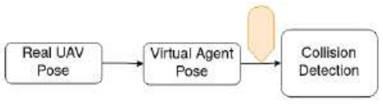


Figure 3.39: Design of the event interface.

The Message Stream communicates the event data using ROS Messages [392]. ROS messages cater to a variety of sensor formats, from cameras, to pointcloud data, allowing for the ROS backend to make further decisions upon processing this data.

Registering an Event in the Robotics backend

The robotics interface, explored in Chapter 2, functions on a Task basis. Events that are streamed on the network therefore need to be connected to processes for task rescheduling as well as drone state changes. Using the Topic Monitor from Section 2.6, changes in a streamed message can be made to induce state changes which, in turn, affects task management processes.

Until the event reaches the task management interface, the real drone is programmed to fly using Chapter 2's high level interface. The full behaviour of the drone can be visualised as in Figure 3.40.

This state machine functions for a single drone: using the swarm building blocks developed in Section 2.6, the real drone to move to certain waypoints indefinitely. When a virtual collision is detected by the robotics

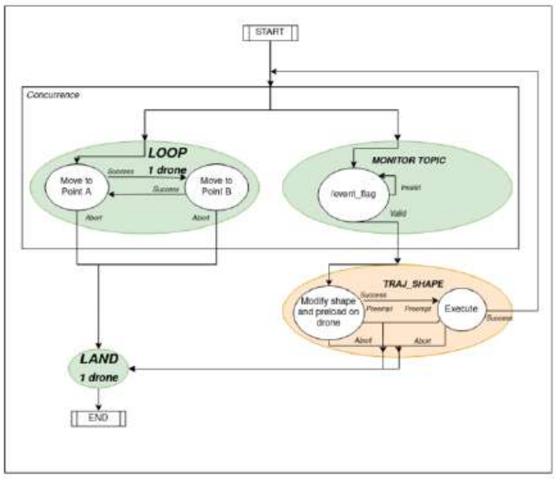


Figure 3.40: State Machine Implementation for Collision Experiment.

backend, it induces a state change. The next state loads a custom trajectory on the drone, which is executed, before returning to an its looping trajectory.

In the next section, an experiment demonstrates the proposed workflow with a collision between a drone and a virtual body, and then to examine the performance of such a system.

3.4.5 Performance Analysis

We set up a virtual interface between real and virtual objects in real time. This MR simulation consists of a network interface between a robotics backend (ROS) and virtual environments (Unity3D). Similarly to [369], the pipeline is then evaluated in terms of communication latency for two separate scenarios.

- when transmitting parameters into the simulated environment
- when transmitting parameters to the robotics backend.

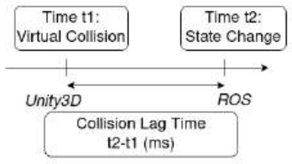


Figure 3.41: Calculation of Pose Transmission Lag

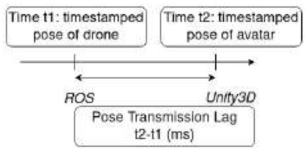


Figure 3.42: Calculation of Collision Lag Time

Prediction

Latency of drone pose into a virtual environment

The latency of the pose injection is measured by determining the time difference between the ROS position and the time when it was received by the simulator.

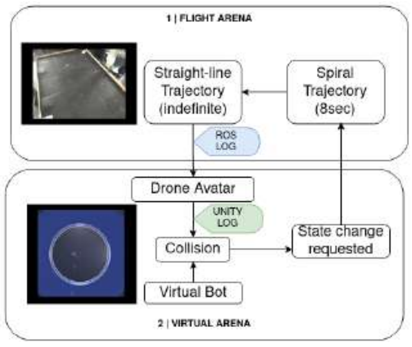
Latency of event from the simulator to ROS

We can answer the performance question by investigating the lag time between the moment of collision and the moment the drone reacts. We choose the moment of a Virtual Collision because it is the ideal moment of a collision between the drone's virtual avatar, and the bot agent. The collision lag time is illustrated in Figure 3.42.

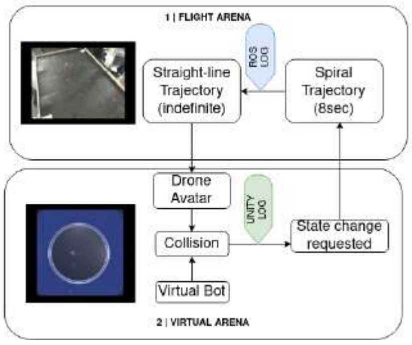
Method

The time of different events is recorded as shown in Figure 3.43. The experiment runs as such:

1. a single drone is flown in the Flight Arena and it is virtualised as the drone agent.
2. Likewise, a virtual bot agent flies a trajectory in a game Engine.
3. When the drone and the bot collide, the drone is designed to react, by flying a pre-programmed spiral trajectory.



(a) Determining the latency of injected pose data



(b) Determining the latency of state changes

Figure 3.43: Data logger setups for both experiments

This requires two separate data loggers: the one, monitoring the Unity environment, logs the timestamp and pose upon the virtual collision, and the other logs the timestamp of the drone State Change from within the Task Manager.

3.4.6 Results

Real-to-virtual Recorded Positions The drone poses are obtained through ROS and the simulator. In Figure 3.44, these posees are superposed. The drone can be seen in green for ROS-times at high-rate sampling (120Hz) and in purple for Unity-times at a lower sampling (10Hz). The positions superpose perfectly. This is expected. This low sampling is sufficient to show the accuracy of the real-to-virtual procedure.

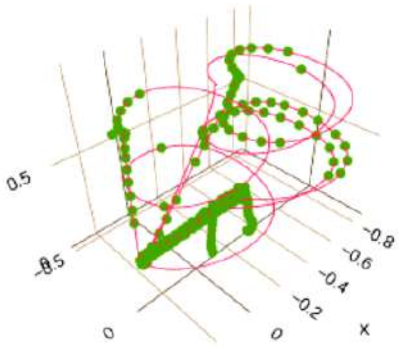


Figure 3.44: Superposition of logs from Unity (green) and ROS (red).

Real-to-virtual Timestamps A second graph examines the differences in timestamps of simulator time (**green**) relative to ROS-time (**red**). Figure 3.46 shows a substantial lag in positions.

The avatar positions occur "before" real positions. This is due to the logs being based on simulation time, which records events slower than real time. This simulator clock seems to be affected by a system latency.

Latency of Pose Injection We determine the latency of elements when injected into the virtual environment. The latency of the system is associated to the time between simulation and ROS time when recording the same drone pose. The resulting graph is plotted in Figure 3.47 and shows a linear trend.

By adapting a linear regression, we determine a gradient of 89t ms of cumulative latency.

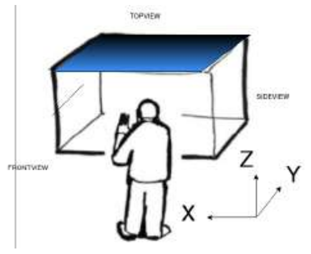


Figure 3.45: The frontview, sideview and topview are taken with respect to this frame of reference.

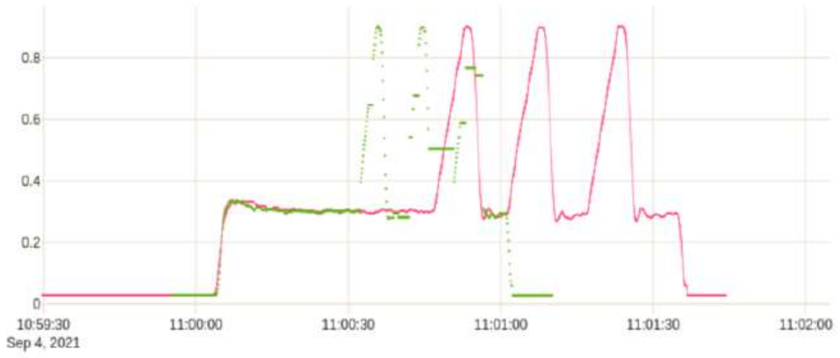


Figure 3.46: Pose Latency Graphs: Z position of Drone over time, for avatar (green) and real drone (red).

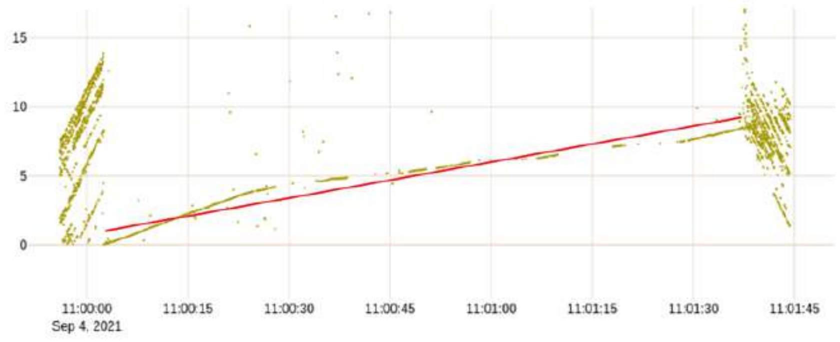


Figure 3.47: Accumulated Latency of System (in Seconds).

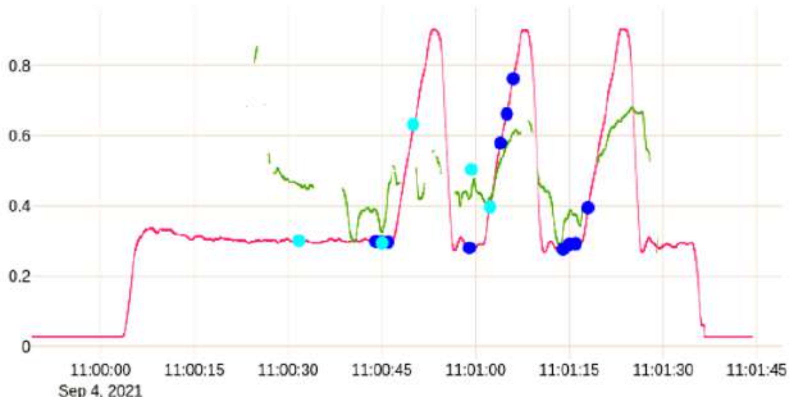
Latency During State Change In order to visualise the event sequence, each collision is assigned a **red** marker, with the moment of robotics backend state change being assigned a **yellow** marker. All the collisions are taken from the virtual logs and assigned ROS-compatible timestamps.

The first plot shows a timeline view, where the moments of start and end of the experiment clearly show a change in Z, on three different occasions. These three collisions are associated to state changes.

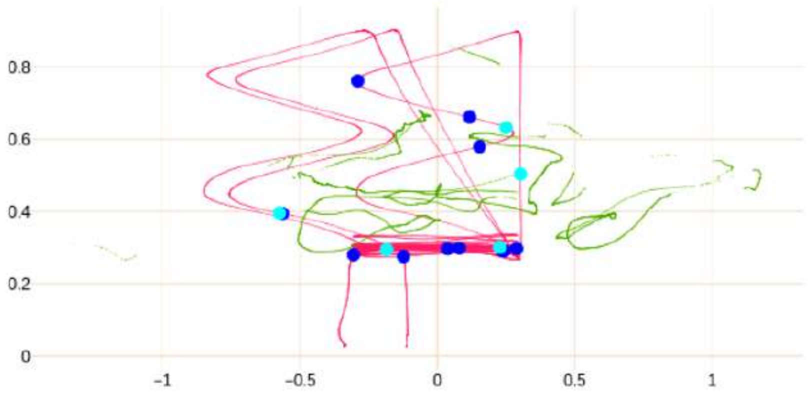
These three collisions in particular are investigated, occurring 15 seconds from each other. Each collision latency is calculated according to the method set in the methodology. Figure 3.51 demonstrates a growing lag time that approximates an exponential - a similar performance bottleneck to the previous section.

·	drone agent
·	virtual bot
●	trajectory execution
●	collision detection

Figure 3.48: Legend for Collision Graphs.



(a) Z position of Drone (green) and Bot (purple) over time



(b) Trajectory along the Side Plane (YZ)

Figure 3.49: Collision Graphs of the Latency Experiment

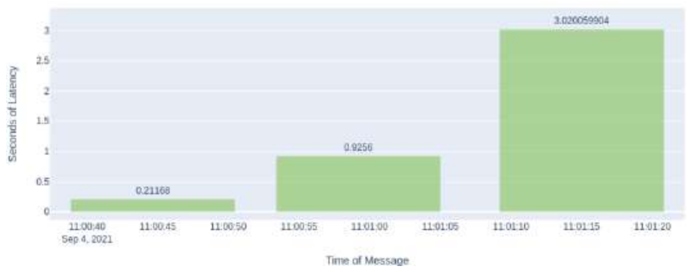


Figure 3.51: Resulting Latency of System with Time.

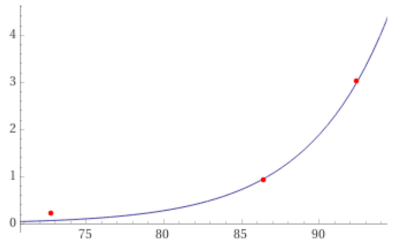


Figure 3.50: Exponential fit for exit latency

This trendline is modelled after an exponential as follows:

$$L_{out} = (6.67589 \times 10^{-8})e^{0.190664t} \text{ ms} \tag{3.6}$$

We can model the full end-to-end system latency as the addition of the $(6.68 \times 10^{-8})e^{0.19066t}$ ms latency and the 98t ms above: $89(7.5 \times 10^{-7}e^{0.19066t} + t)$ ms

$$L_{system} = L_{in} + L_{out} = 89(7.501 \times 10^{-7}e^{0.190664t} + t)\text{ms} \tag{3.7}$$

3.4.7 Evaluation

Similarly to the Swarm Application Interface of Chapter 2, Flightmare offers several tasks as part of their simulator, however they do not undergo tests with real hardware. This discrepancy naturally reflects in the differences in latency, where our system is dependent on a robotics backend on top of a simulator. This chapter can be considered a perception pipeline as opposed to a set of tests that undergo in simulation.

As opposed to simulators with an independent block for the physics engine, this experiment has focused mainly on visualising drone flight. Flight physics modelling is deliberately excluded. This lends itself well to a more photo-realistic, but slower, configuration.

On a **functional** standpoint, the proposed workflow worked as expected. A virtual body did come in collision a number of times with the drone's avatar; through event data, the drone has reacted accordingly. This sequence of events was ensured by the choices of software architectures.

To respond to the **performance** question, we focus on the three key aspects highlighted in the Mixed Reality literature review:

- Fast prototyping of new environments: **programmability**.
- A wide suite of sensors and of physical effects: **variability**.
- A true-to-reality physical environment: the **physical** model.

While the Mixed Reality Interface provides us with a simulated graphics engine, a communication channel was put in place that would communicate virtual events to the robot swarm. However, the collision experiment has demonstrated a cumulative delay of 89t ms for a single quadrotor, and this can only increase with larger swarms and more complex manoeuvres. Since latency is a primary measure for image streaming and high performance drone tasks, we suggest the exploration of a network interface more focused on performance, and possibly the integration of existing simulators like Flightmare within the testbed.

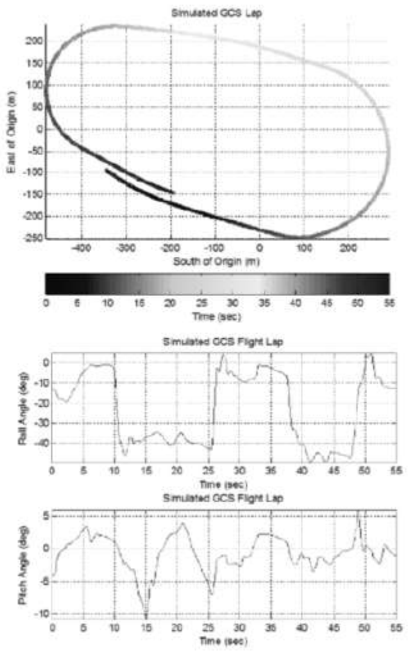


Figure 3.52: UAV poses as measured by the GCS simulator in [359]

Table 3.6: Key findings in Chapter 3.

Test Description	Result
Gesture Piloting	
Gesture Recognition Effectiveness	56.3%
System Response Time	271.0 ms
Mixed Reality Interface	
Latency of Pose Transmission	$(6.68 \times 10^{-8})e^{0.19066t}$ ms
Latency of State Changes	89t ms
System Latency	$89(7.5 \times 10^{-7}e^{0.19066t} + t)$ ms

Summary

According to [359], a fully functional GCS provides for research capabilities which cannot be achieved through R/C flight alone. They emphasize Parameter Identification (PID) research as an alternative for flying a UAV by tracking values of flight and performing precise maneuvers. The second emphasis is into "research into new applications of subscale aircraft for otherwise dangerous or long mundane tasks". These goals echo the elements for live performances and flight recording used in our testbed.

Preiss et al. [376] envision that mixed reality would interconnect a wide variety of physical spaces for collaboration. Humans can work safely within their own physical confines, while their intelligent counterparts can operate in more hazardous environments.

With these new mixed reality tools, Preiss et al. position their robotics testbed as "serving to acclimate end users with autonomous systems". They believe their approach is also well suited for mixed reality prototyping since they "will be able to substitute networking and AI components with alternative implementations", for instance by substituting onboard path finding onto offboard components. They further demonstrate that peer-to-peer networking can better simulate intercommunication between drones. With this, Preiss et al. uphold that mixed reality is a vital addition to better human-drone interfaces.

This chapter demonstrates a similar vision, through the practical means of human-agent interactions. With the data streaming interfaces of this Mixed Reality section, as well as from the Piloting section, this shows that new modalities can be created for autonomous vehicles.

Using the networked approach of events and sensor data, further tasks can be prototyped. This aligns with the goals of better service drones. Using the example tasks developed in the previous chapter, various GNC algorithms can be programmed, developed, and tested on hardware conditions.

This method encompasses a major amount of development on the drone platform. From the development of a custom backend, to the interconnection of a graphical simulator, this completes an ecosystem for research and development. While this work has been in major part, infrastructural, it opens the door to the development and testing of GNC algorithms, for instance Reinforcement Learning algorithms, a common occurrence in recent robotics.

3.5 Conclusion

This chapter presents a streaming architecture for piloting UAVs using a webcam, and various forays into Human-Drone interactions. This architecture, which makes use of the drones' command architecture, but also of a shared network, has lent itself to integrating various inputs – in this case webcam images. The output of this exercise is evident in the precision of the drones' movement, as it was noticed in the various visualisations of the data.

However, While the Mixed Reality Interface provides us with a simulated graphics engine, a communication channel was put in place that would communicate virtual events to the robot swarm. However, the collision experiment has demonstrated a cumulative delay of 89t ms for a single quadrotor, and this can only increase with larger swarms and more complex manoeuvres. Since latency is a primary measure for image streaming and high performance drone tasks, we suggest the exploration of a network interface more focused on performance, and possibly the integration of existing simulators like Flightmare within the testbed.

Using the networked approach of events and sensor data, further tasks can be prototyped. Various GNC algorithms can be programmed, developed, and tested on hardware conditions.

All in all, this work has come to demonstrate that a swarm setup can be rather easily adapted to human-drone research, and despite the performance bottlenecks, it succeeds in providing an end-to-end experience for the pilot, and further work should aid in streamlining this.

In Vivo Deployment for Industrial Environments 4

4.1 Introduction

UAVs provide us with a novel way to capture data; they help us to gain a perspective of the Earth that is simply not possible with instruments that are based on the ground. In contrast with the platforms of manned aircraft and satellite, the UAV platform holds many promising characteristics [339]: flexibility, efficiency, high-spatial/temporal resolution, low cost, easy operation, etc., which make it an effective complement to other remote sensing platforms and a cost-effective means for remote sensing. As a result, low-cost, light-weight UAVs have been used extensively within research for more than a decade to measure, for instance, air quality, mapping of 3D geodata and remote sensing within agriculture [342]. However, until recently, UAVs were build with a single purpose and with one particular sensor on board.

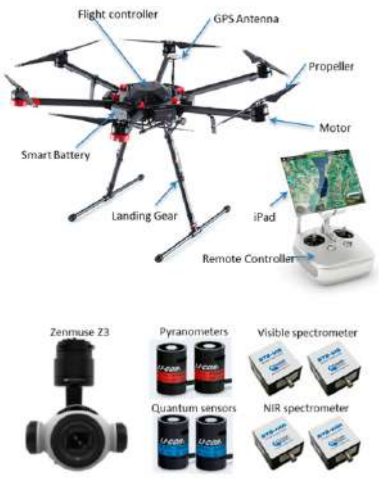


Figure 4.1: example of a remote-sensing vehicle: The DJI's Matrice 600 for Measuring Land Surface Albedo [400]

From about 2010, the stability, flight duration, and load capacity of UAVs increased significantly with the development of flight-control and battery technology, which enable more sensor varieties (optical sensor, lidar or radar) to be mounted on smaller UAVs [351]. Figure 4.1 alongside demonstrates this: a multi-rotor based UAV platform was developed and tested for measuring land surface albedo and spectral measurements at user-defined spatial, temporal, and spectral resolutions. This drone onboards an RGB camera and a set of four downward pointing radiation sensors.

In line with the thesis goal, we pay attention to the interplay between smart systems, and the real-world deployment of a UAV. The carrier drone aids in streamlining the data collection process, by automating different fly-by procedures, safeguards, and scheduling the data collection. Two payloads are tested in outdoor flight, for atmospheric data and vibration data, and the sensors used in these tests are evaluated. In this way, onboard systems can aid with the practitioner's task.

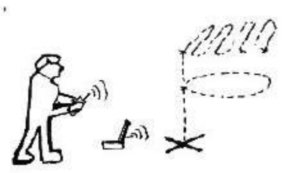


Figure 4.2: An environment for outdoor deployments.

4.2 Related Work

In the literature, there is increasing reference made to the potential of using UAVs as autonomous or semi-autonomous operating data acquisition platforms, often referred to as Mobile Sensing Platforms (MSPs) [341] [342]. They can be equipped with state-of-the-art measurement instruments offering extremely high resolution and are ideally suited to access otherwise prohibitingly inaccessible locations or to operate in hostile environments that would be lethal to the human operator.

The field scope is restricted to mobile sensing upon UAVs. Furthermore, the focus is in data inspection and data gathering in outdoor inspections as opposed to inspecting underground and pipeline plants, as well as urban spaces, as opposed to powerplants, solar farms, wind farms, or precision agriculture. As a result, we have restricted our investigations to two sectors of research: remote sensing research and structural inspections.

4.2.1 Remote Sensing Research

We see great growth in mobile mapping research, that is "the acquisition of spatiotemporal phenomena by using a mobile multi-sensor platform" [343]. The UAV is a platform that greatly simplifies research. One such field remote sensing, the acquisition of information about an object or phenomenon without making physical contact with the object [344]. Recently, UAVs have enabled research towards contact sensing, for the acquisition of information in direct contact with the object [345], on a platform that serves for optimal sensor placement [345] [358].

In contrast with other aerial platforms like manned aircraft and satellite, the UAV platform holds many promising characteristics [339]: flexibility, efficiency, high-spatial/temporal resolution, low cost, easy operation, etc., which make it an effective complement to other remote sensing platforms and a cost-effective means for remote sensing. As a result, low-cost, light-weight UAVs have been used extensively within research for more than a decade to measure, for instance, air quality, mapping of 3D geodata and remote sensing within agriculture [342]. However, until recently, UAVs were build with a single purpose and with one particular sensor on board.

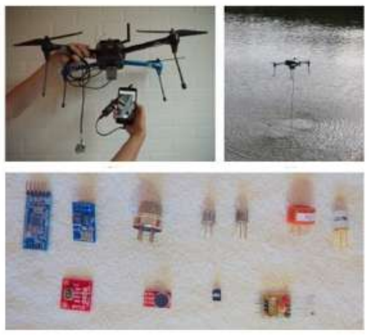


Figure 4.3: A low-cost, open-source, modular sensor platform [341].

[341] *Lars Yndal Sorensen, Lars Toft Jacobsen, and John Paulin Hansen* propose a multi-sensor platform based on commercial off-the-shelf components that provides a modularized sensor system and data acquisition infrastructure. This "Mobile Sensor Platform" (Figure 4.3) is a commercial quadcopter that allows for attaching external sensors and relaying the data back to a ground station using a telemetry communications link. The platform supports a simple and expandable interface for attaching custom sensors to the UAV, overcoming the limitation of single-purpose platforms which are costly to convert for other tasks.

Unmanned aerial vehicles have been shown to be useful for the installation of wireless sensor networks. Wireless sensor network (WSN) technology refers to a group of sensors used for monitoring and recording the physical conditions of the environment and organizing the collected data at a central location [342]. Sensor nodes can be placed more accurately with the development of autonomous aerial robots capable of physically interacting with the environment.

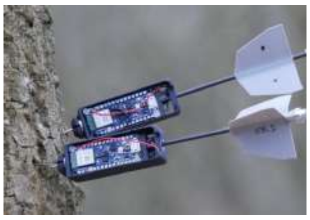


Figure 4.4: Impulsive launching of sensors to monitor the health of forests [345]

[345] *Andre Farinha et al.* present a novel method for aerial sensor placement in hazardous environments. This method is applied to Wireless Sensor Network deployment. The proposed system does not require direct physical interaction to accurately place sensors which brings significant advantages in cluttered environments and in overall operational safety.

4.2.2 Structural Inspections with UAVs

[347] *Shi Zhou and Masoud Gheisari* explore peer-reviewed bibliographic databases and offer an interesting picture of the last 10 years. A growing number of publications are identified, and the publications

are classified by Shi Zhou and Masoud Gheisari into five main topic categories: building inspection, damage assessment, site surveying and mapping, safety inspection and progress monitoring.

Drones are a disruptive technology regarding the structural assessment for reinforced concrete bridges [401]. Inspectors used to employ ladders, scaffolding, or lifters to reach the parts of beams and piers of the bridge that are not easily accessible. In contrast to inspections done by snooper trucks(Figure 4.5), the inspection detail that UAS provide effectively replicates some of the detail learned through the use of snoopers , without the traffic control requirements, and at significantly lower cost in terms of equipment and traffic control needs [401]. UAS can provide both infrared and 3D modeling detail of bridges, effectively identify concrete delamination, cracks, rebar corrosion, topographic mapping detail and other visual inspection.

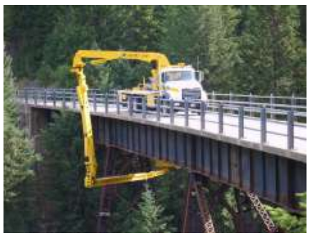


Figure 4.5: Bridge inspections with a snooper truck [401] (2016).

[402] *Kun Feng, Miguel Casero, and Arturo Gonzalez* affect direct contact of the sensor or device with the bridge surfaces. Structural inspections of the internal state of a bridge require the measurements of beam deflections with direct contact. An operator manually places a reflector prism attached to a pole in contact with different points at the beam and then measures the position of the prism with a laser total station.

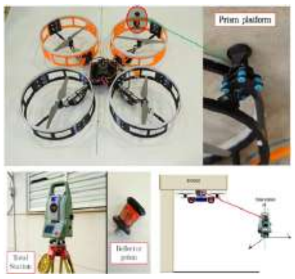


Figure 4.6: Bridge inspections, with a Laser Prism. [402]

[358] *Robert Stewart et al.* design and test an unmanned aerial vehicle with seismic sensing capabilities. The seismic sensing platform consists of four 100 Hz geophones and recording electronics. This is attached to a 3DR Solo Quadcopter drone. The geophone spikes become the drone's landing legs. The drone and its geophone payload have been successfully flown a number of times with take-off, programmed or remotely controlled navigation, landing, and recording. We have conducted tests (using hammer and weight drop sources) to compare the response of the landed seismic-drone system to planted geophones and a conventional cabled seismic system. The seismic traces from the drone are assessed as similar to those of the planted geophones.

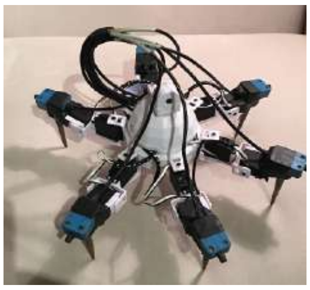


Figure 4.7: 100 Hz geophones plus recording electronics attached on a 3DR Solo Quadcopter drone. [358]

4.3 Carrier Drone Design

A carrier drone is developed to simplifying the integration of new sensors in preparation for outdoor operation. An unmanned aerial vehicle system has two parts, the drone itself and the control system. The design elements examined here are the following:

- **Drone Components**: onboard sensors and navigational systems for stable, semi-autonomous flight.
- **Control system**: Flight system features like piloting modes and datalinks that are specific to the types of operation.
- **Navigation parameters**: safeguards and automated parameters to provide safe and secure flight.

This sections examines the flight-related functionality, including piloting features, navigation safeguards and other automation tools. Sections 4.4 and 4.5 document the design of two modular payloads for collecting Atmospheric Data and Vibration Data.

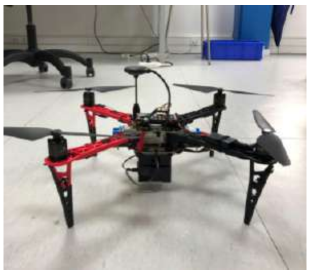


Figure 4.8: Final Drone Setup

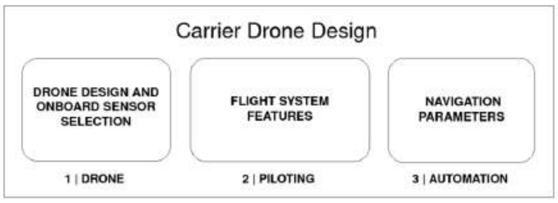


Figure 4.9: Carrier Drone Design and Flight Environment.

4.3.1 Onboard Components

The body of an unmanned aerial vehicle is designed for propulsion, via electrical motors and their associated navigation system. This is where all the sensors and navigational systems are present.

Drone Frame and Power Distribution

The UAV chosen for this system is the DJI's F450 Flame Wheel frame, a self-assembled platform designed for aerial photography and entertainment purposes. This quadrotor is common in research as it provides the space required to assemble new modules [341] [357].

The F450 is comparatively larger than many other drones; however, it is built of light-weight stiff carbon fibre and the battery and GPS can be stored away. It has an approximately 40 minute hover time without a payload and 20 minutes with maximum payload (3kg).

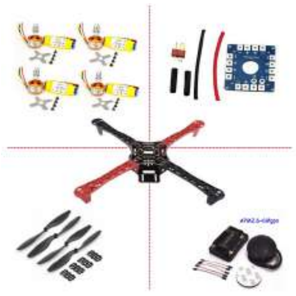


Figure 4.10: The F450 drone, along with (clockwise from top) a Power Distribution Board, a GPS setup, propellers and motor-ESC pairs.

Flight Autopilot Controller

The Flight Controller has inputs both from the Ground Station and the RC Controller. It acts upon the UAV Motors, logs data from the sensors and interacts with various payloads. Pixhawk [403] is an industry standard autopilot developed and jointly developed by 3DR Robotics and Ardupilot Group. Various robots such as RC cars, airplanes, and multicopters can be made, and firmware is provided for them using Pixhawk. Motor control is achieved automatically, with enhanced stability to recover from wind surges.

Figure 4.11: The Pixhawk4 and GPS: an advanced autopilot adopted by academic and commercial developers [403].

Onboard Sensor Selection

The carrier drone aims to simplify the integration of new sensors and associated data acquisition systems. To do this, a modular approach is adopted, whereas the setup is designed to simplify the board reconfiguration in new flights.

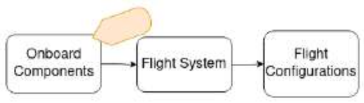


Figure 4.12: Carrier Design, step 1.

Custom criteria are used to better understand the level of modularity for several sensor systems.

- **Large Field of View**: Whether said sensor requires a large field of view for optimal functioning,
- **Vibration Sensitivity**: the vibration sensitivity of equipment in the case of parasitic vibrations. This element is desirable when carrying vibration-sensitive loads, for instance lidars, or items that might dismantle with vibrations.

- **Independent Power Source**: an independent power source in the case of self-powered electronic circuits.

Table 4.1 looks at four possible sensor systems.

Table 4.1: Payload Selection Matrix.

Criteria	Vibration Data	Atmospheric Data	Rangefinder Data	Camera Stream
Independent Power Source	●	●	○	○
Vibration Sensitivity	●	○	●	●
Large Field of View	○	○	●	●
Selection	✓	✓	✗	✗

With a fully independent power source and less importance on the field of view, two sources of data are examined in more depth: the Vibration Data Subsystem and the Atmospheric Data Subsystem. The two DAQ systems are designed as independent subsystems.

DAQ Onboard Setup

A payload box can be used to encapsulate these subsystems, such that each module can be installed and removed during on-site drone reconfiguration. In the next section, a rectangular box is designed and 3d printed to attach upon the UAV. This box is designed for a variety of payloads. It is screwed on and can be removed easily, making it highly modular.

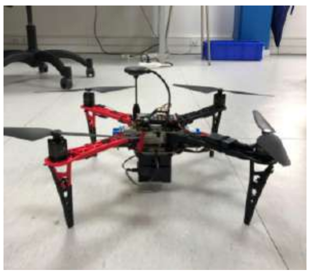


Figure 4.13: DAQ Installation

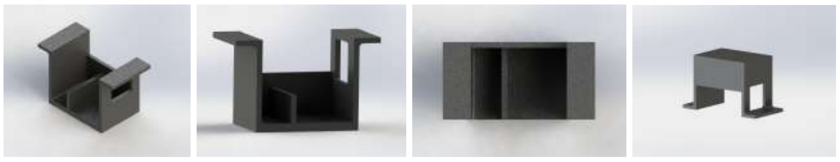


Figure 4.14: Renders of payload box.

The payload box offers an open window for cabling, as well as interstitial walls for the battery and the slice module. Using this design, the DTS Slice is installed on the UAV, with a Battery as an energy source. The end result fits neatly on the drone (Figure 4.13).

4.3.2 Flight System

The software controlling the UAV is a combination of elements. The flight system consists of a drone and all its associated components, for navigation, for the mounting of sensors, and for the logging the collected data. Figure 4.8 presents the final drone configuration.

The flight system consists of a drone and all its associated components, for navigation, for the mounting of sensors, and for the logging the collected data. Figure 4.8 presents the final drone configuration.

The data acquisition systems seen here (DTS Slice and Arduino Nano) are expanded upon in Sections 4.4 and 4.5.

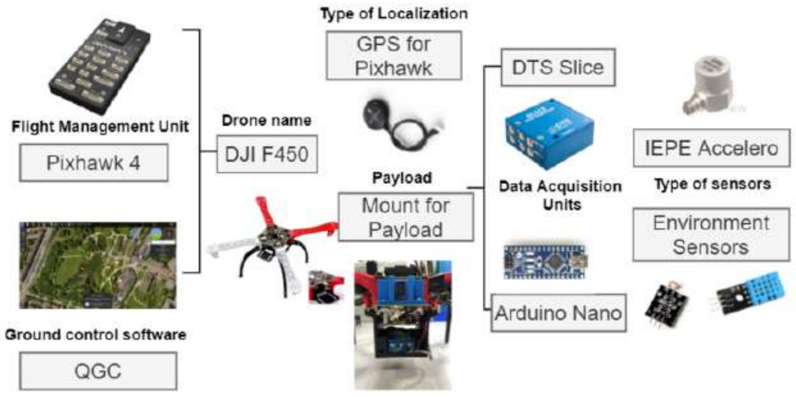


Figure 4.15: Expanded View of the UAV Ecosystem

UAV Flight Stack

We chose to use the open-source PX4 flight stack [403] on the Pixhawk, running on top of the Nuttx real-time operating system. It has a number of mission critical features that are favorable for our research requirements. PX4 has a large research-oriented community of open-source practitioners, and tools for programming custom applications.

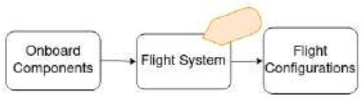


Figure 4.16: Carrier Design, step 2.

Remote Controller and Datalink

The Taranis is a common drone flight RC controller [405]. It features 24 channels with a rapid baud rate and low latency, thanks to its high-speed module digital interface. The Q X7 transmits in the 2.4GHz range provides a secure and reliable link.

The OpenTX firmware [405] onboard the Remote Controller offers the possibility of assigning various switches to different flight modes, such as an arming mode in Figure 4.9. Using two switches, we alternate between modes in the following order:

Figure 4.17: The Taranis X7 Remote Control: a reprogrammable remote for RC flight [404].

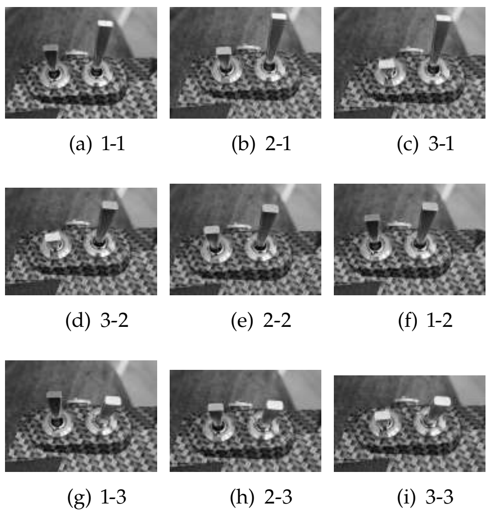


Figure 4.18: Switch combinations and order of transitions.

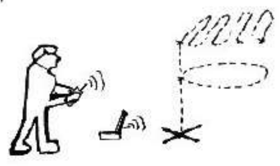


Figure 4.19: An environment for outdoor deployments.

Using this technique, there are 9 modes that can be assigned a functionality. In this way, the pilot is given access to automatic functions. These assignments are done in Section 4.3.3.

Ground Control Station and Telemetry

Ground Control Stations (GCS) are sets of ground-based hardware and software that allow UAV operators to communicate with and control a drone and its payloads [351]. The QGroundControl ground control station [403] is used to communicate with the Pixhawk Autopilot via a Telemetry Radio. The communication is ensured by a Serial UART connection (MAVLink protocol [403]).

Figure 4.20: QGroundControl: an in-flight communication station [403].

4.3.3 Configurations for Flight Operation

Through the software outlined above, the drone can be remotely controlled or fly autonomously. Further mode configurations assist to program the drone for specific tasks. A flight setup is adopted here for the experiments in the next chapters, that is advantageous in terms of ease-of-use, programmability and safety.

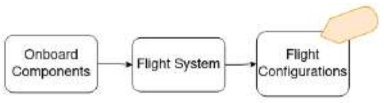


Figure 4.21: Carrier Design, step 3.

Pilot Configuration

The switch is used to coordinate multiple flight modes. These modes are useful to alternate between very controlled motion, and smoother motion for data acquisition.

These three modes are configured to switches 1-1, 1-2 and 1-3. In a single flight operation, Position mode is used to position the drone at a hover, Altitude Mode is used to move on a horizontal plane, and Stabilized mode is used to move freely in 3D space.

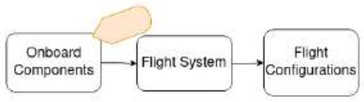
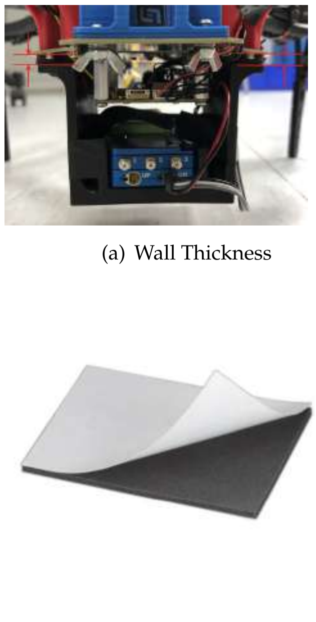


Figure 4.22: Carrier Design, step 1.

4.3.4 Damping Optimisation Study

The last criterion of the design process is the safety of payloads onboard the drone during flight. For this reason, the payload is required to be vibration-sensitive.

The goal of this section is to minimize the vibrations experienced by the payload during flight. A damping test is carried out to determine the amount of damping material on the payload box. This procedure forms part of the drone design process determine an optimal damping volume.


(a) Wall Thickness

(b) AET vibration damping material

Figure 4.23: Installation of Damping Material

Hypothesis

More shock absorption material leads to a smoother frequency response.

Prediction

Two payloads A and B are attached to the drone. They vary by the amount of damping material (see Figure 4.23). The hypothesis is validated if payload A has better damping sensibility than payload B. The damping sensibility is associated to the Frequency Response of each payload.

Methodology

Measurement Equipment Setup Our approach is a comparison of vibrations experienced by the drone payload during flight, compared to the vibrations on the drone frame. We do so by gathering acceleration data. For each flight, two accelerometers are used: a reference accelerometer on top of the drone serves as the control, as well as a payload accelerometer installed within the payload box (Figure 4.24).

- Payload A has 8 mm high absorbers.
- Payload B has 4 mm high absorbers.

AET material is used as shock absorbers in the payload attachment (Figure 4.23). It is fitted between the payload and the drone, and screwed tight. This is the only point of contact.

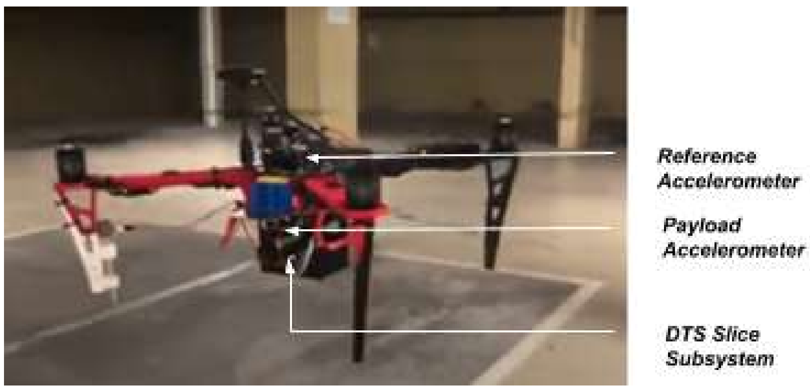


Figure 4.24: View of the two accelerometers in Experiment setup

Acquisition Parameters The DTS Slice is set up according to a few parameters. DAQ parameters are listed in Table 4.2. The DAQ was sampled at 25kHz. The anti-aliasing filter (AA), output scale factor and sampling rate are defined, among other factors.

Test Description	
Test Date	23/08/2021
Test Time	11:30:49
Sample Rate	25000 Hz
Hardware AA Filter	5000
Data Channels	2
Accelerometer Type	3055B2T
Nb of Post-Zero Data Pts	750002

Table 4.2: Parameters of the Payload Damping Test.

Experiment Procedure This experiment has two phases: in the first, Shock absorbers A are fitted; in the second, Shock absorbers B are fitted. This test proceeds in the following manner:

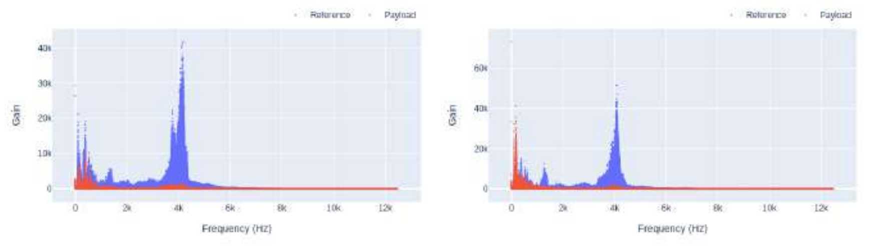
1. The drone is fitted with Payload A: a payload with a **greater** volume of shock absorber. See Figure 4.24 as a reference.
2. The drone is flown for 1 minute: forward, backward, to the left, and to the right. It is then landed.
3. The drone is fitted with Payload B: a payload with a **smaller** volume of shock absorber.
4. The experiment repeats.

Post-processing For each test, we calculate the PSD functions using the Welsh method, and calculate a ratio of the payload accelerometer's PSD over the reference accelerometer's PSD. This produces the Frequency Response Function.

Dataset The procedure was carried out on 23 August 2021 in an indoor parking space. The Google Drive experiment dataset [406] and a Youtube presentation video [407] are publicly available.

Results

PSD Plots For each of the two payloads, we calculate the PSD functions using the Welsh method (Figure 4.25).



(a) PSD Graph for Payload A and Reference (b) PSD Graph for Payload B and Reference

Figure 4.25: Development of the Gain Graphs to compare Payload A and B

Frequency Response Function We calculate a ratio of the payload accelerometer's PSD over the reference accelerometer's PSD. This produces the Frequency Response Function (Figure 4.26).

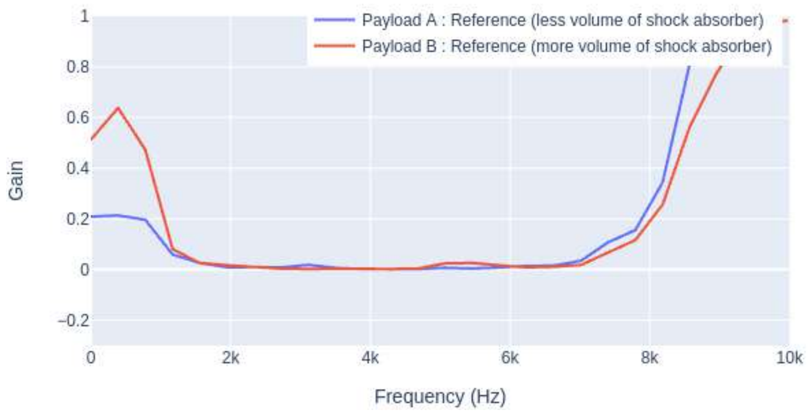


Figure 4.26: Gain difference between the two payloads

The Frequency Response graph spikes notably in frequencies above 8000 Hz. These appear to be unstable behaviour at the extremes. However, when examining this in relation to the FFT-graph about 8kHz, we see that the input and output magnitudes are very small values, and their ratio evidently produces this unstable behaviour. These are not in the range of interest and can be neglected.

The initial hypothesis assumed that more damping material would be associated to better damping. This result suggests that the volume of material may help at higher frequencies, but this has no apparent benefit in the 1-2kHz range of interest.

Conclusion

Through the means of a payload damping test, the hypothesis is rejected, insofar as more shock absorption material did not give a lower gain in the frequency response. Since the objective of this test is about maximising the damping in the range of interest, Payload A is installed onto the drone. We suggest that that this test be carried out over a wider range of materials, and for other types of flight, in order to determine an optimal damping volume for different drone designs.

4.4 Semi-autonomous scan of a zone

Environmental sensing usually requires substantial time for data collection or more distributed sensing systems. The use of atmospheric sensors is a major element in remote sensing [342]. With a platform that can carry multiple types of sensors, a simple field scan can help understand the limitations of a drone task, and the potential for further operations. As an air monitoring solution, this demonstration can be extended to industrial-type solutions, such as air pollution monitoring [341]. All these factors show that drones form part of a trend towards service automation for industrial purposes.

Figure 4.27: Factories besides a body of water. Many organisations aim to use drones to scan air pollution in the surrounding areas.

We do remote sensing upon a UAV to simplify the task of environment sensing. A drone is equipped with remote sensors. It is flown in a field where it detects physical changes in the environment: lighting, humidity, and temperature. This data is then processed to determine the effectiveness of the survey. A presentation video is available (Google Drive) [371]. The experiment data is available on Google Drive [370].

4.4.1 DAQ System Design

This section documents the design of the Atmosphere Data Payload selected in Section 4.3.1. It is composed of three stages.

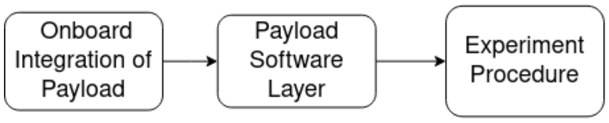


Figure 4.28: Preparation of Atmospheric Data Collection.

The payload is installed on the drone in Step 1. Flight procedures are designed in Step 2. The DAQ Activation is done in Step 3.

System for Low-Cost Sensors

The Arduino sensor range is chosen for prototyping for its low-cost sensors. Using an Arduino Nano [408], such sensors can be easily integrated. The Arduino Nano is sold as a small, complete, and breadboard-friendly board based on Arduino's larger counterpart, the ATmega328. It only weighs 7g with minimal volume.

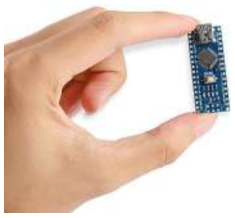


Figure 4.29: The Arduino Nano: a small, breadboard-friendly, rapid prototyping board [408].

Module Design

Two separate modules are designed for the Atmospheric Data and the Vibration Data. In each, an independent battery powers the logging unit, and the sensors attached to it. The Pixhawk board is included in the first

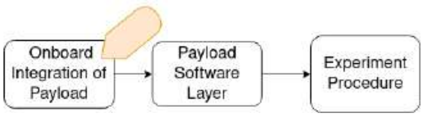
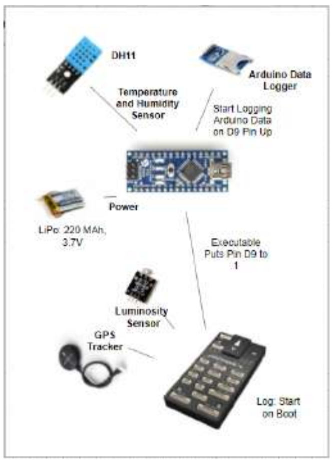
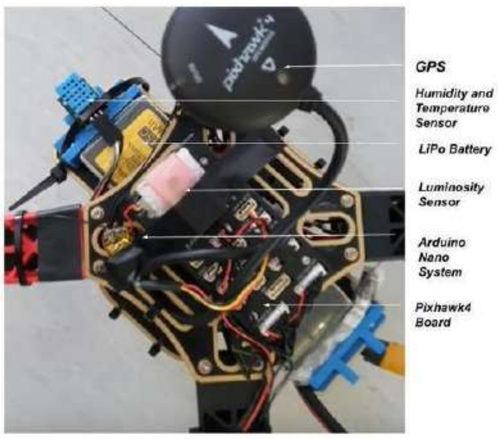


Figure 4.30: Setup Step 1.

system since some the GPS and luminosity sensor are logged via the Pixhawk board.


DH11
Arduino Data Logger
Temperature and Humidity Sensor
Start Logging Arduino Data on D9 Pin Up
Power
LiPo: 220 MAh, 3.7V
Executable Puts Pin D9 to 1
Luminosity Sensor
GPS Tracker
Log: Start on Boot
(a) Arduino DAQ Components



(b) DAQ Integration on the UAV

Figure 4.31: Arduino DAQ setup for field scan

DAQ Control Layer

The DAQ is configured to activate and deactivate the datalogging process on command. To achieve this, we use custom activation firmware on the PX4 operating system.

Table 4.3: DAQ Activation Procedure.

Boards	Switch	Activation	Deactivation	Prior to Activation
2. Arduino Nano	2-2	On boot	On shutdown	None
3. Pixhawk	2-1	On boot	On shutdown	None

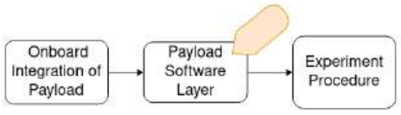


Figure 4.32: Setup Step 2.

The data-logging activation file was coded in C++, and compiled into an executable via the MAVLink protocol. During operation, it toggles a pin (Pixhawk's FMU Channel 6), which is then detected by the Arduino Nano in order to begin and end the logging on the Datalogger.

A separate custom logger detects Arduino activation and records its timestamp in the PX4 debug log.

4.4.2 Sensor System Evaluation

Aim

Using three separate atmospheric variables, we determine the accuracy of the drone sensing solution.

Prediction

Sunlit and shaded regions were scanned for relative humidity, luminosity and ambient temperature. The drone's flightpath is changed randomly by the operator to create region overlaps. The trajectory plots demonstrate any inconsistencies in the readings. We determine the maximal variation per second and per meter as a measure of the fluctuations in lighting and in temperature.

Method

Measurement Equipment Setup Instruments that were used in the system are pictured in Figure 4.31. These include the DH11 Temperature and Humidity sensor, as well as a 5mm LDR Luminosity Sensor, and finally, the companion GPS. A particularity is that the DH11 is attached to the Arduino Board, and Logged with the use of an Arduino Data Logger, while the 5mm LDR is connected to an ADC input on the Pixhawk board, containing a self-enclosed data logger.

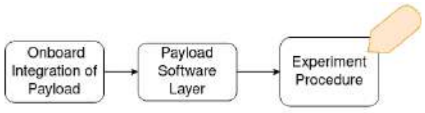


Figure 4.33: Setup Step 3.

The Pixhawk logger supports 100Hz data logging while the Arduino data logger averages at 10Hz logging. Both data loggers support Micro SD cards with a capacity of up to 64GB to store high-resolution video data, photos and flight telemetry.

Experiment Procedure The flight takes place in an empty field of approximately 100x60m, identified for the differences in lighting between the tree shade and the sunlit field. The drone is piloted by hand. This requires a certain method:

1. System checks (battery monitor, screws, etc.)
2. Activating the drone.
3. Drone takeoff and moving to an altitude of 2m.
4. Altitude lock.
5. Activating the Arduino data acquisition with a PX4 trigger application.
6. Piloted flight across the field, along sunlit and shaded regions.

Data collection The data was collected on 24 August 2021, over an empty field of approximately 100x60m. Both the lighting and the GPS data are taken from the Pixhawk Log. They were both sampled at a frequency of 98 Hz. The Arduino Logger was activated 248s after the Pixhawk Logger. The Arduino Logger was active for a duration of 712 seconds, of which 464s are common to both boards. Both the temperature and the humidity are taken from the Arduino Datalogger. They were both sampled at a frequency of 11.7 Hz.

Figure 4.34: Presentation video [371]

Dataset A presentation video is available (Google Drive) [371]. The experiment data is available on Google Drive [370].

Results

Timeline of Environment Sensor Readings There is a stark contrast between sunny and shady regions in the data.

This is a change of 20% of the luminosity range, where sunny regions saturate the sensor, and shady regions are marked by sudden drops.

Trajectory Plot Figure 4.36 plots the lighting readings over the trajectory.

This confirms that the data is very sensitive to lighting differences. The darker patches in the sunlit area might be explained by the passage of

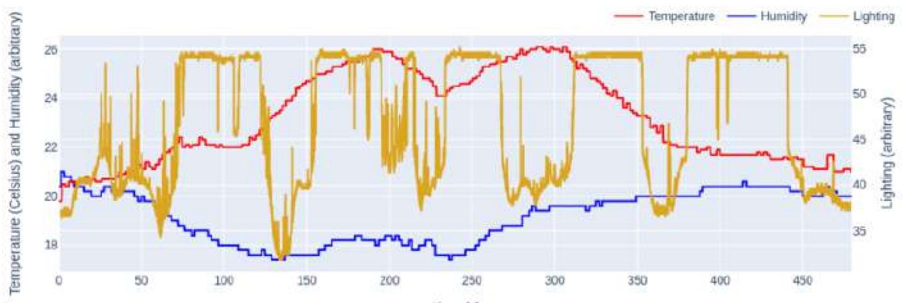
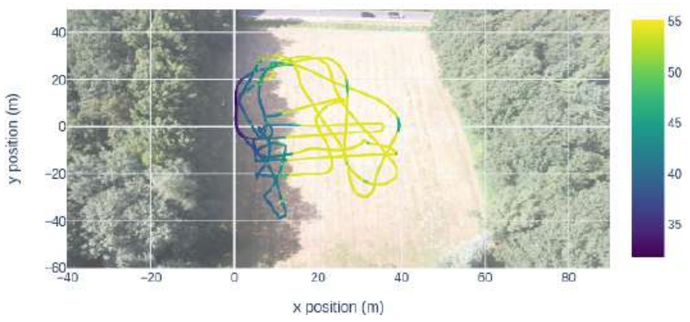
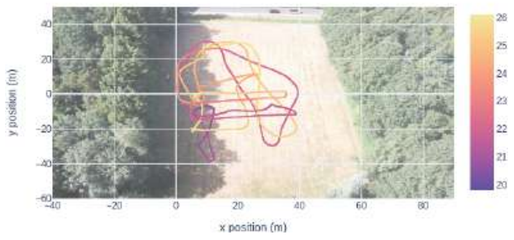


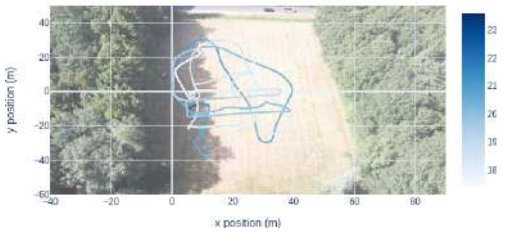
Figure 4.35: Temperature, Humidity and Lighting Recordings during Flight



(a) Trajectory with Lighting as a Colour



(b) Trajectory with Temperature as a Colour



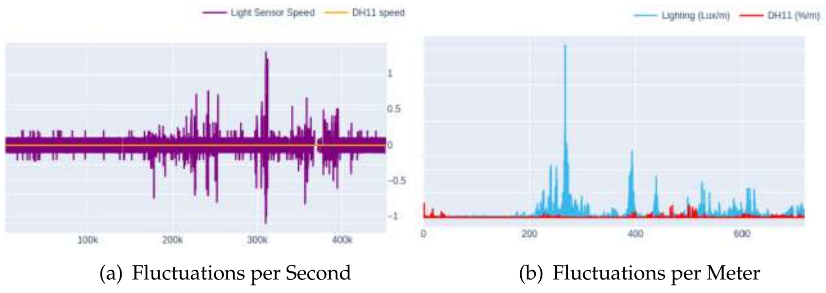
(c) Trajectory with Humidity as a Colour

Figure 4.36: Plot of Drone GPS Position during Flight with Lighting represented as a Colour

clouds during the procedure.

Sensor Range in Time A first graph presents the magnitude of the changes in light and temperature, by computing their rates of change over time. The magnitudes are normalized by their operating ranges: 20-90% of Relative Humidity for the DHT11 sensor, 20-150mV of ADC voltage for the LDR sensor.

$$\frac{\Delta r}{t} = \frac{r_f - r_i}{r_{max} - r_{min}} * 100 \tag{4.1}$$



(a) Fluctuations per Second (b) Fluctuations per Meter

Figure 4.37: Fluctuations in Measurements of LDR and DHT11 Sensors

According to (a), the maximum values suggest that the LDR sensor records changes of 0.23 % to 0.69 % shift in operating range/reading vs the DHT11 sensor's 0.14% to 6.62 % shift in operating range/reading. The temperature and humidity vary less rapidly, and this is very apparent in the plots.

Sensor Range in Space To better evaluate the sensing speed, we investigate the maximum fluctuations per second, and then per meter. This data recording speed is used in (b) in coordination with the drone velocity as recorded by the Pixhawk setup, in order to obtain fluctuations per meter, independent from speed. This is done with the following equations.

$$\frac{\Delta r}{m} = \frac{r_f - r_i}{t_f - t_i} * \frac{1}{\bar{v}} \tag{4.2}$$

The following equation is taken from [409], whereas David Williams determines an empirical formula to convert the ADC voltage to lux for the Arduino's Light Dependent Resistor module:

$$\log(L_{lux}) = -1.4 * \log\{\max|V_{adc}|\} + 7.098 \tag{4.3}$$

Whereas the LDR sensor records 57.915-270.276 lux variation/meter, the DHT11 detects 0.062-45.556 % of Relative Humidity variation/meter. This shows the range of local changes over the field and it seems reasonable for stark changes in light vs more gradual changes in humidity.

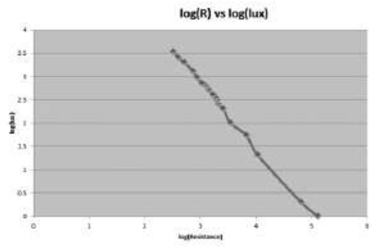


Figure 4.38: Empirical relationship [409]

4.4.3 Discussion

The proposed UAV architecture has proven itself effective at capturing fluctuating environment data. When examining the luminosity readings, the measurements are very consistent with shade/light regions, by changes of as much as 20% of the luminosity range. Sunny regions saturate the sensor, and shady regions are marked by sudden drops. This suggests a high accuracy, especially seen as the drone was piloted by hand.

The luminosity plot demonstrates very precise readings despite the drone's velocity. This is facilitated by rapid data logging at 100Hz. The fact that the drone was piloted by hand, on an arbitrary path with region overlaps, illustrates plainly how mobile mapping is a worthwhile tool for rapid data collection.

There is much less of a correlation between the readings and their position in space. We suggest two potential issues with the DH11 sensor:

- Movement may affect the temperature and humidity. We recommend to further investigate how the readings vary with altitude, speed and acceleration.
- Heat convection on the drone and the sensor itself may be recorded by the sensor instead of environment temperature. We recommend to further investigate how accumulated heat affect the readings.

Additionally, this experiment demonstrates that a payload drone can be extended to other types of sensors for other applications. The data acquisition setup proves to be functional. This setup was developed prior to the experiment with the goal to integrate many other types of sensors.

The flight was quite smooth and simple to undertake. As opposed to conventional means of environmental sensing [339], this flight requires no site preparation. This is largely due to the selected drone system, as well as the work done to automate the data acquisition procedures.

This procedure was greatly assisted by the datalogger, whereas GPS data and atmospheric data could be correlated without major issues. The correlation between different elements have uncovered a topography in an unexpectedly accurate manner. As we examine the systems that aid in practice, we note the importance of the drone as a platform for capturing scans of a 3 dimensional environment in a rapid, and timely manner. At the time of writing, Alliantech is compiling a marketing video for this environment sensing solution.

4.5 Structural Inspection of Vibrations

We sometimes wish to monitor the vibrations of a large structure. With conventional methods, this usually requires reaching the zone by foot and attaching the sensor in some form, and these pipes may be in dangerous or inconveniently placed. For example, it is a challenging task to monitor water flow through a sewer outlet pipe (Figure 4.39). In this project, we explore sensor placement with a UAV to demonstrate a simpler way to do vibration monitoring.

Figure 4.39: Instance of a water outlet where a drone can land and monitor water flow.

We designed a drone to land on the measurement site. To do this, a drone is equipped with the vibration data acquisition systema. A vibration probe mount is designed to fit the accelerometer on a leg of the drone. This full system is evaluated, first to determine whether it detects footsteps accurately, and the second to determine a range of frequencies at which the setup remains functional.

4.5.1 DAQ System Design

This section documents the design of the Vibration Data Payload selected in Section 4.3.1. It is composed of three stages.

The payload is installed on the drone in Step 1. Flight procedures are designed in Step 2. The DAQ Activation is done in Step 3.

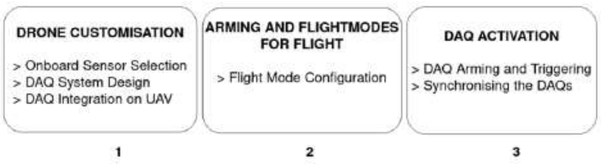

DRONE CUSTOMISATION
> Onboard Sensor Selection
> DAQ System Design
> DAQ Integration on UAV
1
ARMING AND FLIGHTMODES FOR FLIGHT
> Flight Mode Configuration
2
DAQ ACTIVATION
> DAQ Arming and Triggering
> Synchronising the DAQs
3

Figure 4.40: Preparations for Vibration Data Collection.

System for High-Sensitivity and High Sampling Rate

The Micro Slice by DTS [410] is a modular data acquisition system featuring unmatched flexibility and reliability for critical test applications. UAV flight is once such application that requires a small, reconfigurable and robust system. They offer a wide range of sampling rates of 10Hz to 500kHz and data storage of up to 16Gb. The Slice is used in aerospace analysis, automotive safety, biomechanics, and other safety-critical applications.

Figure 4.41: The DTS SLice Micro: a miniature, modular, rugged data acquisition system [410].

Module Design

An independent module is designed for the Vibration Data. An independent battery powers the logging unit, and the sensors attached to it. The Pixhawk board is included in this system since the Slice is armed via the Pixhawk board.

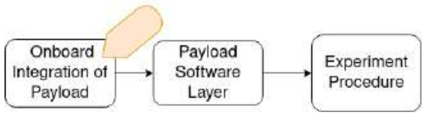


Figure 4.42: Setup Step 1.

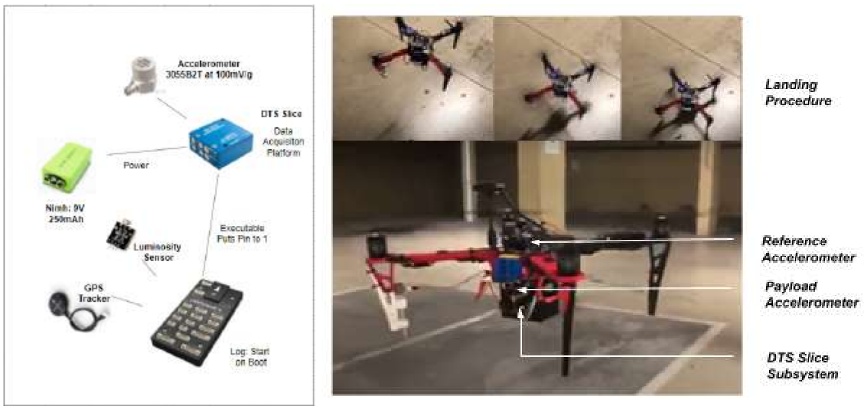


Figure 4.43: DTS Slice DAQ Subsystem

Instruments that were used in the system are pictured in Figure 4.43. These include a 3055B2T accelerometer and a data logging system. This accelerometer is directly connected to the DTS slice data acquisition system, which is a self-enclosed data logger.

Drone Mount Design

Hand probes are conventionally used to in frequency response tests on the terrain, as shown in Figure 4.44. This is used in our design to make contact between the drone and the ground. A mounting part is required to fix this hand probe to the drone. A part is designed and 3D printed for this purpose.

Figure 4.44: A conventional vibration probe tip is pressed against a surface by hand.

This mount places itself in the place of a drone leg (Figure 4.43). It measures the same size as the other legs, such that the drone's weight is equally distributed on the drone legs.

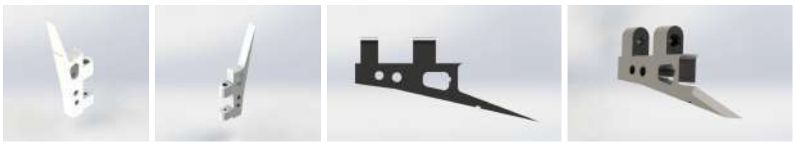


Figure 4.45: Renders of vibration probe mount.

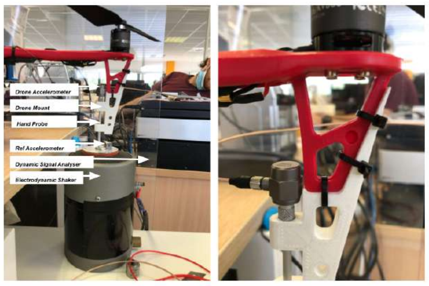


Figure 4.46: Installation of External Accelerometer

DAQ Control Layer

The DAQ is configured to activate and deactivate the datalogging process on command. To achieve this, we use custom activation firmware on the PX4 operating system.

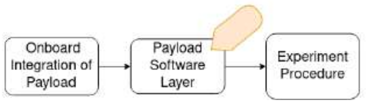


Figure 4.47: Setup Step 2.

Table 4.4: Flight Mode Assignment for RC Controller.

Boards	Switch	Activation	Deactivation	Prior to Activation
1. SLICE Micro	1-1	Custom App	Custom App	Arming
2. Pixhawk	1-3	On boot	On shutdown	None

The data-logging activation file was coded in C++, and compiled into an executable via the MAVLink protocol. During operation, it toggles a pin (the Pixhawk's FMU Channel 6), which is then detected by the data acquisition system in order to begin and end the logging.

The Pixhawk logger is used to log debugging messages at 100Hz, while the Slice data logger ranges from at 1kHz to 500kHz logging. The Pixhawk data logger is fitted with a 16Gb Micro SD card while the DTS slice supports up to 16Gb of memory.

4.5.2 Experiment 1: Detection of Footsteps

Experiment Procedure

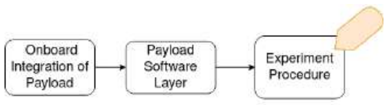


Figure 4.48: Setup Step 3.

Flight Procedure The experiment is conducted indoors. As the subject of the experiment is the measurement of vibrations, a sequence of steps is put in place.

1. The data acquisition system is activated.
2. The drone is flown and lands.
3. The passer-by walks at 1m from the drone.

Data Collection The DTS Slice is set up according to a few parameters. DAQ parameters are listed in Table 4.5.

The DAQ was sampled at 10kHz. The anti-aliasing filter (AA), output scale factor and sampling rate are defined, among other factors.

Table 4.5: Parameters of the Footstep Detection Test.

Test Description	Ground Accelerometer (ref)
Test Date	23/08/2021
Test Time	16:08:18
Sample Rate	10000 Hz
Hardware AA Filter (-3dB)	2900
Data Channel Number	1
Channel Description	3055B2T_REF
Software Filter (SAE Class)	NONE
Software Filter (-3dB)	NONE
Engineering Unit	g
Number of Pre-Zero Data Pts	0
Number of Post-Zero Data Pts	240042
Data Zero (CNTS)	57
Scale Factor (EU/CNT)	0.003781480491161
Scale Factor (mV/CNT)	0.378148049116135

Dataset The procedure was carried out on 23 August 2021 in an indoor parking space. The Google Drive experiment dataset [406] and a Youtube presentation video [407] are publicly available.

Results

The raw data is represented as the red markers in Figure 4.49.

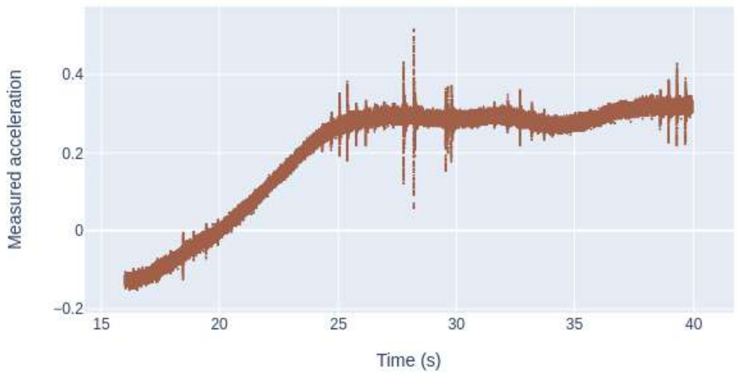


Figure 4.49: Raw data and final sensitivity curve

This raw data has considerable drift, which is due to an uncalibrated tool. This is a common issue when measuring with a single accelerometer as per [372]. To mitigate this drift, we make use of a Butterworth band-stop filter in Python, as it has a maximally flat frequency (ie. no ripples in the passband). This makes it one of the most popular and used band-stop filters.

```

    filtered_data = butter_bandpass_filter(raw_data, start_hz=7,
                    end_hz=12, sample_hz=25000, order=5)*0.8
```

Figure 4.50: Parameters for Band-stop Butterworth Filter

The filter is run between 7Hz and 12Hz, on a sampling frequency of 25kHz. The filter's transfer function is:

$$H(z) = \frac{(z^{-2} + 2z^{-1} + 1)^3}{(z^{-2} - 1.9z^{-1} + 0.9)^4} \tag{4.4}$$

The result is the following graph.

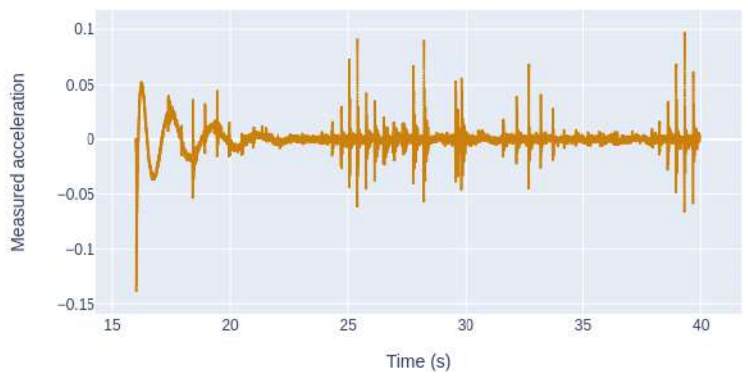


Figure 4.51: Filtered vibration curve

A single footstep is isolated from 23s to 28s. This data shows each footstep clearly with a gradual increase in signal amplitude (energy) followed by a gradual decrease. This same behaviour is seen in other parts of the rectified data.

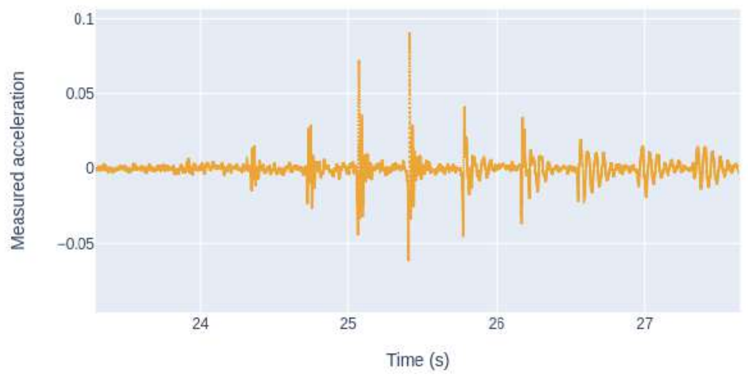


Figure 4.52: Inspection of a set of footsteps

The raw floor vibration signal behaves similarly to those in Figure 4.53 with footsteps increasingly far from the sensor. The decrease in signal amplitude (energy) correlates with increasing footstep-sensor distance.

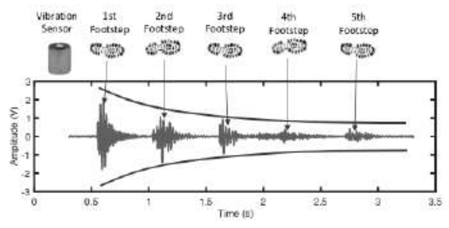


Figure 4.53: Footsteps as detected by [372]

Hypothesis Validation

The final graph shows footsteps quite neatly as they approach and leave the accelerometer. The hypothesis is therefore valid, and further discussion features in the Chapter Discussion, Section 4.5.4.

4.5.3 Experiment 2: Sensor Sensitivity Evaluation

Vibration tests reproduce the vibrations undergone by goods during transport. It is a good design practice to ensure the resonant frequencies are well above any vibration loads the product is likely to experience.

In order to perform these tests, there are different types of machines, such as the vertical vibration testing systems that only have one degree of freedom, or vertical, pitch and roll systems which have three degrees of freedom and very accurately reproduce the vibrations suffered during the transportation of goods [411].

Procedure

Experiment Equipment A vibration shaker is a device used in vibration testing to excite a structure. The shaker can perform a sinusoidal sweep, that is, a sinusoid whose frequency varies with time. This will reveal a resonant frequency, which means the vibration induced displacements are maximized.

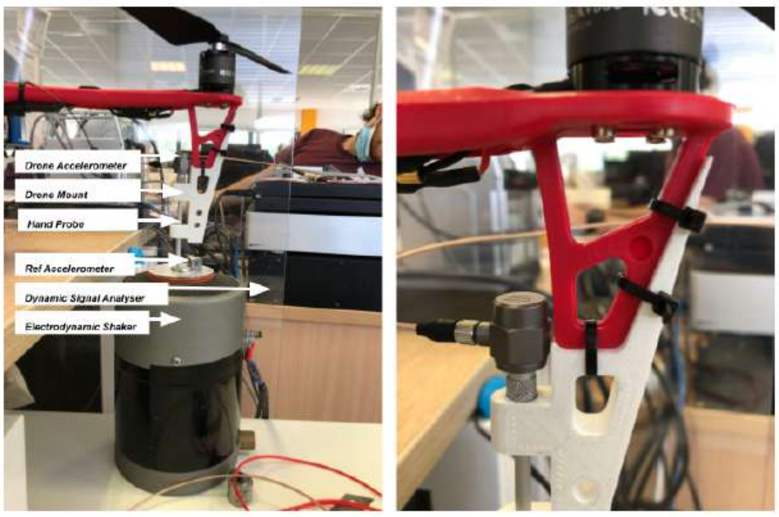


Figure 4.54: Vibrating Pot Experimental Setup

Data Collection The Dynamic Signal Analyser is set up according to a few parameters. DAQ parameters are listed in Table 4.6.

A sinusoidal sweep runs on a frequency range from 50Hz to 2kHz, with an average sweep rate of 2 octaves per minute.

Parameters	
Start amplitude	0.30g
Start frequency	50.00 Hz
End amplitude	0.30g
End frequency	2000.00 Hz
Sweep rate	2.00 Oct/min
Min to Abort	-6.00dB
Min Tolerance	-3.00dB
Max Tolerance	3.00dB
Max to Abort	6.00dB

Table 4.6: Parameters of Vibrating Pot Test.

Results

The Dynamic Signal analyser shows the z-translational response of the drone-mounted accelerometer (dark blue curve) as opposed to the drive accelerometer (light blue curve). Translations on the x and y axes are also included here (black curves) for comparison.

Response Frequency of Vibration System Of particular note are the noticeable wobbles in gain below 1kHz, these seem highly correlated to x and y translational forces. The translational forces on the vibrating pot can be attributed to an less-than-optimal calibration of the pot. The z-translation response peaks around 1.8kHz, with no similar response in the drive accelerometer. Following this, we develop a sensitivity curve from the ratio between the reference and drive translations. This sensitivity curve demonstrates a gain change away from the neutral value of 1:1.

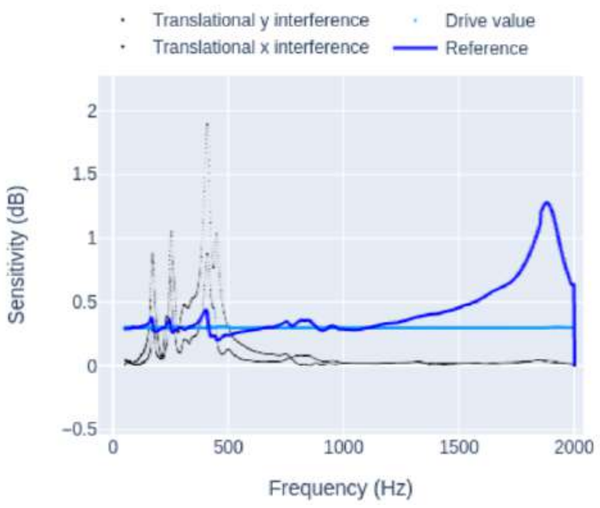


Figure 4.55: Vibration of Accelerometer During Vibration Pot Experiment

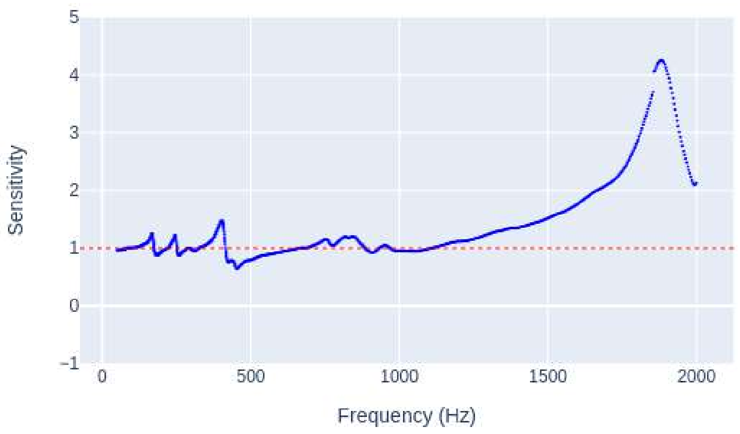


Figure 4.57: Final Sensitivity Curve for UAV Vibration Probe

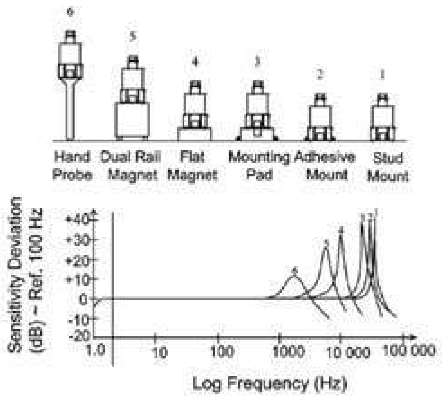


Figure 4.56: Sensitivity deviation according to measurement technique [411].

This sensitivity curve demonstrates the gain over the range of frequencies. The wobbles remain within a 50% distance from 1, until about 1.5kHz, increasing to a discontinuity at 1.7kHz. This discontinuity in output occurs at the peak frequency, suggesting a resonant frequency.

4.5.4 Discussion

A quick demonstration in Experiment 1 shows that that the drone can carry such an inspection autonomously, by recording data from the moment that it is landed to the moment that it takes off again. This opens up to other usecases. For instance, monitoring vibrations as signs of activity in an area, and leaving the area on command. A UAV is well-suited to this activity. The UAV can be placed on the side of a road where it detects the number of cars driving past. It can be placed on large sewer pipes to monitor the rate of flow. In these two examples, the UAV has clear advantages in that it can be placed without physical intervention, and it can remain in place to monitor the situation. Further studies have proposed solutions [412] using Unmanned Ground Vehicles (UGVs) for robotic crack inspection and mapping, delamination and concrete quality assessment, and even a complete mechatronic system

Table 4.7: Key findings in Chapter 4.

Test Description	Findings
Semi-autonomous scan of a zone	
DHT11 Sensor Range in Time	0.14% to 6.62 % shift in operating range/reading
DHT11 Sensor Range in Space	0.062-45.556 % of Relative Humidity variation/meter
LDR Sensor Range in Time	0.23 % to 0.69 % shift in operating range/reading
LDR Sensor Range in Space	57.915-270.276 lux variation/meter
Structural Inspection of Vibrations	
Range of Vibration Sensing System	0-1.8kHz
Margin of Error up to Natural Frequency	50% margin of error

for high-efficiency bridge inspection. These are foreseeable within the scope of this work.

Feng et al. [402] envision a scenario in which accelerometers are mounted onto UAVs, which then are able to gather acceleration signals by self-attaching to beams under bridges. In order to identify the bridge acceleration response, Feng et al. perform a simulated frequency domain decomposition (FDD) based on mode shape extraction.

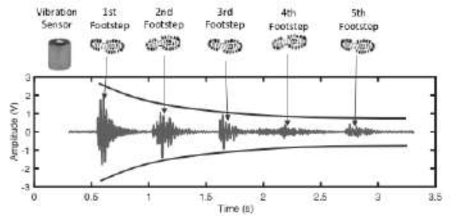


Figure 4.58: Footsteps as detected by [372]

The Accelerometer Mount designed in this section was based to house a Hand Probe. When used in conventional work, as per Figure 4.59, hand probes show losses in sensitivity around 1000 Hz. The sensitivity of the accelerometer remains within a 50% error margin until about 1.5kHz. With this magnitude of a margin, the data gathered seems comparable to conventional hand probes.

All in all, an inspection of the results shows that this solution is no less effective than a hand probe. Quite starkly, the literature lacks mention of vibration probes on UAVs, and yet the experiment has interesting results. A UAV that can be placed autonomously to detect human footsteps, could also monitor other sources of activity on a 1kHz range with no major losses in sensitivity.

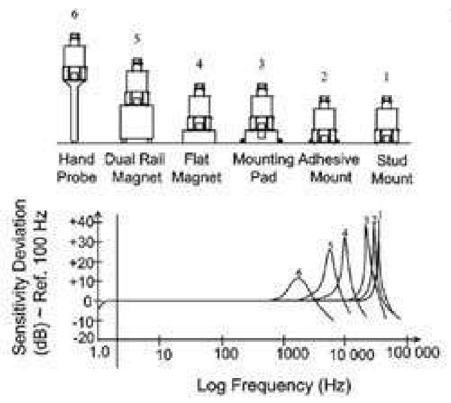


Figure 4.59: Sensitivity deviation according to measurement technique [411].

It can safely be said that this experiment has managed to demonstrate a few elements:

1. High-sampling and high-precision equipment can be of interest to UAV practitioners.
2. A vibration probe can assist in Structural Inspections.

This also highlights the utility of the data acquisition system employed in this experiment: the DTS Slice can collect data at a sampling rate of 1kHz, suggesting that the drone can be used for finely tuned measurements. With this experiment, there is a case for sensitive equipment that can be placed autonomously, remotely, and possibly even perform automatic tasks. In other words, High-sampling and high-precision equipment can be of interest to UAV practitioners.

At the time of writing, the author has been informed that Alliantech will apply for a Soleau envelope on the accelerometer mount.

4.6 Conclusion

This chapter marks a distinct turn towards outdoor UAV operation. This has required the development of a drone flight ecosystem as well as a custom environment for integrating new sensors easily. The experiments in this chapter have expanded the usecase of a drone as we thought it possible initially: from gathering data with one sensor, the drone could gather from multiple sensors over a large field. These experiments are not major advances in the field - but as proof of concepts, they validate the use of onboard sensing for further industry-ready applications. The environment monitoring experiment uses common atmospheric sensors, but it can be extended to air pollution monitoring, a subject that is currently gaining traction. The vibration probe experiment sets the scene for automated drone services, as it explores the very specialised task of vibration monitoring in-situ. A key takeaway are the tests performed during drone development: as custom drones are developed for various clients, the very platform involved in operation can be optimised using AllianTech's savoir-faire. All in all, this chapter has ventured well into the current state of the art of drone engineering, and we hope it has offered insights for innovation at AllianTech.

This automation of the procedure, albeit a proof of concept, is a first step towards more complex structural inspections. A carrier drone has shown to streamline the data collection process, by automating different fly-by procedures, safeguards, and scheduling the data collection. The drone's functionalities can be fitted to aid professional practitioners in the course of their work. Regarding the thesis problematic, these tools allow for many tasks, and services, to be automated. Drone tasks take on a new form: as collaborative endeavours, managed smoothly and according to the needs at hand.

Conclusion 5

Chapter 2: A Testbed Environment for Task Development

Table 5.1: Key findings in Chapter 2.

Test Description	Value
Volume of Flight Arena Localized by Motion Capture	90.46%
Maximum Flight Error Recorded in Hover Test	±72.24 mm ±0.096 rad

We document the design of a development and demonstration testbed conform to existing research. We describe a procedural task-based architecture to complement an existing swarm stack. We demonstrate that the runtime environment is capable of coordinating multiple robots. A custom high level interface wraps the testbed towards more complex tasks, and it is demonstrated in a multi-drone choreography.

Drone 2 is further from the arena and exhibits more stability. The drone that is furthest from the antenna does not have more pose error, and the hypothesis is rejected.

The drone choreography has shown a functional workspace for multi-robot groups, whereas a predictable, controllable group of drones with well-defined goals executed tasks. This culmination of swarm and task interfaces the infrastructure for new technologies and for prototyping functionalities. Similarly to other drone laboratories [375][393] and experimental spaces[360], we've put in place a smart ecosystem for multi-robot development.

The development of a testbed, and all its associated functionality, reveals the complexity of robotic development. Just as robots demand attention to detail, so do the frameworks that surround them. As UAV research continues to grow, so do the number of applications. Flight Testbeds are quickly becoming a vital element of the drone development process. A framework for multi-robot task execution is a stepping stone to more complex tasks. It may be expanded to other models of robots, groundbased or airborne.

Chapter 3: Experimentations for Human-Drone Interfaces

We investigate a Mixed Reality Interface for the Testbed, as well as methods of drone Piloting using a Computer Vision algorithm. The utility of the framework is demonstrated by using it for two different tasks: quadrotor piloting using computer vision and collision-free flight of multiple UAVs. Building on existing frameworks like MediaPipe Hands, and Unity3D, we create perception pipelines for semi-autonomous flight, and we proceed to evaluate the response latency of these pipelines.

Table 5.2: Key findings in Chapter 3.

Test Description	Result
Gesture Piloting	
Gesture Recognition Effectiveness	56.3%
System Response Time	271.0 ms
Mixed Reality Interface	
Latency of Pose Transmission	$(6.68 \times 10^{-8})e^{0.19066t}$ ms
Latency of State Changes	89t ms
System Latency	$89(7.5 \times 10^{-7}e^{0.19066t} + t)$ ms

The gesture interface used to pilot the drones is given 56% accuracy. While the pipeline is based on MediaPipe Hands, the pose classification was hardcoded, and the software can then be improved with a neural classifier or an ML pipeline. In practice, the errors were filtered out by the drone control pipeline.

We have developed a Mixed Reality pipeline that transmits real objects into a simulator and game engine. While this approach is accurate and shows high image fidelity, the pose transmission suffers from a linear $(6.68 \times 10^{-8})e^{0.19066t}$ ms delay. Further work might be able to detect the root cause of this issue.

While the Mixed Reality Interface provides us with a simulated graphics engine, a communication channel was put in place that would communicate virtual events to the robot swarm. However, the collision experiment has demonstrated a cumulative delay of 89t ms for a single quadrotor, and this can only increase with larger swarms and more complex manoeuvres. Since latency is a primary measure for image streaming and high performance drone tasks, we suggest the exploration of a network interface more focused on performance, and possibly the integration of existing simulators like Flightmare within the testbed.

With Human-Drone Interfaces, a new reality is offered to us: one where drones and humans meet, where drones can somewhat become extensions of our human selves. These experimentations have been a taster of what is possible, as laying down the framework is a first step towards a better world. The concept of human extension is fascinating, for it suggests that the extension is an addition: in that way, a drone assumes the role of a habilitating tech.

Chapter 4: In Vivo Deployment for Industrial Environments

Table 5.3: Key findings in Chapter 4.

Test Description	Findings
Semi-autonomous scan of a zone	
DHT11 Sensor Range in Time	0.14% to 6.62 % shift in operating range/reading
DHT11 Sensor Range in Space	0.062-45.556 % of Relative Humidity variation/meter
LDR Sensor Range in Time	0.23 % to 0.69 % shift in operating range/reading
LDR Sensor Range in Space	57.915-270.276 lux variation/meter
Structural Inspection of Vibrations	
Range of Vibration Sensing System	0-1.8kHz
Margin of Error up to Natural Frequency	50% error margin

Applications are explored for UAVs as Mobile Sensing Platforms, with high-sampling and high-precision equipment. We design a carrier drone and Onboard Data Acquisition systems and we put them to practice along standards defined by industrial practitioners. Two payloads are tested in outdoor flight, for atmospheric data and vibration data, and we characterise the sensors used for these tests. A vibration probe is designed and our tests demonstrate its relevance in the field of mobile sensing.

Through the means of a payload damping test, the hypothesis is rejected, insofar as more shock absorption material did not give a lower gain in the frequency response. Since the objective of this test is about maximising the damping in the range of interest, Payload A is installed onto the drone. We suggest that that this test be carried out over a wider range of materials, and for other types of flight, in order to determine an optimal damping volume for different drone designs.

The luminosity plot demonstrates very precise readings despite the drone's velocity. This is facilitated by rapid data logging at 100Hz. The fact that the drone was piloted by hand, on an arbitrary path with region overlaps, illustrates plainly how mobile mapping is a worthwhile tool for rapid data collection.

The footstep detection tests demonstrate that the drone can indeed record data with similar quality as that of a planted geophone. This finding is significant because it demonstrates that an activity like vibration monitoring is not confined to a manual exercise by practitioners. A drone can gather data remotely. At the time of writing, Alliantech is compiling a marketing video for this vibration sensing solution.

We have characterised a drone vibration monitoring solution and we confirm that our drone can monitor sources of activity on a 1kHz range with no substantial losses in sensitivity. This finding is significant because it demonstrates that the drone's vibration mount is comparable to professional vibration monitoring methods, and the UAV can therefore be used as an alternative for structural inspections.

An industrial perspective has revealed the opportunities for drones in sensor networks: carrying sensors, placing and retrieving sensors, and overall assisting in the data collection process. This instrumentation perspective has proven to be full of potential: one flight test fuses sensor data without much issue, and in another, a vibration probe has shown its efficacity. This UAV can now be one tool in a toolkit for a company wishing to take measurements of a zone or to simplify a logistical operation.

Perspectives

UAVs have shown to be a rapidly expanding market : with new applications found every day, UAVs are the center of a range of research, scientific and industrial fields.

Human-Drone Interfaces are still under development: The state of the art of HDI includes many modalities of interaction, and seeking new and more intuitive ways of interfacing with drones.

Finally, remote sensing and structural inspections are benefiting from the low-cost, time-cutting solutions that come with mobile sensing. In this paper, we have identified several new approaches to conventional tasks.

5.1 Acknowledgements

DE VINCI INNOVATION CENTER

Chapter 2. A Testbed Environment for Task Development was developed at the De Vinci Innovation Center. Clement Duhart of the DVIC showed much faith to enable such a project.

Chapter 3. Experimentations for Human-Drone Interfaces was developed with the help of the DVIC Drone Team. Adem, Matt and Berenice, thank you for joining yourselves to me. I trust that future generations of students will make good use of this work.

The De Vinci Innovation Center is a growing community of students, startups, hackers and fabricators at La Défense, Paris. They have a laissez-faire that lends itself to multi-disciplinary projects and a growing innovation laboratory.

AllianTech® SENSORS, SYSTEMS AND SOLUTIONS

Chapter 4. In Vivo Deployment for Industrial Environments was developed with members of AllianTech, a company specialised in the supply of sensors and sensor system technologies.

While AllianTech are rooted in France, they liaise with global brands on subjects of acceleration, angular speeds, force, pressure, temperature and mechanical vibrations. Aside from their sales and marketing department, they have a technical service to adapt products to the client, offer after-sale support and assembly solutions for several clients. Their head office is based in Paris and they have a design office in Toulouse.

A final thanks to Clement Duhart and Marc Teyssier, without whom I would not have strived for this goal.