Universiteit Leiden
The Netherlands

Bachelor Datascience and Artificial Intelligence

Transforming the Crazyflie 2.1 Nano-Drone into an Interactive Pet

Ernie Tsie-A-Foeng

Supervisors:
Mike Preuss
Evert van Nieuwenburg

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

Abstract

This thesis explored the Crazyflie system, the Flow Deck, the MultiRanger deck and some of its limitations as well as the development of a multimodal drone control system that enables interaction with the Crazyflie 2.1 nano quadrotor using hand gestures and voice commands. Traditional drone control relies on joystick based input, which can be unintuitive and requires significant practice. This system explores more natural human communication methods to lower the learning curve and improve user experience by the means of an interactive pet.

The project uses gesture recognition using MediaPipe Hands with a K-Nearest Neighbors classifier and voice recognition using the VOSK speech engine coupled with natural language processing techniques such as sentence embeddings and cosine similarity. This thesis used a design-based research approach.

Results showed that under ideal conditions, both input modes performed reliably, with added features like idle animations and obstacle avoidance improving the pet like experience. The drone responded in real time and could be operated hands free. This work serves as an early step in drone research, with the Crazyflie 2.1, at Leiden University.

The full implementation, including earlier prototypes, is available on GitHub:
https://github.com/exu0201/crazyflie-interactive-pet

Contents

1 Introduction

The use of drones has seen a significant increase in popularity in various fields, such as research, education, logistics and recreation. These unmanned aeruial vehicles (UAVs) can be used for a wide range of tasks, including surveying, delivery, filming and autonomous navigation. Most drones are controlled manually by a human with the use of a remote controller. These controllers typically consists of two joysticks, one for managing the altitude (thrust) and rotation (yaw) and the other one for forward/backward (pitch) and lateral (roll) movement.

While the usage of these joystick based control remains the standard, it does present several limitations. This method of controlling can be unintuitive and often requires the usage of both hands, which increases the difficulty to master the controls of a drone for users. But humans have evolved a more natural and intuitive way of communicating, namely gestures and voice. These two are natural communication methods and would lower or even remove the learning curve of controlling a drone. By implementing a gesture and voice based interface, we can make drone operations more user friendly and efficient.

We will be exploring the implementation of such multi modal interface that will allow users to control a drone through its voice and gestures.

The Crazyflie 2.1+, categorized as a quadrotor, is an open-source experimental platform that is used widely for research and education in robotics and control engineering [Gie17]. The Crazyflie is an easy-to-use and modifiable UAV, making it perfect for educational purposes. Using tools such as MediaPipe, Vosk and a K-Nearest neighbors classifier, voice commands and hand gestures will be used to control the drone.

Since Leiden University has not really done research involving drones, this project represents a great first step toward integrating UAVs into its research and educational activities. As one of the first students to work with drones in this context, this thesis aims to explore and establish a foundation for future drone related research at the university.

Thus the primary aim of this thesis is to explore the Crazyflie system, some of its limitations and controlling of the Crazyflie drone without relying on traditional joystick based input. In addition to the drone having a pet like behaviour.

> **Can a Crazyflie 2.1+ nano-drone be transformed into an interactive pet that is controlled by hand gestures and/or voice commands?**

This question will be addressed through the development and testing of a multimodal control system. The effectiveness of this system will be evaluated based on responsiveness, accuracy and the interactive behaviour of the drone.

1.1 Thesis overview

This bachelor thesis, carried out at the Leiden Institute of Advanced Computer Science (LIACS), under the supervision of Mike Preuss, explores the implementation of a multimodal drone interface

using gesture and voice control. This chapter contains the introduction; Section 2 discusses related work; Section 3 provides technical background on the Crazyflie 2.1 platform, its supporting hardware decks, and the software libraries used; Section 4 describes the design-based approach used and the integration of each pipeline; Section 5 discusses the results and limitations. Finally, Section 6 talks about the conclusion and suggests directions for further research and system improvements.

2 Related Work

2.1 Gesture Based Drone Control

Prior work has explored gesture recognition as an intuitive control interface for drones. Dadi [Dad18] developed a multi tier system enabling gesture based navigation of Crazyflie drones using a smartwatch (Moto 360) equipped with a gyroscope. The sensor data was transmitted from the wearable to a smartphone via Bluetooth, then forwarded to a PC using socket programming, and finally delivered to Crazyflie drones via a Crazyradio dongle. This architecture enabled real time, wireless gesture based control of single or multiple drones.

The system demonstrated that inertial sensor data from consumer wearables could be effectively repurposed for drone navigation tasks. Simple arm movements such as lifting or tilting the wrist could be mapped to basic drone maneuvers like takeoff, directional movement, and landing. While the approach allowed for hands free control, it relied on a layered communication pipeline that introduced latency and increased system complexity. Additionally, the need to wear a specific device constrained the naturalness of interaction and limited accessibility for general users.

2.2 Gesture Recognition Using Wearables

Choi et al. [CHO17] proposed an intuitive drone control system using hand gestures captured by a smartwatch's IMU. The system uses a recurrent neural network with LSTM cells to classify nine distinct gestures in real time. These gestures are mapped to drone commands like takeoff, hover, and land. The Crazyflie nano quadrotor responds to recognised gestures via a low latency ROS based interface. The system also aligns the drone's heading with that of the user for intuitive maneuvering, demonstrating high classification accuracy and minimal latency in real world tests. Their method leverages the temporal features of IMU signals using a hybrid deep learning architecture composed of convolutional and LSTM layers. The gesture classifier is trained offline but operates in real time, achieving 98.9% accuracy on a test set of 3,000+ samples. The inclusion of a heading alignment feature based on magnetometer data addresses the common problem of spatial disorientation, enabling the drone to move in a direction relative to the user rather than its own frame of reference.

2.3 Simulink Based Control of Nano quadrotors

Meghana Gopabhat Madhusudhan [Mad16] developed a nonlinear mathematical model of the Crazyflie 1.0 nano quadrotor using Simulink. The study involved designing and tuning attitude and altitude controllers based on the quadrotor's dynamic equations of motion. The full model incorporated six degrees of freedom and considered rotor thrust, body frame transformations, and

aerodynamic drag. These controllers were then implemented in a real time Simulink environment, enabling stable autonomous flight using model-in-the-loop simulation.
The project validated that Simulink could serve not only as a modeling tool but also as a control interface for small UAVs. Real time controller output was routed through an interface board to the drone, demonstrating closed-loop operation without the need for onboard computation. Although the system did not include gesture or voice interfaces, it represents a significant contribution in terms of control stability and precise system modeling. Compared to the current work, which prioritizes real time human interaction through voice and vision, Madhusudhan's study emphasizes system-level dynamics and controller fidelity.

2.4 Skeleton-Pose-Based Gesture Recognition with Monocular Vision

Marinov et al. [MVW+21] introduced Pose2Drone, a modular Human Drone Interaction framework built entirely on RGB based skeleton pose estimation using OpenPose. The system classified user orientation (front, side, back) and recognized arm gestures through geometric analysis of 2D joint positions, enabling both gesture based control and a face following mode. A neural network based monocular distance estimation, trained on body and face features, allowed the drone to maintain a safe distance without depth sensors.

The gesture vocabulary included eleven commands covering translational and rotational movement as well as special actions like "circle around" and "take a photo," achieving an average recognition accuracy of 93.5% in varied lighting and orientation conditions. By relying only on the drone's onboard camera, the framework offers greater portability than depth based solutions and is well suited for outdoor use. However, performance is sensitive to pose estimation quality, and certain gestures, particularly "down," were harder to detect due to physical execution challenges.

2.5 Body Gesture Control Using Depth Cameras

Gio et al. [GBV21] developed a Natural User Interface (NUI) for drone control using a Microsoft Kinect depth camera to capture full body skeletal data. The system calculated joint angles in real time and mapped them to drone movement and rotation commands, while incorporating continuous flight speed control by linking gesture amplitude to velocity. To extend beyond basic piloting, an interactive gesture controlled menu enabled tasks such as photo and video capture, flips, and takeoff/landing, all with visual feedback from the drone's camera and a live skeleton overlay. Battery status monitoring and audio alerts supported operational safety.

User studies with both experienced and new pilots found the interface intuitive, particularly for beginners who required minimal instruction. While experienced pilots completed tasks slightly faster with traditional controllers, the gesture system offered a more natural interaction style. However, its dependence on Kinect hardware restricted use to indoor or fixed-sensor environments, and latency from the Tello SDK introduced noticeable delays between gesture execution and drone response.

3 Background

This chapter will provide information get familiar with the Crazyflie and to understand the implementation of the multimodal drone control system based on gestures and voice.

3.1 Drone Control Basics

quadrotor drones like the used Crazyflie 2.1 are controlled by adjusting four main parameters: **thrust, yaw, roll and pitch.**

- **Thrust** - controls the altitude of the Crazyflie 2.1
- **Yaw** - controls the Crazyflie's rotation around its vertical axis. This rotates the quadrotor left or right and thus changes the direction the front of the quadrotor is facing
- **Roll** - controls the Crazyflie's rotation around its horizontal axis from front and back, which causes it to move left or right.
- **Pitch** - controls the Crazyflie's rotation around its horizontal axis from left to right, making the Crazyflie able to fly forward or backward.

These drones can be controlled with either low level commands, which is the direct control of the motor, or high level commands, in which a target position is specified. This project will focus on the high level commander of the Crazyflie. This allows the Crazyflie to be able to move to a specific coordinate in 3D space using a go_to(x, y, z, yaw, duration) command.

Figure 1: The Crazyflie 2.1 with Flow and MultiRanger Decks attached

3.2 Crazyflie Overview

The Crazyflie 2.1+ is an open-source experimental platform, easy-to-use and modifiable UAV, which makes it perfect for this project. Researchers have used the Crazyflie in various studies in autonomous navigation, swarm coordination, and human-drone interaction. It weighs 27 grams and is even smaller than some modern smartphones. The Crazyflie 2.1+ is also equipped with Bluetooth LE and a long-range and low latency radio [AB25]. This equipment gives the user the added ability to control the quadrotor with their mobile as their controller. In combination with a Crazyradio 2.0, it is able to display data on the computer and use a generic game controller as its controller.

3.2.1 Crazyradio 2.0

The Crazyradio 2.0 is an open source, long range USB radio dongle built around the Nordic Semiconductor nRF52840 chipset. It features a 20 dBm power amplifier and low noise amplifier (LNA) to enhance signal performance. Although it is designed for the Crazyflie, its open firmware and Python based API make it also an useful tool for applications requiring lower latency communication than Wi-Fi, with less emphasis on bandwidth. The Crazyradio 2.0 is also equipped with an USB bootloader, which allows for firmware updates without additional hardware. It also has a 64 MHz Cortex-M4F processor, 1 MB of flash memory and 256 KB of RAM. The Crazyradio 2.0 operates in the 2.4 GHz ISM band and supports multiple radio modes, including Bluetooth Low Energy (BLE) and IEEE 802.15.4 [Bit25a]. This radio dongle allows the wireless communication between a PC and one or more Crazyflies. It is used as the primary way of controlling the Crazyflies via cflib, which will be discussed in more detail later in this thesis.

Figure 2: The Crazyradio 2.0 used in this project

3.2.2 Flow Deck

The Flow Deck is an expansion board that adds optical flow and time of flight capabilities to the Crazyflie. The Flow Deck is mounted under the Crazyflie and includes a PMW3901 optical flow sensor and a VL53L1x ranging sensor. The PMW3901 optical flow sensor is used to measure the movements of its horizontal plane and the VL53L1x is used to measure the drone's altitude [Bit25b].

The optical flow sensor works by capturing motion relative to the ground by comparing image frames at high speed, which allows the Crazyflie to estimate horizontal position changes. The ranging sensor provides accurate vertical distance measurements from the floor, enabling altitude control. These two sensors combined allow the drone to be able to fly pre-programmed scripts.

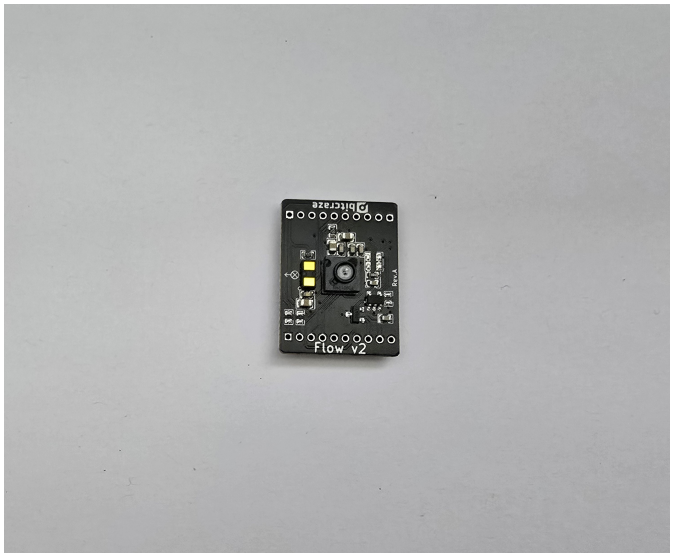


Figure 3: The Flow Deck used in this project

3.2.3 MultiRanger Deck

The Multi-ranger Deck is also an expansion deck that is mounted on top of the Crazyflie. It has the ability to detect nearby objects by measuring distances in five directions: front, back, left, right, and up. It uses multiple VL53L1x time-of-flight sensors to provide millimeter level precision for ranges up to 4 meters. This allows for implementing basic collision avoidance, range based behaviours and interactive responses to the environment [Bit25d].

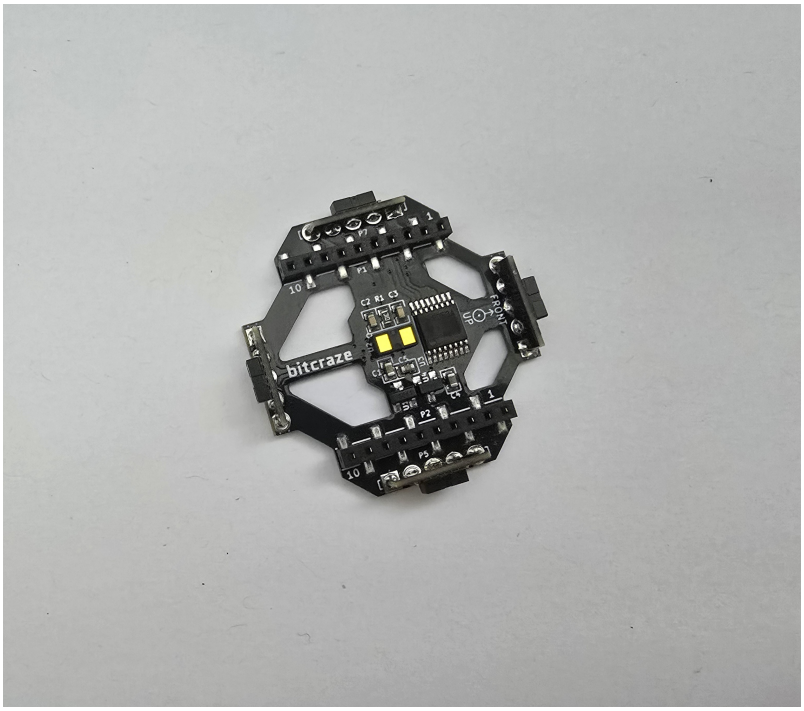


Figure 4: The MultiRanger used in this project

For optimal performance, the Multi-ranger Deck is often used in combination with the Flow Deck V2. Together, these two decks give the Crazyflie a great understanding of its surroundings, enabling it to sense nearby objects.

3.3 CFlib (Crazyflie Python Library)

The CFlib is the official Python library developed by Bitcraze to communicate with and control Crazyflie drones. It provides a high level and low level interface to interact with the drone wirelessly using radio links. The library contains tools for initialization, logging, communication, and direct motor control, making it a core component in almost any Crazyflie based project.

CFlib abstracts away the complexities of packet communication, allowing developers to focus on sending commands such as takeoff, land, or go to a specific location. It also includes classes for logging drone sensor data, interfacing with the Kalman filter (for position estimation), and managing connections.

3.4 HighLevelCommander

The `HighLevelCommander` is a module within cflib that allows for simple movement commands to be issued to the drone at a higher level of abstraction. Rather than manually calculating motor thrust or using PID control, this interface enables the user to send pre-built commands like takeoff](), land(), and go_to(x, y, z).

This simplifies movement control significantly, especially in position controlled environments

where a Flow Deck or Loco Positioning System is used. It handles things like trajectory generation and position maintenance internally, which is useful for projects focused on gesture or voice input rather than raw flight dynamics.

3.5 Kalman filter

The Crazyflie nano quadrotor uses an onboard Kalman filter for real time state estimation. This filter fuses data from multiple onboard sensors, including the inertial measurement unit (IMU), optical flow sensor, and range sensors, to estimate the drone's position and orientation in 3D space. One of the key outputs of this estimator is kalman.stateZ, which represents the estimated altitude of the drone above ground level, measured in meters. This variable is essential for height-dependent flight maneuvers, such as takeoff, landing, and vertical translation.

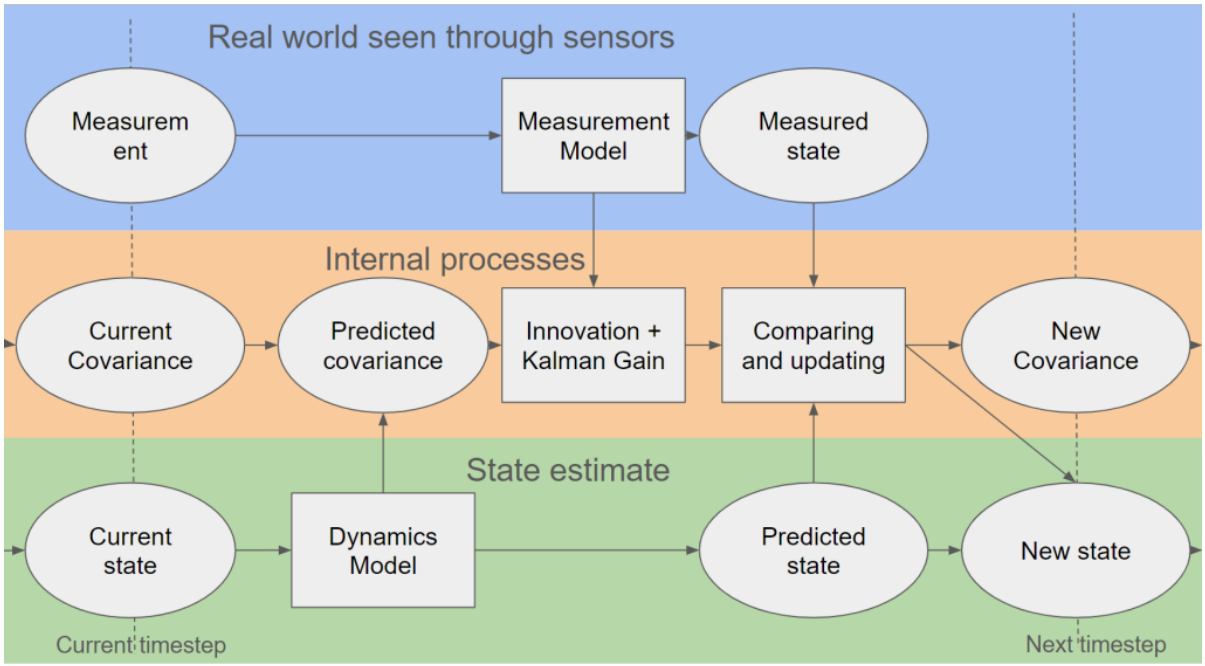


Figure 5: Visual representation of the Kalman filter process used in the Crazyflie estimator, showing prediction, measurement fusion, and state correction. Taken from Bitcraze [Bit24].

The Kalman filter operates by predicting the drone's state based on physical models and then correcting those predictions using incoming sensor data. In early stages of flight or when the environment lacks sufficient features (e.g., for optical flow), the estimate may be noisy or unreliable[Bit24]. As shown in Figure 5, the Kalman filter integrates predicted and measured states to estimate position variables like `kalman.stateZ`.

3.6 MediaPipe & Hand Landmarks

MediaPipe Hands is a machine learning framework developed by Google that enables real time hand tracking using computer vision. It is capable of detecting 21 unique landmarks on a human hand, which correspond to key joints and fingertips [Goo19]. These landmarks are tracked in a

three dimensional space (x, y, z), resulting in 63 values per hand. When both hands are detected, the total number of features becomes 126.

The system is commonly used in gesture recognition tasks, as it provides detailed structural information about hand poses. Each landmark represents a consistent location, such as the tip of the index finger or the base of the palm. For example, by analyzing the relative distance between the fingertips and the palm, the system can distinguish between an open hand and a closed fist. This makes MediaPipe a useful tool for applications ranging from sign language recognition to augmented reality interaction.

3.6.1 K-Nearest Neighbors

The K-Nearest Neighbors algorithm is a widely used and intuitive method for classification. It is considered a nonparametric algorithm, which means it does not make strong assumptions about the underlying data distribution. Instead, KNN operates by comparing new input data to previously seen examples.

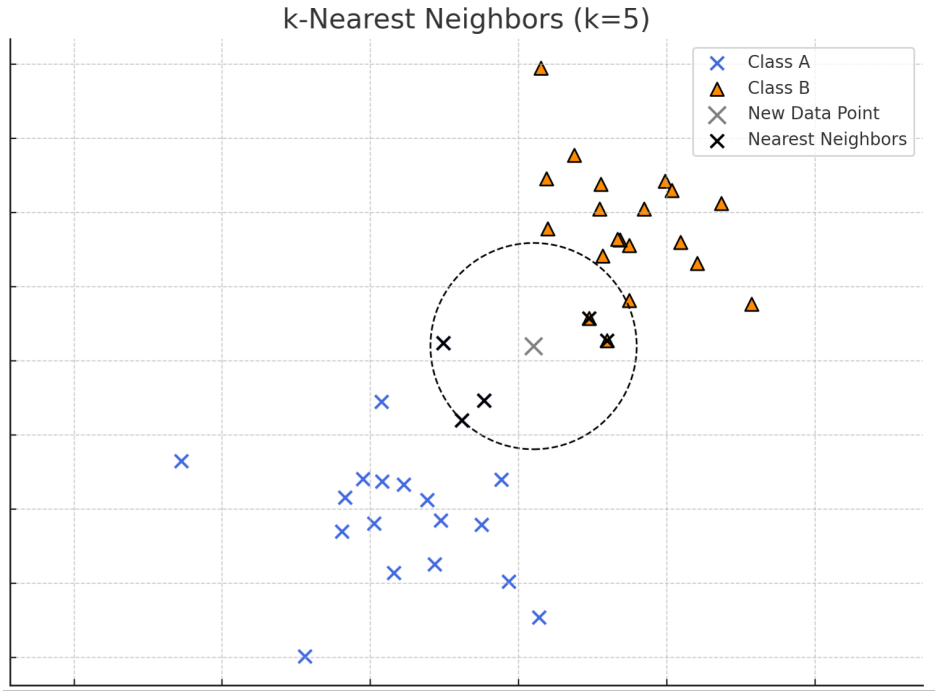


Figure 6: Simple visual representation of the K-Nearest Neighbors

The way it works is simple: when a new data point needs to be classified, the algorithm calculates the distance from that point to all examples in the dataset. The "k" closest data points are selected, and the most common label among them becomes the predicted outcome. For instance, if a system is trained to recognise hand gestures and a new hand pose is input, KNN will identify the gestures from the training set that are closest in shape and position to the new one, and classify it accordingly. A simple generic visualisation can be seen in Figure 6. In this example the new data point would

be categorised as Class A for k = 5.

KNN is often used in scenarios where interpretability, ease of training, and flexibility are important. Its main limitation lies in performance, especially with larger datasets, since it stores and compares all examples directly.

3.7 VOSK & Natural Language Processing

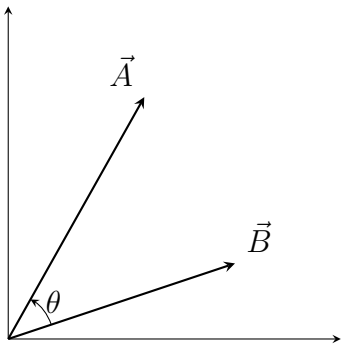
VOSK is an open-source speech recognition toolkit that allows for real time, offline transcription of spoken language. It supports a variety of languages and is designed to work on lightweight devices such as smartphones or embedded systems. Unlike cloud based alternatives, VOSK processes all data locally, making it a privacy friendly option for speech driven applications [Inc19].

Speech recognition is only one part of the puzzle when it comes to understanding language. Natural Language Processing (NLP) refers to a broad set of techniques that allow computers to analyze, interpret, and generate human language. In the context of voice control, NLP is used to go beyond simple word detection and understand the actual meaning or intent behind a sentence.

For example, phrases like "could you take off", "please lift off", or "start flying" might be worded differently but share the same intention. NLP techniques make it possible to interpret these kinds of variations by converting text into formats that are more meaningful to machines.

3.7.1 Cosine Similarity

Cosine similarity is a mathematical technique used to compare two vectors by measuring the angle between them (see Figure 7). It is commonly used in NLP to evaluate how similar two pieces of text are, regardless of their length or specific words used.



$$\text{cosine\_similarity} = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

Figure 7: Cosine similarity between two sentence embedding vectors. A smaller angle θ implies greater semantic similarity.

In a typical application, sentences are first converted into semantic vectors using a model like Sentence Transformers [RG19a]. These vectors represent the semantic meaning of the text. Cosine similarity then compares the direction of the vectors: a score close to 1 means the texts are very similar, while a score close to -1 means they are completely opposite in meaning.

This approach is useful when analyzing freeform speech or text, as it can identify the underlying meaning even when different words or phrases are used. It is particularly effective for applications like intent recognition, document clustering, or search engines.

4 Methodology

4.1 Design-Based Iterative Approach

This project followed a design-based research methodology, where the system was developed through multiple iterations and gradual refinement. Instead of designing the final system from the start, a series of prototypes were created, tested, and improved upon based on performance and usability.

The first prototype featured a simple voice interface with hardcoded commands. It relied on direct string comparisons and allowed only fixed phrases like "takeoff" or "land." This initial version showed the possibility of voice control, but lacked flexibility.

In the second iteration, gesture control was introduced using hand landmark data. However, this version used basic pattern matching logic and struggled with reliability due to the lack of trained models. From this stage onward, both gesture and voice pipelines were continuously improved.

Later prototypes incorporated a trained K-Nearest Neighbors model for gesture classification, and a sentence similarity system for more robust voice command understanding. Each iteration introduced more realistic control, better error handling, and increased safety mechanisms, such as cooldown timers and obstacle detection.

This iterative design allowed the system to evolve through experimentation and feedback. The final system combines voice and gesture recognition with safety constraints and supports a range of commands, offering a more natural interaction model for drone control.

4.2 System Overview

This project required both software and hardware components to function. Before any control logic could be implemented, the drone itself had to be assembled. This included mounting the motors, attaching the propellers and battery and connecting expansion decks such as the FlowDeck and MultiRanger. This was done quite easily, though extra attention is needed when attaching the propellers as there are two kinds of propellers, one being clockwise and the other one being a counterclockwise propeller.

Once the Crazyflie was built, the installation of the Crazyflie client software was the next step.

This client can be used to connect to the Crazyflie via the Crazyradio 2.0 to control the Crazyflie with a compatible game controller. While the official documentation states the use of a Playstation 3 and Xbox 360 controller [Bit25c], a Playstation 4 controller was used, which also functioned correctly to control the Crazyflie.

4.3 Flight Stabilization

Before executing any movement commands, the system performs a stabilization check using the Crazyflie's onboard Kalman filter. A logging configuration is set up to continuously monitor both the vertical position estimate (`kalman.stateZ`) and the roll angle from the onboard stabilizer. The drone is only considered ready for flight once the altitude estimate stabilizes and the roll deviation stays within a safe threshold (in this project set to 1.5) for several consecutive readings.

This check helps ensure that the estimator has properly converged and that the drone is physically upright and stable before initiating flight. Without this step, issuing high level movement or take off commands could result in erratic behaviour, such as uncontrolled drift or sudden tilting, especially in cases where the estimator hasn't locked onto the correct position yet. Essentially, this acts as a basic safety gate that blocks flight commands until the Crazyflie is in a trustworthy state.
In addition to this preflight check, a custom takeoff routine was created to replace the default `take_off()` function from the HighLevelCommander. Rather than issuing a single takeoff command, the system gradually increases the altitude in small, incremental steps, giving the drone time to stabilize after each lift. This staged ascent leads to a noticeably smoother and more controlled takeoff, reducing wobble and mid air corrections during initial lift.

However, even with these improvements, takeoff stability remains somewhat affected by the surface the drone is flying over. The FlowDeck, which handles optical flow and altitude estimation, performs best over matte surfaces. When flying over shiny, reflective, such as polished tiles or white desks, the optical flow sensor struggles to detect movement, leading to erratic hovering, misjudged height changes or drifting.

4.4 Gesture Command Pipeline

The gesture control pipeline is based on real time input from a laptop camera. For this project, the integrated webcam of a Samsung Book5 Pro 360 was used to capture the user's hand movements. These frames are processed using Google's MediaPipe Hands framework, which detects 21 landmark points per hand. Each landmark contains three coordinates (x, y, z), and since the system supports detection of both hands, a single frame can yield a total of 126 numerical features (21 points $\times$ 3 coordinates $\times$ 2 hands). To ensure consistent recognition regardless of distance or hand size, the raw coordinates are normalized by selecting a base landmark (the wrist) and scaling all points relative to a reference fingertip.

Before the system could recognise gestures, a dataset had to be created. This was done by recording multiple samples of hand gestures directly using the same MediaPipe pipeline. For each recognised gesture, such as "takeoff", "land", "forward", or "spin", dozens of frames were recorded and saved

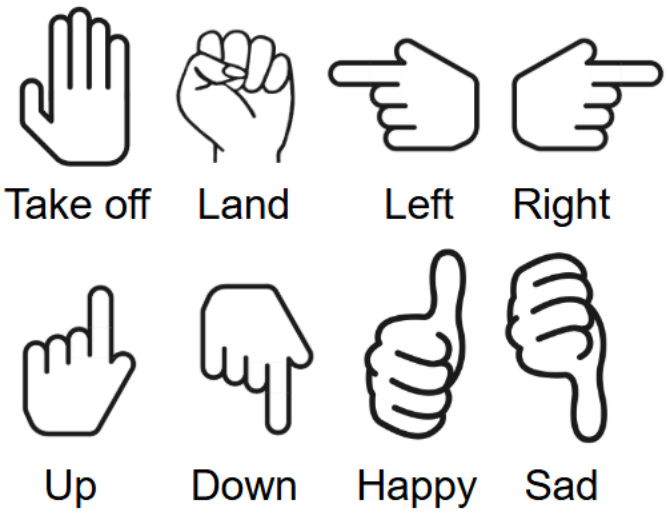


Figure 8: The implemented hand gestures, drawn using Autodraw [Goo23].

into a CSV file, with each row containing the 126 normalized landmark features and a label representing the intended gesture (see Figure 8). The labeled dataset was then used to train a K-Nearest Neighbors (KNN) classifier using the scikit-learn library.

KNN is a nonparametric classification algorithm that works by comparing the input sample to the stored training examples. It calculates the Euclidean distance between the input gesture vector and each sample in the dataset. The gesture is classified based on the most common label among the "k" nearest samples in this feature space. This approach is easy to retrain and extend, adding a new gesture simply involves collecting more labeled data and retraining the model.

During runtime, the camera continuously captures hand positions, and the extracted landmark vectors are passed into the trained KNN model to make a real time prediction. Once a gesture is recognised with high enough confidence, the corresponding drone command is executed. A cooldown timer is implemented to avoid unintentional rapid triggering of repeated commands and to provide smoother flight control. The gesture recognition process is illustrated in Figure 9.

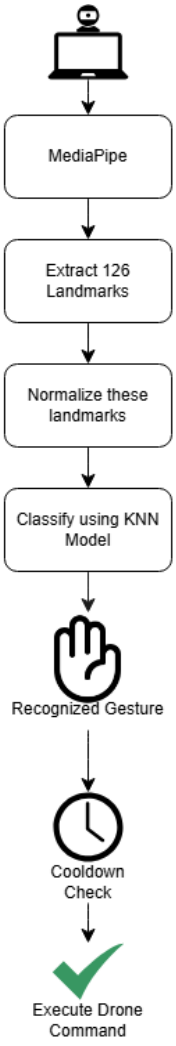


Figure 9: Gesture input pipeline

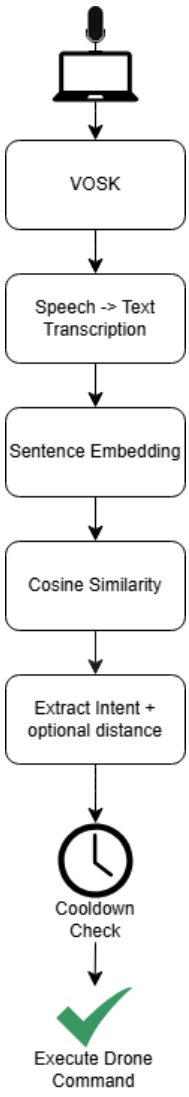


Figure 10: Voice input pipeline

4.5 Voice Command Pipeline

Voice input in this system is captured using the built in microphone of the used laptop or an external microphone connected to the laptop. For this project, the audio is processed in real time using VOSK, an open-source offline speech recognition engine. VOSK converts the audio stream into text, making it possible to understand user commands without relying on an internet connection.

Once the speech has been transcribed to plain text, the system needs to understand the meaning behind the user's words. Rather than matching commands through simple string comparison, a more flexible Natural Language Processing (NLP) approach is used. The spoken sentence is passed into a sentence embedding model. Specifically, the 'all-MiniLM-L6-v2' model from SentenceTransformers [RG19b]. This model encodes the sentence into a fixed length vector that captures semantic meaning, allowing similar sentences with different wording to produce similar vector representations.

To determine which command the user most likely intended, the sentence vector is compared to a pre-encoded list of example command vectors. Each command, such as “takeoff,” “land,” “go forward,” or “spin,” has several example phrases associated with it (e.g., “please take off,” “can you lift off,” etc.). The comparison is done using cosine similarity, a method that evaluates how close the angle between two vectors is. The higher the cosine similarity score (closer to 1), the more semantically similar the sentences are. If the similarity score between the user’s sentence and any of the predefined examples exceeds a defined threshold, the command is accepted and interpreted as the corresponding action.

This approach gives the system a level of language flexibility that traditional hardcoded command systems lack. For example, the phrases “can you please fly up one meter” and “rise by a meter” would both be interpreted as the same “go up” command. Additionally, the system scans the transcribed text for numerical values. If the user includes a distance modifier (e.g., “move forward two meters”), that number is extracted using regular expressions and used to scale the drone’s movement. If no number is found in the sentence, a default movement value is applied instead (in this case 0.3 meters).

To prevent misinterpretation or frequent command execution, a cooldown period is introduced after each recognised command. This ensures that repeated voice triggers don’t result in uncontrolled behaviour. Throughout this pipeline, debug statements are printed to the console, including what the system heard and which command it thinks was run, allowing the user to understand how their input is being processed in real time. An overview of the voice input pipeline is shown in Figure 10.

4.6 Trick learning

An additional feature developed in the final prototype is the ability for the drone to learn new ”tricks” or command sequences during runtime. This allows users to define and teach the drone a custom sequence of actions using their voice.

The trick learning process begins with the user activating learning mode via the voice command “learn a new trick.” The system then prompts the user to provide a name for the new trick. Once named, the user can perform a sequence of voice commands (such as “takeoff,” “spin,” “land”) that will be stored in memory as the steps for that custom trick. The process concludes with the voice command “end trick,” which saves the sequence to a temporary in-memory structure.

The system stores the trick name along with the sequence of associated commands in a dictionary, and dynamically updates the NLP embedding model used for intent recognition. This allows the newly learned trick name to be matched as a valid command in future interactions during the same session. Each time the user calls the custom trick (e.g., “do the happy spin”), the drone executes each saved step in sequence, giving the impression of a choreographed behaviour.

This feature transforms the Crazyflie from a reactive drone to a programmable, interactive companion that can learn and repeat custom behaviours. It showcases not only flexibility in control, but also a form of personalization that strengthens the pet like experience for the user.

4.7 Drone Communication

The Crazyflie 2.1 quadrotor is wirelessly controlled through the use of the Crazyradio PA, a USB radio dongle that establishes a real time link between the host computer and the drone. This radio connection allows for low latency communication, which is essential for both flight stability and timely command execution. The system transmits commands over a specific radio channel, and the drone is identified via a unique URI (`radio://0/80/2M`).

To issue movement commands, the system makes use of the Crazyflie's HighLevelCommander interface. This interface provides higher level movement instructions such as position based navigation, where the drone is instructed to fly to a specific (x, y, z) coordinate with a given yaw angle. All position data is managed relative to the drone's current estimated location, which is continuously tracked via the FlowDeck.
Movement is carried out using the `go_to` function, which sends a command to the drone to fly to a new 3D coordinate. To maintain safety and stability, each movement is small and incrementally updated based on user input. The system includes a local position tracker to store and update the drone's estimated location after each command.

Beyond basic command execution, the drone is also designed to behave in a more lifelike manner. The system introduces a pet like interaction model, where the Crazyflie responds to certain moods based on user engagement. If the drone receives frequent commands, it maintains an "active" or "happy" state. However, if it does not receive any gesture or voice input for an extended period of time, it gradually transitions into "bored" or "sad" moods. These moods are expressed through movement animations: for example, in a bored state, the drone may rotate gently left and right in place; in a sad state, it may dip slightly downward and then hover quietly.
This idle mood logic is implemented using a timer loop that periodically checks the time elapsed since the last user interaction. If the user resumes communication (by issuing a command), the drone returns to its neutral or happy state. This behaviour adds an element of personality to the Crazyflie, adding to the idea of it being more than just a machine.

4.8 Safety, Stability and Environment Awareness

Before executing any commands, the system ensures that the onboard position estimator is stable. This is crucial, as all subsequent movements rely on accurate and reliable position data. The FlowDeck, which combines an optical flow sensor with a height sensor, provides position estimation by tracking movement across surfaces. At startup, the software waits for the estimator to stabilize by logging the vertical position (`kalman.stateZ`) and roll angle to confirm that the drone is level and hovering stably.

To enhance environmental awareness and indoor safety, the drone is equipped with the MultiRanger deck. This deck provides real time distance measurements in five directions: front, back, left, right and up. Before executing any directional movement, the system queries the MultiRanger data to check whether the path is clear. If an obstacle is detected within a predefined minimum distance (e.g., 300 mm), the movement in that direction is blocked and the command is ignored. This ensures that the drone does not collide with walls, furniture, or people during indoor operation.

In addition to environmental safety, an emergency stop mechanism is implemented. This can be triggered at any time through either a specific voice command or gesture. When activated, the drone performs an immediate and controlled landing, bringing itself to a zero altitude state and disabling further movement commands until the system is manually reset.

Together, the layered communication system, real time environmental sensing, idle mood system, and emergency failsafes contribute to a safe and engaging user experience. The Crazyflie acts not only as a programmable drone, but also as an interactive and semi-autonomous companion.

5 Results

5.1 Gesture Recognition Accuracy

The performance of the gesture recognition system was evaluated by training a K-Nearest Neighbors (KNN) classifier on a dataset of labeled hand poses. Each gesture sample consisted of 126 normalized landmark features, extracted using the MediaPipe Hands framework. The model demonstrated high classification accuracy, particularly for gestures with distinct hand shapes and spatial configurations. Common commands such as *takeoff*, *land*, and *forward* were consistently detected with near-perfect accuracy, especially when performed clearly in front of the camera.

Overall, the trained model achieved an accuracy of 99.66% on the test set. Table 1 shows the precision, recall, and F1-score for each gesture class. Most classes achieved an F1-score of 1.00, with only minor variation in cases such as *left*, which occasionally overlapped with similar gestures like *right*.

Despite the high performance, recognition accuracy showed some sensitivity to environmental conditions. In low lighting, shadows could partially obscure hand landmarks, leading to misclassifications. Gestures performed too quickly or outside the center of the camera frame were also more likely to be missed. Additionally, hand poses with similar structural features, such as *left* and *right*, sometimes resulted in false positives due to their spatial resemblance.

These results suggest that while the system performs reliably under optimal conditions, minor environmental changes or user inconsistency can impact prediction quality. Table 1 summarizes the performance metrics:

Table 1: Classification performance of the gesture recognition model on the test set.

Gesture	**Precision**	**Recall**	**F1-Score**	**Support**
down	1.00	1.00	1.00	179
happy	1.00	0.99	1.00	122
land	1.00	0.99	1.00	140
left	0.98	1.00	0.99	97
right	1.00	0.99	1.00	158
sad	1.00	0.99	1.00	151
takeoff	0.99	1.00	1.00	172
up	0.99	1.00	1.00	154
Accuracy		**0.9966**		1173
Macro Avg	1.00	1.00	1.00	1173
Weighted Avg	1.00	1.00	1.00	1173

5.2 Voice Command Interpretation

The voice control system demonstrated consistent performance under optimal conditions. In a quiet environment, VOSK handled real time transcription with minimal delay and high accuracy. The integration of sentence embeddings allowed for flexible phrasing, meaning that users were not restricted to memorizing exact command sentences. For instance, both "fly forward" and "move ahead" were correctly interpreted as the same "forward" intent due to the semantic similarity measured via cosine similarity.

This flexibility improved the usability of the system, allowing users to speak naturally rather than staying to strict templates. However, in noisier settings or when the user mumbled or spoke too softly, recognition errors occurred more frequently. In some cases, VOSK would misinterpret the sentence or return an incomplete transcription. To mitigate false positives, a cosine similarity threshold was introduced to ensure that only semantically confident matches triggered drone commands. This filter proved effective in rejecting unintended inputs, although it occasionally ignored commands that were semantically close but fell below the similarity cutoff. Overall, the voice interface worked reliably as long as the user spoke clearly and environmental noise was minimal.

5.3 System Responsiveness

System responsiveness was a critical aspect of this multimodal drone interface. Once a gesture or voice command was detected and interpreted successfully, the drone typically responded within 1 to 2 seconds. This delay accounts for the processing time required to analyze camera input or audio, classify the intent, and communicate the command to the drone via the Crazyradio 2.0.

Drone movements were executed correctly. For example, when a command such as "go up one meter" was issued, the drone would ascend steadily and return to its current tracking point after the motion was completed. If no numerical value was included in a command, a default step size of 0.3 meters was applied, which proved sufficient for most indoor navigation tasks.

5.4 Safety and Obstacle Handling

Safety was also considered during system development, especially since the drone was flown indoors. The MultiRanger deck was used to provide real time distance readings in five cardinal directions: front, back, left, right and up. These sensors allowed the drone to assess its surroundings before moving. If an obstacle was detected within a defined safety margin—typically around 30–40 cm—the drone would cancel the intended movement in that direction and remain in place. This helped prevent collisions with walls or objects.

Downwards obstacle avoidance was not implemented, as the MultiRanger deck does not include a downward sensor. As a result, the drone still depended on the operator's awareness of ground clearance. Emergency landing was included as a failsafe and was triggered via either the "stop" voice command or the corresponding gesture. In all tested cases, this command resulted in a safe, controlled descent to zero altitude and a full stop of all commands.

5.5 Limitations and Observations

Despite the system's good performance, several limitations and technical challenges became apparent during testing and real-world use.

Gesture recognition was most accurate when the user's hand remained within the center of the webcam's field of view. If the hand drifted too far to the sides or moved too quickly, MediaPipe tracking would lose precision or fail entirely. Lighting conditions also played a major role. Dim environments or strong shadows reduced detection confidence, which increased the likelihood of misclassifications. Inconsistent or cluttered backgrounds occasionally introduced false positives as well.

The Flowdeck, while generally reliable, had a few notable shortcomings. When flying over shiny or reflective surfaces, the optical flow sensor would often return incorrect data, causing the drone to drift unpredictably or misjudge its altitude. In some cases, this led to unstable takeoffs or crashes. Following a crash, both the program and the drone had to be restarted to reestablish a working state. Additionally, the onboard Kalman filter would sometimes report a high roll angle at startup, preventing a clean takeoff altogether and thus the drone having to be restarted again. Even during normal operation, the drone occasionally failed to maintain a stable hover.

Battery life was another practical constraint. The standard Crazyflie battery allowed for roughly 5 minutes of flight time per charge, while recharging took between 45 minutes and 1 hour. A larger battery was tested and provided a few additional minutes of flight, but also required a longer charging time and increased the drone's overall weight. The effect of this larger battery on the drone's stability has not been tested in this research.

Space was also a consideration. While the Crazyflie is small, it still requires a sufficiently large and unobstructed area for initial takeoff. Small rooms or environments with obstacles come with a risk to both the drone and the surroundings.

On the voice recognition side, the system generally performed well with the lightweight VOSK model. However, a larger and more accurate VOSK model was also tested. While this improved transcription accuracy slightly, it introduced noticeable delays. Voice commands took longer to process and respond to, which affected the system's overall responsiveness and real time feel.

Finally, a small mention goes to the use of the wireless charging deck. While it is technically functional and can recharge the Crazyflie without requiring a physical cable, it proved to be largely impractical in this setup. The main issue is that the wireless charging deck occupies the same expansion slot as the Flowdeck, which is essential for flight stabilization and position estimation. Since only one deck can be used on the bottom port at a time, the wireless charger cannot be attached simultaneously with the Flowdeck. This means the drone has to be manually disassembled and reassembled each time to switch between flying and charging configurations, adding unnecessary steps to the development and testing process. As a result, while the idea of wireless charging is appealing in theory, especially for autonomous or long term applications, it was not a viable solution in this system's current design and workflow.

These limitations highlight both the technical and practical boundaries of the current system. While it is usable in controlled environments, improvements in robustness, hardware, and recovery handling will be necessary for deployment in less predictable settings.

6 Conclusion and Further Research

This thesis explored the Crazyflie system and some of its limitations as well as the development of a multimodal drone control system that allows users to interact with the Crazyflie 2.1 nano quadrotor using hand gestures and voice commands. The goal was to replace traditional joystick control with something more natural. Something that feels closer to how humans actually communicate. Through tools like MediaPipe for gesture recognition, VOSK for voice transcription, and machine learning techniques like K-Nearest Neighbors and sentence embeddings, the system was able to translate real time human input into drone behaviour.

This project also introduced pet like behaviour that made it feel more interactive. Idle animations and mood states gave the Crazyflie a sense of personality, also being able to teach the drone tricks, a sequence of commands, making the experience more engaging than a standard drone flight.

Technically, both gesture and voice control worked reliably under good conditions. The modular structure of the system made it easy to expand and debug. While there were still challenges, such as gesture sensitivity to lighting and background noise affecting voice recognition, the core idea was functional to use.

This project also laid the foundation for future work with drones at Leiden University. Since drone-based research hasn't been a big focus here yet, this system can serve as a starting point for new ideas, whether in robotics, HCI or AI. This project also shows the limitations of the system and useful insights of the expansion decks.

While there's still room to improve stability, scalability, and robustness, this thesis shows that it's absolutely possible to build a small, expressive drone system.

Future Work

There are several directions for continued development and improvement of the system:

First, the gesture recognition pipeline could be enhanced by replacing the current K-Nearest Neighbours classifier with a more advanced model, such as a neural network or transformer based architecture. This would likely improve classification accuracy and generalisation across different users. Expanding the dataset to include more diverse hand shapes, sizes, and motion styles would further increase robustness, making the system better suited for real world deployment.

Second, the voice interface could be extended to support custom command training, allowing users to define their own phrases and associate them with specific behaviours. This would introduce a layer of personalisation, particularly useful in shared or multi-user environments. Persisting these custom commands using a local JSON configuration or lightweight database would ensure that user preferences are retained across sessions.

Third, the current system depends on stable indoor conditions and well lit environments for accurate gesture recognition. Future iterations could explore the integration of depth sensing hardware or stereo vision to improve reliability in low light, cluttered, or more dynamic settings. This would allow for more consistent detection and expand the usability of the system beyond controlled environments.

Another promising direction involves fusing gesture and voice input in real time. For example, the system could interpret vague instructions such as "go there" by combining the spoken phrase with a pointing gesture, effectively resolving spatial references. This kind of multimodal fusion would enable a more natural and context aware interaction model.

In terms of navigation and autonomy, the drone's obstacle avoidance capabilities could be significantly improved. While the MultiRanger deck provides basic proximity sensing, integrating additional computer vision or SLAM (Simultaneous Localisation and Mapping) techniques would offer more advanced spatial awareness, enabling autonomous path planning and dynamic avoidance of moving obstacles.

To improve accessibility, a graphical user interface (GUI) or mobile companion app could be developed. This would allow users to visualise drone behaviour in real time, calibrate gestures, define and test new tricks, and manage custom voice commands. All without requiring direct interaction with code or command line tools.

Finally, Another valuable direction would be the introduction of adaptive learning and long-term memory within the system. By allowing the drone to incrementally update its gesture and voice command models based on user feedback or repeated usage, the interaction could become more personalised over time. For example, the system could learn subtle variations in a user's gestures

or preferred phrasing and refine its classification confidence accordingly. Such adaptability would make the system more resilient to variability and could also reduce the need for explicit retraining. Incorporating this form of online learning would move the drone closer to a truly interactive companion, capable of evolving alongside its user.

References

[AB25] Bitcraze AB. Crazyflie 2.1 product page. https://www.bitcraze.io/products/old-products/crazyflie-2-1/, 2025.

[Bit24] Bitcraze AB. Explaining kalman filters with the crazyflie. https://www.bitcraze.io/2024/01/explaining-kalman-filters-with-the-crazyflie/, 2024. (accessed: 19.02.2025).

[Bit25a] Bitcraze AB. Crazyradio 2.0. https://www.bitcraze.io/products/crazyradio-2-0/, 2025.

[Bit25b] Bitcraze AB. Flow deck v2. https://www.bitcraze.io/products/flow-deck-v2/, 2025.

[Bit25c] Bitcraze AB. Input-devices for the crazyflie client. https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/userguides/inputdevices/, 2025.

[Bit25d] Bitcraze AB. Multi-ranger deck. https://www.bitcraze.io/products/multi-ranger-deck/, 2025.

[CHO17] Yunho Choi, Inhwan Hwang, and Songhwai Oh. Wearable gesture control of agile micro quadrotors. In *2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2017.

[Dad18] Venkata Sireesha Dadi. Drone movement control using gesture recognition from wearable devices. 2018.

[GBV21] Nicolas Gio, Ross Brisco, and Tijana Vuletic. Control of a drone with body gestures. In *Proceedings of the International Conference on Engineering Design (ICED21)*, pages 761–770, Gothenburg, Sweden, 2021. Cambridge University Press.

[Gie17] Skwierczynski M. Witwicki W. Wronski P. Kozierski P. Giernacki, W. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. *2nd International Conference on Methods and Models in Automation and Robotics (MMAR).*, 2017.

[Goo19] Google. Mediapipe hands: Real-time hand tracking. https://ai.google.dev/edge/mediapipe/solutions/guide, 2019.

[Goo23] Google. AutoDraw, 2023. Accessed: 2025-06-22.

[Inc19] Alpha Cephei Inc. Vosk speech recognition toolkit. https://alphacephei.com/vosk/, 2019.

[Mad16] Meghana Gopabhat Madhusudhan. Control of crazyflie nano quadcopter using simulink. Master's thesis, California State University, Long Beach, 2016.

[MVW+21] Zdravko Marinov, Stanka Vasileva, Qing Wang, Constantin Seibold, Jiaming Zhang, and Rainer Stiefelhagen. Pose2drone: A skeleton-pose-based framework for human-drone interaction. *arXiv preprint arXiv:2105.13204*, 2021.

[RG19a] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019.

[RG19b] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. https://www.sbert.net/, 2019.