processes

MDPI

Article

Automatic Quadrotor Dispatch Missions Based on Air-Writing Gesture Recognition

Pu-Sheng Tsai [1], Ter-Feng Wu [2,*] and Yen-Chun Wang [2]

[1] Department of Electrical Engineering, Ming Chuan University, Taoyuan 333321, Taiwan; pusheng@mail.mcu.edu.tw
[2] Department of Electrical Engineering, National Ilan University, Yilan 260007, Taiwan; r1441014@ms.niu.edu.tw
* Correspondence: tfwu@niu.edu.tw

Abstract

This study develops an automatic dispatch system for quadrotor UAVs that integrates air-writing gesture recognition with a graphical user interface (GUI). The DJI RoboMaster quadrotor UAV (DJI, Shenzhen, China) was employed as the experimental platform, combined with an ESP32 microcontroller (Espressif Systems, Shanghai, China) and the RoboMaster SDK (version 3.0). On the Python (version 3.12.7) platform, a GUI was implemented using Tkinter (version 8.6), allowing users to input addresses or landmarks, which were then automatically converted into geographic coordinates and imported into Google Maps for route planning. The generated flight commands were transmitted to the UAV via a UDP socket, enabling remote autonomous flight. For gesture recognition, a Raspberry Pi integrated with the MediaPipe Hands module was used to capture 16 types of air-written flight commands in real time through a camera. The training samples were categorized into one-dimensional coordinates and two-dimensional images. In the one-dimensional case, X/Y axis coordinates were concatenated after data augmentation, interpolation, and normalization. In the two-dimensional case, three types of images were generated, namely font trajectory plots (T-plots), coordinate-axis plots (XY-plots), and composite plots combining the two (XYT-plots). To evaluate classification performance, several machine learning and deep learning architectures were employed, including a multi-layer perceptron (MLP), support vector machine (SVM), one-dimensional convolutional neural network (1D-CNN), and two-dimensional convolutional neural network (2D-CNN). The results demonstrated effective recognition accuracy across different models and sample formats, verifying the feasibility of the proposed air-writing trajectory framework for non-contact gesture-based UAV control. Furthermore, by combining gesture recognition with a GUI-based map planning interface, the system enhances the intuitiveness and convenience of UAV operation. Future extensions, such as incorporating aerial image object recognition, could extend the framework's applications to scenarios including forest disaster management, vehicle license plate recognition, and air pollution monitoring.

Keywords: MediaPipe; convolutional neural network (CNN); quadrotor UAV; flight mission dispatch; graphical user interface (GUI); air-writing; Gesture recognition

check for updates

Academic Editors: Xiaojun Jin, Jian Zhang and Dong-Qing Yuan

Received: 13 October 2025
Revised: 26 November 2025
Accepted: 1 December 2025
Published: 9 December 2025

Citation: Tsai, P.-S.; Wu, T.-F.; Wang, Y.-C. Automatic Quadrotor Dispatch Missions Based on Air-Writing Gesture Recognition. *Processes* **2025**, *13*, 3984. https://doi.org/10.3390/pr13123984

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Multirotor UAVs possess advantages such as compact size, high maneuverability, and deployment flexibility, making them widely applicable in fields including forest disaster

management, air pollution monitoring, environmental sensing, and logistics distribution. By equipping UAVs with cameras and streaming devices, and integrating 5G networks with real-time imaging technology, disaster-site footage can be rapidly transmitted to command centers to support immediate decision-making and dispatching, thereby reducing losses [1]. Incorporating solar cells extends flight endurance, enabling missions such as communication monitoring and land surveying [2]; with air quality sensors, UAVs can also collect environmental data such as PM2.5 concentration and temperature–humidity for air quality monitoring, enabling real-time environmental sensing in smart city applications [3]. In the context of e-commerce logistics, UAVs can overcome limitations of road networks and human labor, improving transportation efficiency and alleviating delivery bottlenecks [4]. At present, multirotor flight and aerial imaging technologies are relatively mature, and innovative concepts can readily generate novel products of practical value. However, most existing UAVs rely on joystick-based control, which presents steep learning curves, user limitations, and non-intuitive operation, making it difficult to satisfy the requirements of specialized applications and human–machine interaction. To address this challenge, this study proposes an automatic flight system that integrates air-writing gesture input with a graphical map-based interface for path planning, with the aim of enhancing the intuitiveness and convenience of UAV control while expanding its potential in intelligent control and applied domains.

The Tello (DJI, Shenzhen, China), developed by Ryze Technology in 2018 with technical support from DJI and Intel, is a compact quadrotor UAV equipped with a DJI flight control module and an Intel Movidius Myriad 2 vision processing unit. It features 720p HD video recording, electronic image stabilization (EIS), and multiple programmable flight modes, such as 360° rotation and bounce mode [5]. In 2019, Tello officially released SDK 1.3, enabling developers to control the UAV using languages such as Python and Swift. Through the UDP communication protocol, users could customize flight modes, establishing Tello as a significant platform for educational and developmental applications. The subsequent release of Tello EDU in 2020 introduced multi-drone formation flight, while the SDK 2.0 update in 2022 further enhanced AI capabilities, supporting applications such as OpenCV-based image recognition, autonomous obstacle avoidance, and gesture control [6]. Structurally, Tello adopts a quadrotor design with a lightweight body of only 80 g. It is powered by a 4.18 Wh lithium battery, allowing stable indoor hovering of approximately 13 min under windless conditions. The front-facing 5-megapixel camera supports 720p video recording with electronic stabilization, offering superior indoor positioning stability compared to GPS. The Wi-Fi video transmission range extends up to 100 m. Its key components and structure are illustrated in Figure 1, including: (1) propeller guard, (2) propeller, (3) motor, (4) micro-USB port, (5) status indicator, (6) antenna, (7) camera, (8) flight battery, (9) power button, and (10) vision positioning system. Figure 1 illustrates the structural components and specifications of the Tello UAV. Building on the foundation of the Tello platform, DJI later introduced the RoboMaster TT (Tello Talent), which incorporates additional hardware and SDK improvements, making it particularly suitable for AI-driven applications such as this study.

Pawlicki and Hulek [7] investigated autonomous localization of the Ryze Tello UAV using its onboard camera for AprilTag recognition. Their experiments demonstrated that both absolute and relative errors decreased significantly as the UAV approached the markers, confirming the method's high positioning accuracy and practical feasibility. Fahmizal et al. [8] developed a control interface for the Tello UAV on the Processing platform, integrating keyboard, joystick, and graphical user interface (GUI) controls. Their system ensured stable real-time feedback of flight status, validating the UAV's control and communication capabilities. Eamsaard and Boonsongsrikul [9] utilized the Tello EDU UAV as

a platform to design a human motion tracking system incorporating modules such as a control panel, tracking algorithm, alert notification, and communication extensions. Their system successfully tracked targets moving at 2 m/s within a 2–10 m range, achieving detection accuracies between 96.67–100%, which highlights its potential for search-and-rescue applications. Fakhrurroja et al. [10] employed YOLOv8 and optical character recognition (OCR) techniques with the DJI Tello UAV's onboard camera to perform automatic license plate recognition. Their results achieved a 100% success rate in license plate detection and 66% accuracy in character recognition, demonstrating high efficiency, low cost, and scalability, making the approach suitable for electronic ticketing and traffic management applications.

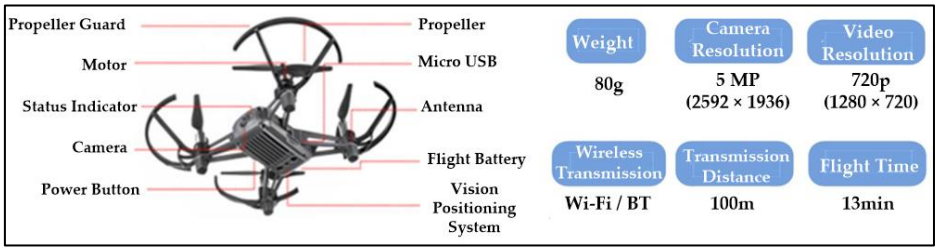


Figure 1. Tello quadrotor structure and specifications.

Although the Tello UAV provides lightweight portability and basic aerial imaging capabilities, its limited computational performance and restricted hardware expandability hinder support for advanced applications. To address these limitations, DJI introduced the RoboMaster TT (Tello Talent), which combines the Tello flight platform with an ESP32 control module, offering greater flexibility and openness. The RoboMaster TT is equipped with a programmable ESP32 microcontroller supporting MicroPython (version 1.17) development, and integrates peripherals such as a dot-matrix LED, ultrasonic module, and time-of-flight (TOF) distance sensor. These enhancements enable a wide range of tasks, including flight control, visual feedback, and environmental sensing. Unlike the original Tello, which could only be controlled via the SDK, RoboMaster TT allows direct control of the ESP32, enabling developers to design hybrid perception–control strategies and leverage wireless communication to improve data transmission and remote operation efficiency. For these reasons, this study adopts RoboMaster TT as the primary development platform, as it combines robust flight performance, expandability, and development flexibility, making it well-suited for gesture recognition, autonomous flight, and remote monitoring applications. In a related study, Iskandar et al. [11] implemented a hybrid gesture-tracking control system on the RoboMaster TT platform, integrating MediaPipe, OpenCV, and djitellopy. Their approach enabled real-time recognition and classification of visual gestures mapped to flight commands. Experimental results demonstrated the system's effectiveness in indoor environments, ensuring safe and intuitive operation while reducing the risk of UAV misbehavior. Thus, RoboMaster TT provides a solid foundation for developing the air-writing gesture recognition and automatic dispatch framework proposed in this study.

To enhance the intuitiveness of UAV operation and the flexibility of mission execution, this study integrates air-writing gesture recognition with map-based route planning, establishing an innovative human–machine interaction system. In this framework, air-writing gestures provide a contactless and intuitive control interface, while map-based planning supports the execution of automatic dispatch missions, forming a multi-modal and integrated flight control paradigm. While prior works on gesture recognition have primarily focused on general human–computer interaction or text entry applications, their integration with UAV control remains limited. This gap motivates the present study to explore air-writing–based gesture recognition as a novel and intuitive input method for

quadrotor UAV dispatch missions, thereby extending the potential of gesture-based interaction into intelligent aerial robotics. To better position this study within existing research, it is necessary to review the evolution of gesture recognition techniques and highlight how recent advances enable the proposed air-writing framework.

Early research on gesture recognition primarily relied on data gloves for information acquisition [12,13]. In recent years, however, image-processing approaches have become dominant, in which hand images are captured through cameras or sensors and then subjected to feature extraction and recognition processes. To further improve input intuitiveness and operational convenience, this study adopts the MediaPipe Hands module to construct an air-writing control system capable of real-time detection, tracking, and recording of hand trajectories in the air, which are subsequently used as training samples for deep learning models. In addition, specific gestures are designed as indicators of writing initiation and termination to prevent unintended motions from being incorporated into trajectory data.

MediaPipe, developed by Google, is a cross-platform inference framework designed to provide an integrated development environment for deep learning applications. The framework supports high-speed inference pipelines and real-time model visualization, enabling developers to rapidly build machine learning systems and deploy application prototypes. At present, MediaPipe offers more than 15 modular solutions, including object detection, face detection, hand detection (Figure 2), hair segmentation, face mesh, iris tracking, pose detection, and holistic full-body detection. In this study, the Hands module was employed to capture hand gestures in real time through a camera, extracting the three-dimensional coordinates of 21 hand landmarks including wrist, MCP, PIP, DIP, and fingertip points [14]. Among these, landmark #8 (index fingertip) was selected as the reference point for recording air-writing trajectories (Figure 2). In addition to recording the complete two-dimensional air-writing trajectory, the system also extracted coordinate variations along the X- and Y-axes to generate horizontal and vertical projection plots. Finally, the original trajectory and the two projection plots were combined into a standardized composite image consisting of three sub-figures, which served as input for the recognition models. This data processing strategy generates three types of input samples, including complete trajectories, X-axis projections, and Y-axis projections, which collectively enhance the accuracy of both gesture and digit recognition. Unlike conventional methods that depend on a single trajectory, the proposed approach provides greater flexibility and achieves higher recognition precision.

0 Wrist	1 Thumb CMC	2 Thumb MCP	3 Thumb IP
4 Thumb Tip	5 Index Finger MCP		
6 Index Finger PIP	7 Index Finger DIP		
8 Index Finger Tip	9 Middle Finger MCP		
10 Middle Finger MCP	11 Middle Finger DIP		
12 Middle Finger Tip	13 Ring Finger MCP		
14 Ring Finger MCP	15 Ring Finger DIP		
16 Ring Finger Tip	17 Little Finger MCP		
18 Little Finger MCP	19 Little Finger DIP	20 Little Finger Tip	

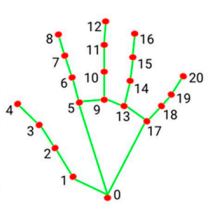


Figure 2. Hand position joint table.

Hsieh et al. [15] proposed a CNN-based air-writing recognition method that captured trajectories using a single camera with an efficient hand-tracking algorithm. Their approach required no delimiters or virtual frames, thereby increasing operational freedom. In addition, the continuous trajectories were transformed into CNN training images, and experimental results demonstrated high recognition accuracy with low model complexity.

Zan et al. [16] employed a standard camera to detect the three-dimensional trajectory of a colored pen, converting 24 consecutive points from the writing process into images, which were then classified using image segmentation and support vector machines (SVMs). Moriya et al. [17] developed a dual-camera system called the "Video Handwriting Board," which utilized color markers and stereo vision techniques to precisely estimate handwritten trajectory coordinates, enabling virtual handwriting input. Zhang et al. [18] introduced a real-time hand-tracking solution by integrating the MediaPipe framework with a two-stage model (a palm detector and a landmark model), which was capable of running on mobile GPUs in real time and was suitable for virtual and augmented reality applications. Harris et al. [14] combined MediaPipe with Kinect devices to create an interactive user guidance application that incorporated multiple gesture recognition functions with real-time visual feedback to enhance the interactivity of digital instructions. Huu et al. [19] integrated the MediaPipe algorithm with a long short-term memory (LSTM) model for healthcare monitoring and smart-device gesture control. Their system was capable of detecting the upper-body skeleton and recognizing nine common gestures with an average accuracy of 93.3%.

Collectively, these studies demonstrate that air-writing recognition technologies have advanced rapidly in recent years, particularly with the aid of hand-tracking and deep learning architectures. Improvements spanning trajectory recording, image preprocessing, and classification strategies have all contributed to high recognition accuracy and strong application potential. Among these, the MediaPipe framework has emerged as a fundamental tool for developing gesture recognition and air-writing trajectory systems due to its real-time efficiency and robust performance. On the other hand, the application value of UAVs in mission dispatch scenarios has also gained increasing attention [20], particularly in improving remote operation efficiency and maintaining mission continuity through energy-aware coordination. Building on these perspectives, the present study combines air-writing recognition with map-based route planning to design an integrated UAV control system, aiming to establish a more intuitive, versatile, and flexible mode of autonomous operation for intelligent monitoring and mission dispatch applications.

2. System Architecture

This section presents the overall architecture and functional modules of the air-writing recognition and mission dispatch control system developed in this study. The overall system architecture is illustrated in Figure 3. To realize an operation mode that integrates both gesture-based control and autonomous flight dispatch, the RoboMaster TT quadrotor UAV with expansion capability was selected as the development platform, complemented by an ESP32 control module and various peripheral sensors. The computational tasks (image processing and gesture recognition) are performed on the ground station Raspberry Pi 4 (Raspberry Pi Foundation, Cambridge, UK), while the ESP32 subordinate controller transmits commands to the onboard RoboMaster TT flight controller via Wi-Fi UDP (Port 8889).

The proposed system adopts a hierarchical control architecture consisting of a ground station and an onboard quadcopter controller. The ground station is implemented on a Raspberry Pi 4–based core platform together with a subordinate ESP32 controller, whereas the onboard unit corresponds to the RoboMaster TT flight control board. On the Raspberry Pi core platform, a graphical user interface (GUI) was implemented using Python Tkinter to provide an interactive control environment. The GUI integrates the MediaPipe framework to perform real-time air-writing gesture trajectory recognition, while also supporting Google Maps–based point selection and coordinate input for mission route planning. The MediaPipe framework performs real-time hand detection and feature extraction, and the recognized gestures are converted into Tello SDK flight commands, which are then

transmitted via UART serial communication to the subordinate ESP32 controller. The ESP32 decodes the received flight commands, converts them into the RoboMaster TT protocol format, and subsequently sends them to the quadcopter through its built-in Wi-Fi interface using the UDP communication protocol. In this communication hierarchy, the client-side UDP process is executed by the ESP32, whereas the server-side UDP process is handled by the onboard flight controller of the quadcopter.

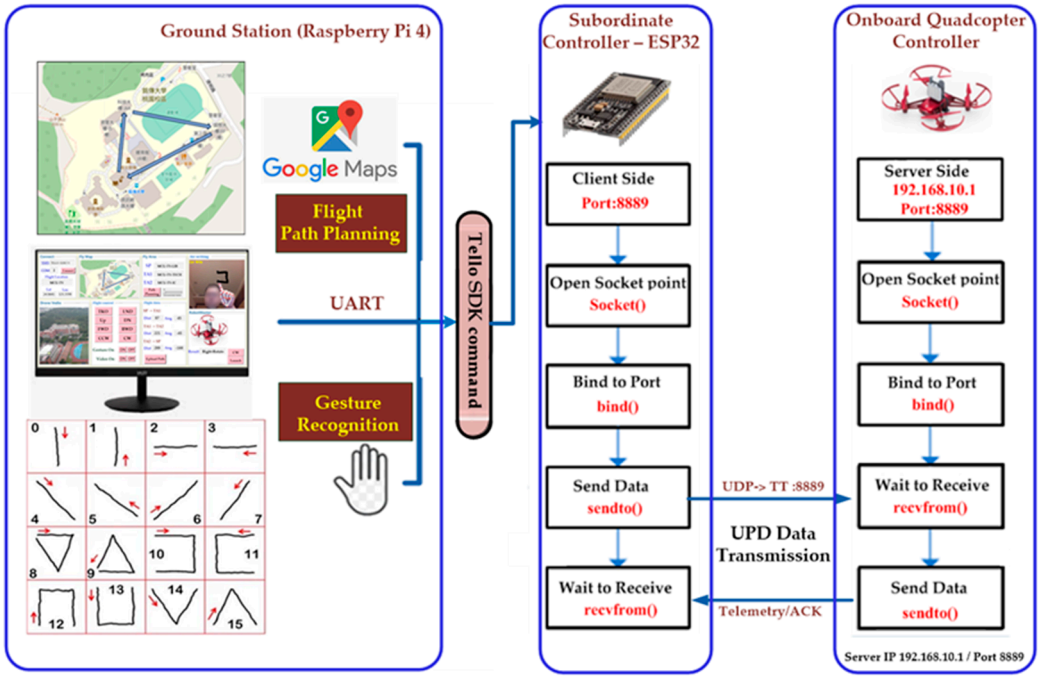


Figure 3. Interaction architecture between ground and onboard controllers. Note: The arrows indicate the drawing direction of each gesture, and the numbers (0–15) denote the gesture class labels.

Consequently, computationally intensive tasks—such as image processing, MediaPipe-based gesture recognition, and mission planning—are executed on the ground station, while the onboard unit focuses on low-latency command execution and flight control. This hierarchical design ensures efficient resource utilization and reliable command transmission in real-time flight operations.

2.1. Quadrotor UAV

The RoboMaster TT (Tello Talent) quadrotor UAV, developed by DJI, was selected as the aerial platform for this study. It is an upgraded educational version based on the Tello framework and retains the same flight control system, communication protocol, and 5 MP front-facing camera as the original model (see Figure 1 for Tello specifications). Compared with the standard Tello, the RoboMaster TT integrates a programmable ESP32 module, an 8×8 LED dot-matrix display, a time-of-flight (ToF) infrared distance sensor, and extended interfaces for additional sensors and wireless communication. The overall weight increases slightly from 80 g to 87 g, while the maximum flight time (13 min) and control distance (100 m) remain unchanged. These enhancements enable onboard data processing and visual feedback, making the RoboMaster TT particularly suitable for AI-driven research and rapid-prototyping applications requiring real-time command execution and mission control. As illustrated in Figure 4, the RoboMaster TT is equipped with a top ESP32 board supporting UDP communication via Wi-Fi, a Python-based SDK for programmable control, an 8×8 red/blue LED matrix, a time-of-flight (ToF) distance sensor with a range of approximately 1.2 m, and expansion ports (UART/I^2C) for peripheral integration.



Figure 4. Additional hardware modules of the RoboMaster TT.

2.2. Socket Communication Interface

In this study, control and feedback messages between the ground station and the RoboMaster TT were transmitted using the User Datagram Protocol (UDP). UDP was selected for its low latency and lightweight overhead, which are essential for real-time UAV control. The ground station, implemented on the Raspberry Pi 4, employed Python's socket library to generate flight commands from the GUI and transmit them via UART to the subordinate ESP32 controller. The ESP32 then established a UDP client connection through its built-in Wi-Fi interface, converted the received commands into the Tello SDK format, and forwarded them to the RoboMaster TT, which acted as the UDP server. Through this communication architecture, command and telemetry data were exchanged efficiently and reliably in real time, ensuring prompt flight responses and stable mission execution, as illustrated in Figure 3 (right).

2.3. Embedded Platform

To support both real-time UAV control and computationally intensive tasks such as gesture recognition and route planning, the system integrates two embedded platforms: the ESP32 microcontroller and the Raspberry Pi 4 microprocessor. Two ESP32 units are employed in the proposed system: an external ESP32 controller, which bridges UART and UDP communication between the Raspberry Pi 4 and the UAV, and an onboard ESP32 module integrated into the RoboMaster TT (top expansion board), which handles UDP reception, SDK command forwarding, and peripheral management. The Raspberry Pi 4 serves as the ground control unit, responsible for image acquisition, MediaPipe-based gesture recognition, GUI management, and mission route planning. Once a gesture command or waypoint is generated, the corresponding flight instruction is transmitted via UART to an external ESP32 module, which acts as a communication bridge. This external ESP32 converts the UART message into a UDP packet and sends it over Wi-Fi to the onboard ESP32 integrated within the RoboMaster TT. The onboard ESP32 interprets the received data according to the Tello SDK protocol, and forwards the corresponding motion commands to the UAV's flight controller.

Image processing and gesture recognition are executed entirely on the ground station, while the onboard Myriad 2 processor handles motion execution and sensor-based feedback control to maintain stable flight. The top ESP32 module on the RoboMaster TT additionally supports an LED matrix display, UART/I^{2}C expansion ports, and a ToF (Time-of-Flight) distance sensor, enabling peripheral integration and interactive feedback functions. This hierarchical division allows computationally intensive processes (image and gesture processing) to remain on the ground station, while time-critical flight control is performed onboard. The overall interaction between embedded modules and the flow of information are illustrated in Figure 5. The onboard flight-control core of the RoboMaster TT adopts an Intel Movidius Myriad 2 (Intel Corporation, Santa Clara, CA, USA) vision processing unit that incorporates an embedded Cortex-M4 (ARM Ltd., Cambridge, UK)

microcontroller for low-level stabilization, sensor fusion, and image-streaming tasks. This separation allows the ESP32 module to focus exclusively on network communication and SDK-level command execution.

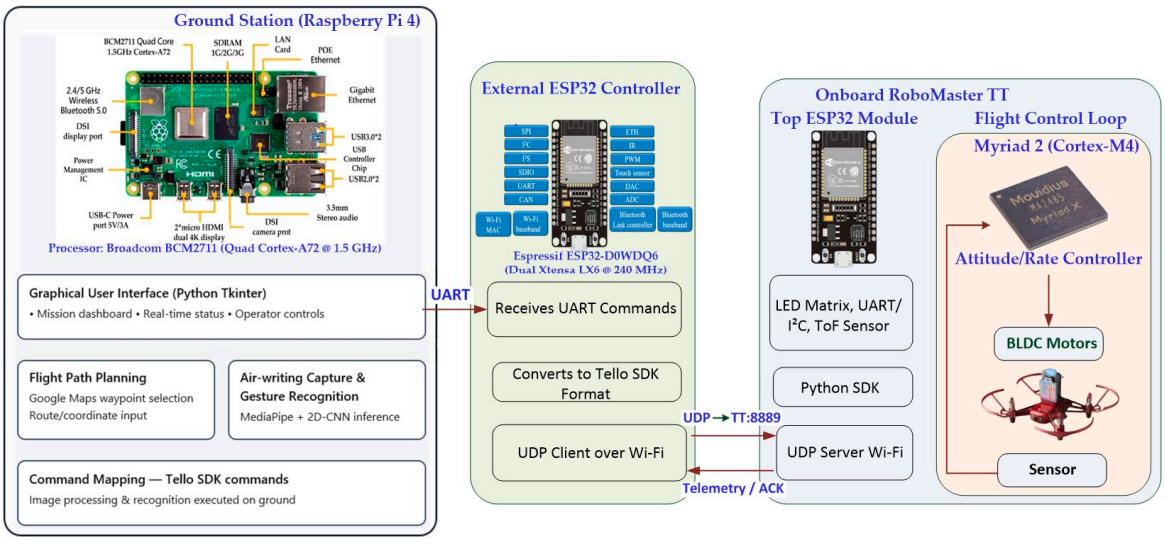


Figure 5. Schematic diagram of the embedded system interaction and data flow.

As illustrated in Figure 5, the overall system operates in an open-loop configuration at the ground control level, where gesture recognition and waypoint planning on the Raspberry Pi transmit flight commands to the UAV through the external ESP32 module. The GUI only monitors telemetry feedback without performing closed-loop correction. The onboard RoboMaster TT, however, contains a closed-loop flight control mechanism implemented by the Intel Movidius Myriad 2–based controller. It continuously regulates the brushless DC motors based on sensor feedback (e.g., attitude, rate, and ToF data) to maintain stable flight in response to external commands and environmental disturbances.

3. Map-Based Route Planning

3.1. Waypoint Coordinate Acquisition

Geocoding is the process of converting an address or landmark into geographic coordinates (latitude and longitude). For example, "Ming Chuan University" can be converted into latitude 24.9845° and longitude 121.3424°, thereby enabling its marking or localization on a map. Conversely, reverse geocoding transforms geographic coordinates into human-readable addresses or landmarks. In earlier studies [21], Google Maps API Geocoding services were commonly employed. In such implementations, when the user entered an address, the system accessed a JSON-based web page, from which the latitude and longitude information was extracted through web scraping. The specific workflow included: (a) constructing the target URL, (b) sending an HTTP GET request using the requests library, (c) obtaining the HTML response and parsing it with BeautifulSoup, (d) retrieving the required latitude and longitude data, (e) inputting the coordinates into folium.Map for map rendering, (f) marking the position with folium.Marker, and (g) exporting the map as an HTML file. However, because the Google Maps API Geocoding service is now subscription-based, this study instead adopts the built-in geocoding functions of folium.Map to obtain geographic coordinates from the input location, thereby achieving equivalent localization and annotation effects without incurring additional costs.

3.2. Great-Circle Distance and Bearing Angle Calculation

After obtaining the geographic coordinates (latitude and longitude) of the dispatch locations, it is necessary to compute the distance between any two points. Since latitude and longitude are expressed in spherical rather than Cartesian coordinates, the calculation must account for the shortest path along the Earth's surface, known as the great-circle distance. These great-circle distance and bearing formulas are derived from spherical trigonometry and are commonly adopted in geodesy and navigation applications [22–24]. Assuming the Earth is a sphere with radius R, let the coordinates of points A and B be (λ_1, ϕ_1) and (λ_2, ϕ_2), where λ represents longitude and ϕ represents latitude, and let $\Delta\lambda = \lambda_2 - \lambda_1$. The great-circle distance can then be computed as follows:

$$d = R \times \cos^{-1}(\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda) \tag{1}$$

where latitude and longitude values must be expressed in radians, and the Earth's radius is taken as $R = 6371$ km.

The initial bearing, which represents the heading direction from point A to point B along the shortest path (great-circle route), is calculated as:

$$\theta_1 = \tan^{-1}\left(\frac{\cos\phi_2 \sin\Delta\lambda}{\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda}\right) \tag{2}$$

The final bearing, which represents the heading direction when departing from point B to return to point A along the same great-circle path, is given by:

$$\theta_2 = \tan^{-1}\left(\frac{-\cos\phi_1 \sin\Delta\lambda}{\cos\phi_2 \sin\phi_1 - \sin\phi_2 \cos\phi_1 \cos\Delta\lambda}\right) \tag{3}$$

The final bearing, also expressed relative to true north, is generally not the reverse of the initial bearing due to the spherical geometry of the Earth. In this study, the great-circle distance and bearing angles between two geographic locations are computed using spherical geometry formulas, yielding the flight route distance together with the initial bearing and final bearing. This approach effectively accounts for the influence of Earth's curvature on heading calculations and provides precise directional and distance information for subsequent UAV route planning.

3.3. Conversion from Geodetic Coordinates to Local Planar ENU Coordinates

In UAV mission planning, waypoints are typically represented in geographic coordinates (latitude and longitude). However, directly using these angular values in route planning and distance calculations increases computational complexity. To facilitate flight data analysis and experimental validation, the geographic coordinates are transformed into a local Cartesian coordinate system. In this study, the East–North–Up (ENU) tangent plane coordinate system is adopted, with the Ming Chuan University library selected as the origin. Its geographic coordinates (ϕ_0, λ_0) are defined as $(x, y) = (0, 0)$. In this system, the eastward direction corresponds to the positive x-axis, the northward direction corresponds to the positive y-axis, and the upward (zenith) direction corresponds to the positive z-axis. For a given waypoint relative to the library, such as the Ming Chuan University Technology Building or the Ming Chuan University International Building, the geographic coordinates (ϕ, λ) are transformed into planar coordinates using the following formulas:

$$x = R \cdot \cos(\phi_0)(\lambda - \lambda_0),\ y = R \cdot (\phi - \varphi_0) \tag{4}$$

where $R = 6371$ km is the Earth's radius, ϕ and λ denote the latitude and longitude of the target point (expressed in radians), and ϕ_0 and λ_0 represent the latitude and longitude of the reference point (e.g., the library). Through this transformation, all waypoints can be projected onto a local planar system, allowing distances to be computed using the Euclidean distance formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5}$$

The distances between waypoints are then calculated for mission dispatch and flight path analysis. In this study, the Ming Chuan University library is defined as the origin of the ENU coordinate system, while the Ming Chuan University Technology Building and the Ming Chuan University International Building are transformed into the ENU tangent plane coordinate system. The resulting planar configuration forms a triangle consistent with their actual spatial arrangement on the map, ensuring the rationality of subsequent simulation and flight control experiments. It should be noted that this planar approximation is valid only for small-scale regions, where the Earth's curvature can be neglected.

4. Air-Writing Gesture Recognition

An air-writing system was developed using the Hands module of MediaPipe, which enables real-time detection, tracking, and recording of writing trajectories in the air. The system is built upon the 21 three-dimensional landmarks of the human hand (Figure 1), with the eighth landmark (index fingertip) serving as the primary reference for trajectory recording. To prevent unintended motions from being misclassified as valid trajectories, specific gestures were defined as triggers for the initiation and termination of writing. In addition to capturing the raw two-dimensional air-writing trajectories, the system generates three distinct image representations as recognition inputs: (i) a trajectory plot of the writing path (T-Plot), (ii) a set of horizontal and vertical projection plots derived from the X- and Y-axis coordinate variations (XY-Plot), and (iii) a composite image that integrates both projections with the original trajectory (XYT-Plot). These standardized image formats provide diversified training samples for subsequent recognition models. At the same time, the corresponding one-dimensional coordinate sequences are preserved in CSV format, ensuring that both visual and numerical features are available for model training and performance evaluation.

The recognition targets of this study consist of 16 types of air-writing gesture trajectories, with the stroke directions illustrated in Figure 6. Each gesture is mapped to a corresponding flight trajectory, as shown in Figure 7, and subsequently converted into executable RoboMaster TT SDK commands for actual UAV control. Through this mapping mechanism, the system enables air-writing gestures to be directly translated into UAV control commands, thereby achieving an intuitive and real-time mode of operation.

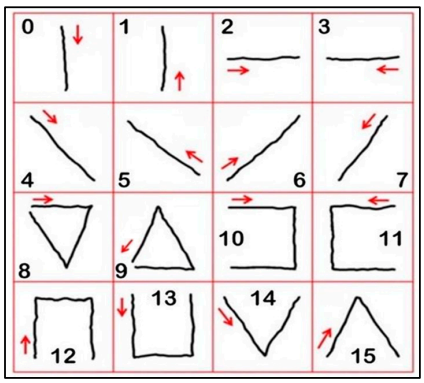


Figure 6. Air-writing gesture strokes.

Shape	Command	Shape	Command
↓	Land	↱	Right Circular Orbit
↑	Takeoff	↰	Left Circular Orbit
→	Move Right	↩	Rotate Right
←	Move Left	↪	Rotate Left
↘	Move Right - Backward	↓	Forward Path
↖	Move Left- Forward	↑	Backward Path
↗	Move Right- Forward	↘	Backward Return
↙	Move Left- Backward	↖	Forward Return

Figure 7. Gesture to flight trajectory mapping.

4.1. Writing System Operation Procedure

When the air-writing system is activated, a webcam window is opened to define the designated writing area. Once the system detects the user's hand, its position is highlighted using skeletal line rendering, as shown in Figure 8a. When the user performs the gesture illustrated in Figure 8b, the system initiates the trajectory acquisition process, with a solid black circle displayed at the fingertip to indicate activation. At this stage, the system begins recording the air-writing trajectory and simultaneously visualizes it within the window, as shown in Figure 8c. Upon completing the writing, the user simply performs the gesture shown in Figure 8d to terminate the procedure. The system then saves the recorded trajectory data as a CSV file and returns to the standby state.

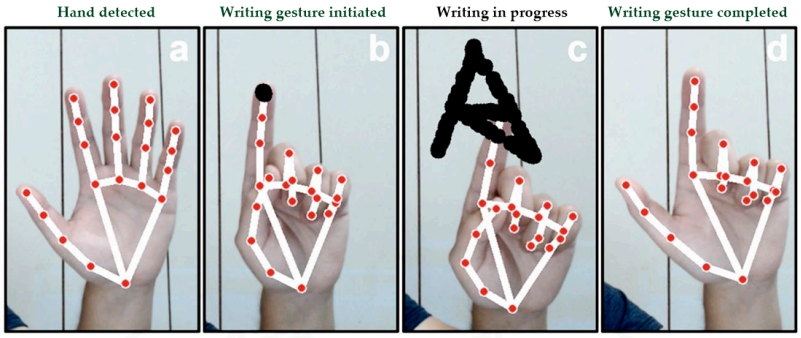


Figure 8. Sequential process of air-writing gesture recognition. (**a**) Hand detected; (**b**) Writing gesture initiated; (**c**) Writing in progress; (**d**) Writing gesture completed. Note: Red dots indicate detected hand keypoints, white lines represent skeletal connections, and the black trajectory shows the estimated air-writing path.

After completing the air-writing process, the system stores the continuous fingertip coordinates as a comma-separated values (CSV) file, with each record containing a timestamp, X coordinate, and Y coordinate. Each writing sample is thus represented in a tabular format with two rows (corresponding to the X- and Y-axis coordinates) and a variable number of columns (corresponding to data points), which are referred to in this study as the original samples (Original Data). To analyze the writing features and facilitate subsequent deep learning model training, the samples are visualized and transformed into three types of image representations: (a) X-axis projection trajectory, which illustrates horizontal variations of the gesture; (b) Y-axis projection trajectory, which represents vertical variations; and (c) complete trajectory plot, which reconstructs the two-dimensional writing path based on the X and Y coordinates. Finally, these three images are combined into a single composite image with fixed layout and normalized dimensions, ensuring a consistent input format

and improving recognition performance. Figure 9 shows the visualization of the original samples, including coordinate distribution plots and trajectory plots, corresponding to 16 air-writing gestures mapped to RoboMaster TT quadrotor flight commands.

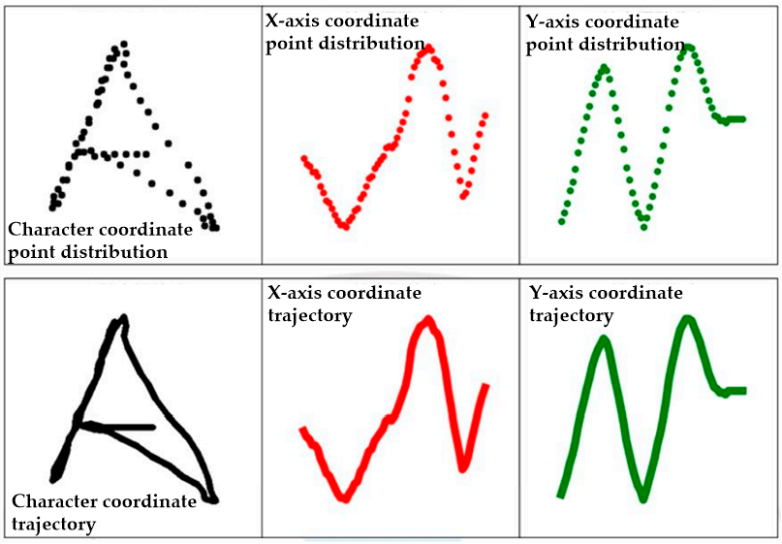


Figure 9. Coordinate distribution and air-writing trajectory.

Since no public dataset provides trajectory-based air-writing gestures with directional variations, a custom dataset was established in this study. Unlike existing gesture datasets (e.g., DHG-14/28 or ChaLearn) that contain coarse-grained motion classes, the proposed dataset focuses on fine-grained trajectory shapes with distinct temporal directions. As shown in Figure 6, several gesture pairs (e.g., (0, 1), (2, 3), (4, 5), and (6, 7)) share identical spatial shapes but differ in motion direction. These gestures can only be correctly distinguished when temporal information is included, highlighting the importance of the proposed XYT-based spatiotemporal representation. The proposed dataset and feature representation address the limitations of existing public gesture datasets, which lack directional trajectory variations essential for accurate air-writing recognition.

4.2. Data Augmentation

Data augmentation is a widely used technique to improve model generalization and reduce the risk of overfitting by applying diverse transformations to existing data, thereby generating new samples that remain consistent with the original data distribution. This effectively expands the dataset without requiring additional data collection, which is particularly beneficial in research scenarios where sample acquisition is difficult or limited.

In air-writing, the lack of a physical surface reduces stroke stability compared with conventional writing on paper. Moreover, the high sensitivity of the sensing hardware captures even slight hand tremors or unintentional movements, leading to skewed or jittered trajectories. To emulate these variations and enhance the diversity of training data, two augmentation strategies—coordinate translation and rotation—are applied. These transformations reproduce the jitter and slant naturally occurring during air-writing, thereby enriching the dataset and improving the robustness of the recognition model.

4.2.1. Coordinate Translation

Coordinate translation involves shifting the positions of trajectory points either horizontally or vertically, using fixed or random displacements, to simulate variations in gesture execution under different positions or conditions. From the trajectory sequence, anchor points are selected at fixed intervals, and displacements are applied to the anchor point

$p(n) = (x_n, y_n)$ and its neighboring points. The displacement decreases progressively with distance from the anchor point, thereby mimicking different levels of jitter. The general formulation is:

$$p'(n+k) = (x_{n+k} + \Delta x(k), y_{n+k} + \Delta y(k)), \ \ k \in \{-2, -1, 0, 1, 2\} \tag{6}$$

where $\Delta x(k)$ and $\Delta y(k)$ represent the horizontal and vertical displacement values, which can be fixed or randomized. The displacement is maximal at the anchor point and decreases outward, simulating horizontal, vertical, or diagonal jitter (e.g., ± 5 units at the anchor point, ± 4 units for adjacent points, and ± 3 units for points two steps away).

4.2.2. Coordinate Rotation

The slanting of characters during air-writing is simulated using a rotation matrix, which applies coordinate transformation to alter the overall orientation of the trajectory while preserving the relative spatial relationships and proportions among strokes. This operation emulates the tilt that may arise from variations in wrist angle or instability during free-space writing. Given a trajectory consisting of points (x_i, y_i), $i = 1, 2, \ldots, N$, the centroid (x_c, y_c) is selected as the rotation center, defined as:

$$x_c = \frac{1}{N}\sum x_i, \ y_c = \frac{1}{N}\sum y_i \tag{7}$$

The trajectory points are then rotated about the centroid using the following transformation:

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_i - x_c \\ y_i - y_c \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{8}$$

By adjusting the sign of the rotation angle θ, clockwise or counterclockwise transformations can be performed. This method enables the batch generation of samples with varying slant angles, thereby enhancing the model's robustness to rotational variations without altering stroke length or shape.

4.3. 1D Sample Preprocessing

Although variations in writing styles among users increase the diversity of sample features, differences in character size and stroke length (i.e., the number of coordinate points) complicate deep learning training. Therefore, this section focuses on one-dimensional (1D) preprocessing to standardize the data format for subsequent deep learning applications.

4.3.1. Interpolation

Inconsistencies in stroke length and the number of coordinate points hinder sample alignment and comparison during deep learning training. To address this, interpolation is applied to standardize stroke lengths and ensure smooth distribution of coordinate points. In this study, the interpolate function from the SciPy numerical library in Python is employed to generate new coordinates through linear interpolation based on the sequence of the original points. Each sample is resampled to 100 coordinate points, a setting that balances the preservation of shape characteristics with computational efficiency, thereby avoiding distortion caused by too few points or excessive computational load from too many. After this step, all samples achieve uniform length, as illustrated in Figure 10.

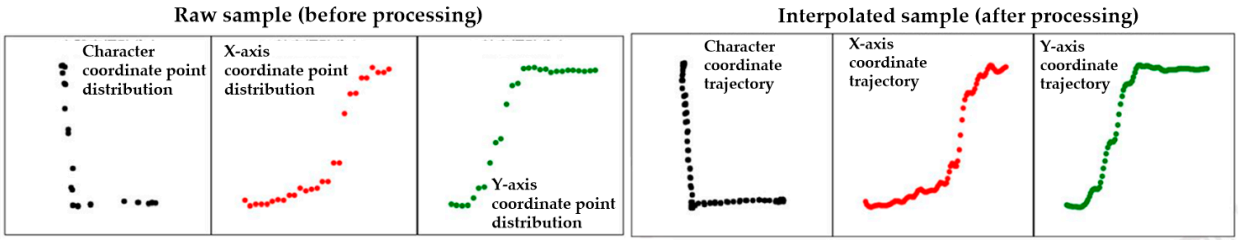


Figure 10. Comparison of results before and after sample interpolation.

4.3.2. Normalization

To ensure geometric consistency across all training samples, this study applies Min–Max normalization, which linearly maps coordinate points to the range $[0, 1]$. This method preserves the shape and proportions of strokes while eliminating numerical variations caused by differences in writing size and position. The normalization process is defined as:

$$\begin{cases} x_i^{nor} = \frac{x_i - x_{min}}{x_{max} - x_{min}} \\ y_i^{nor} = \frac{y_i - y_{min}}{y_{max} - y_{min}} \end{cases} \quad (9)$$

where x_i^{nor} and y_i^{nor} denote the normalized X and Y coordinates of the i-th point, respectively; x_{max}, x_{min} and y_{max}, y_{min} represent the maximum and minimum values of the sample along the X- and Y-axes; and x_i, y_i are the original coordinate values. As a result, all samples are scaled to the same geometric range, enabling consistent input for deep learning models.

4.3.3. Zero Padding

To prevent feature interference when concatenating the X- and Y-axis coordinates into a one-dimensional representation, this study inserts 14 zeros at both the beginning and the end of each axis sequence, thereby creating buffer zones without altering the original data structure. The X-axis coordinates are placed in the first segment, followed by the Y-axis coordinates, forming a fixed-length 1D data structure of size (256×1). This structure, defined as the 1D sample, can be directly used for training and testing with multi-layer perceptrons (MLPs), support vector machines (SVMs), and one-dimensional convolutional neural networks (1D-CNNs). The resulting 1D samples are illustrated in Figures 11 and 12.

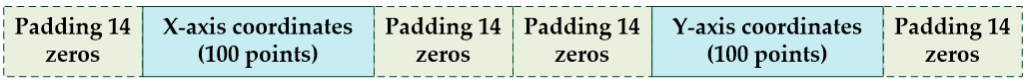


Figure 11. Structure of concatenated coordinate points.

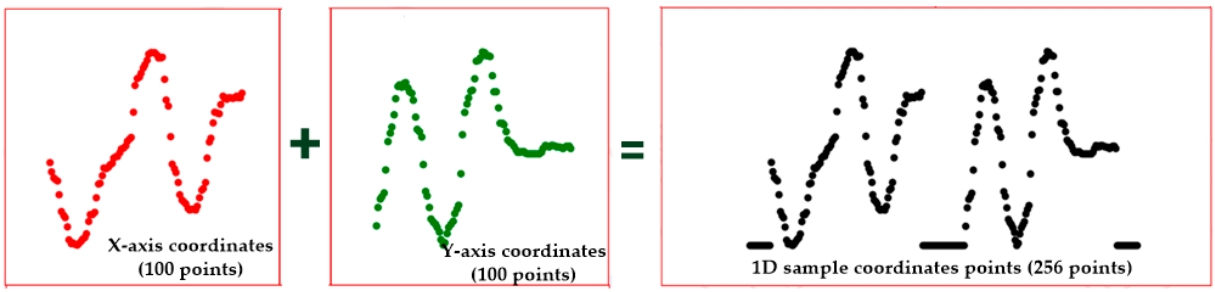


Figure 12. 1D sample generation from interpolated and normalized data.

4.4. 2D-Sample Preprocessing: Coordinate Alignment and Normalization

In air-writing, the same character may exhibit significant coordinate shifts due to differences in writing position. As illustrated in Figure 13a, both samples correspond to the letter "A", yet one is located in the upper-left region of the image window while the other appears in the lower-right, resulting in entirely different coordinate ranges along the X- and Y-axes. Such positional variations introduce structural discrepancies in the numerical representation of identical characters, which can hinder the model's ability to effectively learn from the data if left unaddressed. Therefore, this section focuses on preprocessing two-dimensional (2D) samples to eliminate coordinate shifts caused by writing position, ensuring that all samples are aligned to a unified reference for subsequent deep learning training.

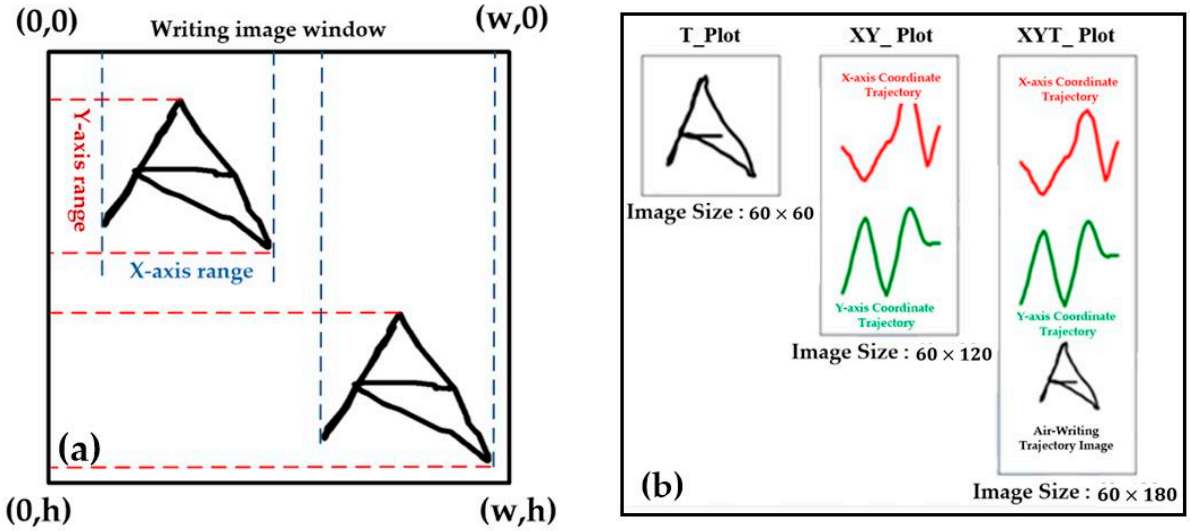


Figure 13. Writing position and coordinate features of air-writing gestures: (**a**) Writing position and coordinate range; (**b**) Coordinate feature plots of air-writing data.

To eliminate structural differences among samples caused by variations in writing position and character size, this study adopts a scaling and translation normalization method for two-dimensional preprocessing. Specifically, the scaling factor is determined based on the maximum stroke span along the X- and Y-axes, mapping the coordinates to a standardized scale. The centroid of the scaled trajectory is then computed and translated to the center of the image window. Through this procedure, all samples are aligned to a consistent two-dimensional coordinate reference, thereby preventing discrepancies in size and position from affecting model learning performance. First, the handwritten trajectory is enclosed within a bounding rectangle, whose side length S is defined as the maximum stroke span along the X- and Y-axes, as expressed in Equation (10). Here, x_{max}, x_{min}, y_{max} and y_{min} represent the maximum and minimum coordinate values of the sample along the X- and Y-axes, respectively. By computing the maximum span, the boundary range of the character in the two-dimensional coordinate system is obtained, which serves as the basis for subsequent scaling operations.

$$S = \max(x_{max} - x_{min},\ y_{max} - y_{min}) \tag{10}$$

To prevent the sample boundary from overlapping with the edges of the image window, a margin coefficient L is introduced, with L = 1.4. The scaling factor D is then cal-

culated by multiplying the bounding box side length S by L, and dividing the result by the target side length k (i.e., the size of the standardized square), as expressed in Equation (11).

$$D = \frac{S \times L}{k} \tag{11}$$

Using the scaling factor D, the original coordinates (x_i, y_i) are mapped to the new coordinates $\left(x_i^{new}, y_i^{new}\right)$, ensuring that all samples are rescaled to a consistent reference dimension.

$$x_i^{new} = \frac{x_i}{D},\ y_i^{new} = \frac{y_i}{D} \tag{12}$$

After applying Equations (10)–(12), all samples are rescaled to a uniform size. However, due to the inclusion of the margin coefficient, the positions of the samples within the image window may still be shifted. To address this, the centroid of the new coordinates $\left(x_{avg},\ y_{avg}\right)$ is further calculated.

$$x_{avg} = \frac{x_{max}^{new} + x_{min}^{new}}{2},\ y_{avg} = \frac{y_{max}^{new} + y_{min}^{new}}{2} \tag{13}$$

In this study, all character samples are uniformly mapped onto a square image window with a side length of w, where the window center is defined as $(w/2,\ w/2)$. Finally, the centroid of each sample is aligned with the window center to obtain the centered coordinates $\left(x_i',\ y_i'\right)$, as expressed in Equation (14).

$$x_i' = x_i^{new} + \left(\frac{w}{2} - x_{avg}\right),\ y_i' = y_i^{new} + \left(\frac{w}{2} - y_{avg}\right) \tag{14}$$

Through the above steps, all handwriting samples are standardized in both size and position, being uniformly rescaled and centered within the image window. This guarantees consistent geometric scaling across the dataset, enabling subsequent deep learning training to focus on the structural characteristics of the characters without being affected by variations in size or position.

In this study, the two-dimensional samples are categorized into three types: the Stroke Trajectory Plot (T-plot), the X–Y Coordinate Trajectory (XY-plot), and the Combined Plot (XYT-plot). The T-plot is generated directly from the character stroke trajectories, emphasizing the structural shape features of the handwriting. The XY-plot represents the stroke coordinates along the X- and Y-axes as time series and plots them together on the same image, highlighting the dynamic variations of coordinates over time. The XYT-plot integrates both the T-plot and XY-plot, thereby preserving the spatial trajectory of the strokes as well as their temporal characteristics, and providing a more comprehensive representation of the two-dimensional samples. The image dimensions of these three types of 2D samples are illustrated in Figure 13b. The preprocessed 1D and 2D samples establish a standardized dataset foundation, enabling consistent and fair evaluation across different classification models.

4.5. Classification Models for Gesture Recognition

To validate the applicability and generalization capability of the proposed method, three commonly used classification models were selected for performance comparison, covering both traditional machine learning and deep learning architectures. Artificial neural networks possess the ability to adjust parameters according to input data, enabling them to learn functional representations from large-scale samples and approximate or predict unknown inputs. Accordingly, this study employs a multilayer perceptron (MLP), a support vector machine (SVM), and convolutional neural networks (CNNs) to train and test four types of data representations: 1D samples, 2D-T plots, 2D-XY plots, and 2D-XYT

plots. The classification performance and cross-validation accuracy of these models are systematically compared to assess their effectiveness.

Specifically, the flight command gestures adopted in this study are illustrated in Figure 6, comprising 16 categories (indexed 0–15). In the case of 2D-T plots, only the overall trajectory shape of the strokes is preserved, without encoding directional information. As a result, certain gesture pairs such as 0 and 1, 2 and 3, 4 and 5, and 6 and 7 exhibit identical shapes in the 2D-T representation, differing solely in orientation. Consequently, this approach can distinguish at most 12 unique gestures. To address this limitation, the study further introduces 2D-XY plots and 2D-XYT plots, which incorporate temporal coordinate trajectories and multimodal feature fusion, respectively, thereby enriching directional information and enhancing the completeness and accuracy of gesture recognition.

4.5.1. Multilayer Perceptron (MLP)

MLP is a feedforward neural network composed of multiple fully connected layers, capable of automatically learning the nonlinear mapping between inputs and outputs through backpropagation [25]. In this study, the preprocessed two-dimensional samples were fed into the MLP model. The ReLU (Rectified Linear Unit) activation function was employed in the hidden layers to enhance nonlinear representation, while the Softmax function was applied in the output layer to accomplish multi-class classification. For the 2D-T experiments, the input layer consisted of 3600 features, corresponding to the flattened pixel values of a 60 × 60 grayscale image. The architecture contained two hidden layers: the first with 256 neurons and the second with 128 neurons, both activated by ReLU. The output layer comprised 16 neurons corresponding to the 16 gesture classes. The total number of trainable parameters was calculated as: 3600 × 256 + 256 + 256 × 128 + 128 + 128 × 16 + 16 = 956,816. Using 800 images for training and 49 images for testing in the 2D-T dataset, the model achieved a classification accuracy of 75%.

In the experiment with 2D-XY images, the input layer feature dimension increased to 21,600, corresponding to the flattened pixel values of 60 × 120 × 3 color images. The network structure remained the same as described above, with two hidden layers of 256 and 128 neurons activated by ReLU, and a Softmax output layer with 16 neurons. However, due to the larger input size, the total number of parameters increased substantially to: 21,600 × 256 + 256 + 256 × 128 + 128 + 128 × 16 + 16 = 5,564,816, which represents a substantial increase compared with the 2D-T case. Under the same training and testing dataset sizes, the classification accuracy was again 75%.

In the experiment with 2D-XYT images, the input layer feature dimension further increased to 32,400, corresponding to the flattened pixel values of 60 × 180 × 3 color images. The network structure remained consistent with the previous configurations, consisting of two hidden layers (256 and 128 neurons with ReLU activation) and a Softmax output layer with 16 neurons. The total number of parameters was calculated as 32,400 × 256 + 256 + 256 × 128 + 128 + 128 × 16 + 16 = 8,329,616, representing a substantial increase compared with both the 2D-T and 2D-XY cases. However, under the same training and testing dataset sizes, the classification accuracy remained at 75%.

To provide a clearer understanding of the network configurations, Figure 14 illustrates the detailed architecture of the multilayer perceptron (MLP) used for gesture classification. The input layer receives a flattened grayscale image of 60 × 60 pixels, represented as a one-dimensional feature vector $\mathbf{x} \in \mathbb{R}^{3600}$. The two hidden layers contain 256 and 128 neurons, respectively, both activated by ReLU functions. The output layer produces a 16-dimensional vector $\mathbf{y} \in \mathbb{R}^{16}$, corresponding to the 16 predefined gesture classes through the Softmax activation. During the testing phase, the MLP model was evaluated on 49 image samples, achieving a Top-1 accuracy of 75% across all three input types (2D-T,

2D-XY, and 2D-XYT). The model correctly classified most gestures but showed confusion among similar trajectories. For instance, vertical lines (Label = 1) were sometimes misclassified as another vertical line (Label = 0), slanted strokes (Label = 9) were mistaken for other slanted strokes (Label = 5), and "V"-shaped gestures (Label = 14) were occasionally recognized as "∧"-shaped gestures (Label = 15). These results demonstrate that, although all three input types yielded the same accuracy, the parameter count of the MLP model increased substantially with higher-dimensional inputs without corresponding performance improvements. This indicates that the MLP lacks scalability for high-dimensional representations, and therefore serves primarily as a baseline model. In subsequent sections, CNN-based approaches are introduced to highlight their advantages in feature extraction and classification performance.

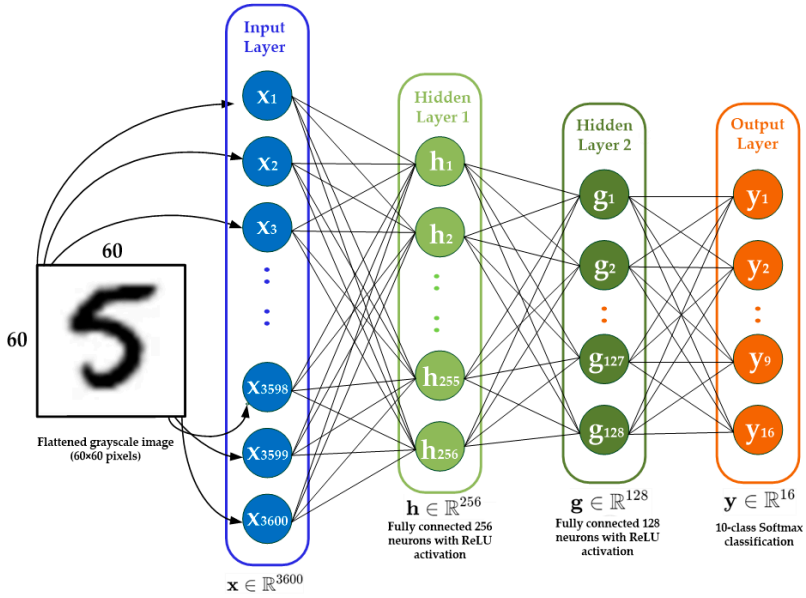


Figure 14. Structure of MLP used for gesture classification.

4.5.2. One-Dimensional Convolutional Neural Network (1D-CNN)

1D-CNN was employed to classify sequential gesture trajectories. In this approach, fingertip coordinates were concatenated, interpolated, and normalized to form sequences of length 256, which were then used as inputs to the model. By sliding convolutional kernels along the temporal axis, the 1D-CNN is able to capture local sequential patterns and extract discriminative features, while pooling operations reduce dimensionality and filter out redundant information. The final classification is performed by fully connected layers with Softmax activation. Compared with traditional feature engineering methods, the 1D-CNN provides a flexible and automated feature extraction capability.

The detailed architecture of the 1D-CNN is presented in Table 1. The model comprises two convolutional layers (Conv1D-1 with 16 filters and Conv1D-2 with 36 filters), each followed by max-pooling layers to progressively reduce sequence length. The extracted features are then flattened and passed through a dense layer of 128 neurons, followed by dropout regularization to mitigate overfitting. Finally, a dense output layer with 16 neurons maps the features to the corresponding gesture classes. Here, L denotes the input sequence length (256 in this study), and the output dimensions (e.g., L/2, L/4) indicate the reduced sequence length after pooling operations. The total number of trainable parameters is 300,436, which strikes a balance between model complexity and computational efficiency. Experimental results on 800 training and 49 testing samples demonstrated a classification

accuracy of 81%, confirming the effectiveness and generalization ability of the 1D-CNN for one-dimensional gesture trajectory recognition.

Table 1. 1D-CNN model parameters.

Layer Name	Output Dimension	#Params
Conv1d-1	(L,16)	416
MaxPooling-1	(L/2,16)	0
Conv1d-2	(L/2,36)	2916
MaxPooling-2	(L/4,36)	0
Flatten	((L/4) × 36)	0
Dense-1	(128)	295,040
Dropout	(L/4,36)	0
Dense-2	(16)	2064
Total	--	300,436

4.5.3. Two-Dimensional Convolutional Neural Network (2D-CNN)

2D-CNN is a deep learning architecture specifically designed for grid-structured data such as images. By leveraging the local connectivity of convolution and pooling layers, 2D-CNNs can automatically extract both local and spatial features from the input, avoiding the need for handcrafted feature design and effectively improving classification accuracy. In this study, the 2D-CNN was employed to process three types of input images—2D-T, 2D-XY, and 2D-XYT plots. Through multiple convolution and pooling layers, the network progressively compresses image information and enhances feature representation, before passing it to fully connected layers followed by a Softmax output layer to complete the 16-class gesture classification task.

The proposed architecture consists of two convolutional and pooling layers, with a dropout mechanism introduced between the convolutional block and the fully connected layer to mitigate overfitting. Here, (w,h) denotes the input image size, which varies depending on the data representation (60 × 60 for 2D-T, 60 × 120 for 2D-XY, and 60 × 180 for 2D-XYT). The first and second convolutional layers apply 16 and 36 filters with a 5 × 5 kernel size, respectively. After the first convolution, the feature map size remains (w,h), which is then reduced to (w/2,h/2) by the first pooling layer. The second convolution further processes the downsampled feature maps, followed by another pooling layer that reduces the resolution to (w/4,h/4). The output feature maps are then flattened and fed into a fully connected layer with 128 neurons, and finally classified by a Softmax layer with 16 output units.

The detailed output shapes and parameter counts of each layer are summarized in Table 2, where the total number of trainable parameters amounts to 1,053,844. This design ensures that the network can effectively extract hierarchical features from different types of input images while maintaining computational feasibility. The model was subsequently applied to all three input types to evaluate recognition performance under varying data representations.

The first experiment adopted 2D-T gesture images as inputs, with the overall network architecture illustrated in Figure 15. During training, the total number of parameters was 416 + 14,436 + 8100 × 128 + 128 + 128 × 16 + 16 = 1,053,844. A dataset of 800 training samples was used, where the preprocessed image features and corresponding labels were fed into the 2D-CNN for learning, followed by validation on 49 test images. The final classification accuracy reached 83%. In the second experiment, the inputs were replaced

with 2D-XY gesture images of size 60 × 120. The network structure remained the same as that used in the 2D-T experiment, but the total number of trainable parameters increased to 1216 + 14,436 + 16,200 × 128 + 128 + 128 × 16 + 16 = 2,091,444. With the same training and testing datasets, the classification accuracy improved to 87%. Finally, in the third experiment, the inputs were 2D-XYT gesture images with a size of 56 × 176. The network structure was identical to the previous two cases, with the total number of trainable parameters reaching 1216 + 14,436 + 24,300 × 128 + 128 + 128 × 16 + 16 = 3,128,244. Using 800 training samples and 49 test images, this configuration achieved the best performance, with a classification accuracy of 95%. For comparison, Figure 15 depicts the structure of the convolutional neural network (CNN) applied in the same task. The CNN accepts the same input image $\mathbf{x} \in \mathbb{R}^{60\times60\times1}$, and employs two convolutional layers with 16 and 36 filters of size 5 × 5, each followed by a 2 × 2 max-pooling operation. After flattening, the feature map of 15 × 15 × 36 is fed into a fully connected layer (128 neurons, ReLU) and a Softmax output layer (16 classes). The corresponding feature dimensions are annotated in Figure 15 for clarity.

Table 2. 2D-CNN model parameters.

Layer Name	Output Dimension	#Params
Conv2d-1	(w,h,16)	416
MaxPooling-1	(w/2,h/2,16)	0
Conv2d-2	(w/2,h/2,36)	14,436
MaxPooling-2	(w/4,h/4,36)	0
Flatten	((w/4) × (h/4) × 36)	0
Dense-1	(128)	1,036,928
Dropout	(128)	0
Dense-2	(16)	2064
Total	--	1,053,844

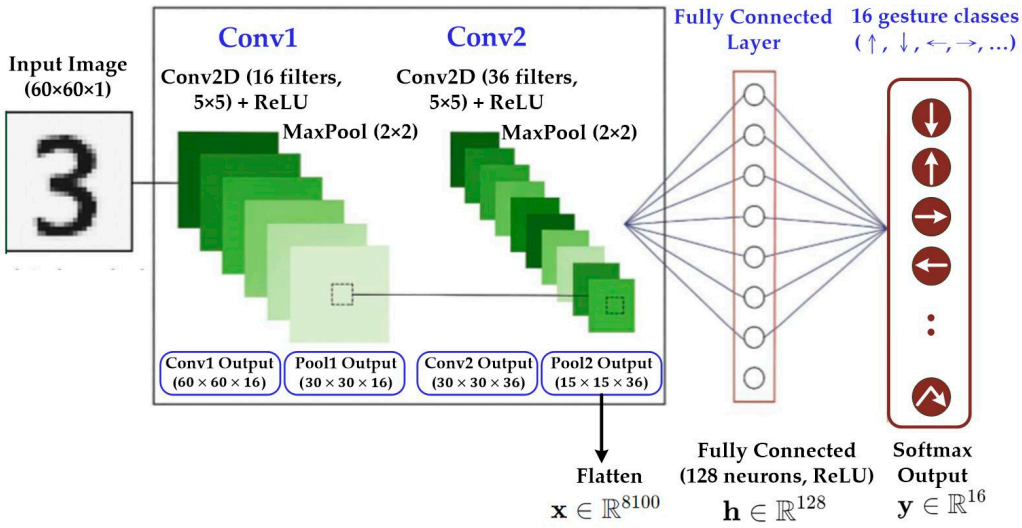


Figure 15. Structure of CNN used for gesture classification.

From the results of the three experiments, the recognition performance of the 2D-CNN model exhibited significant differences across different input image types. When trained with 2D-T images, the model relied solely on static trajectory representations derived from the gesture shape, without incorporating temporal or directional information. As a result, gestures that differ only in orientation—such as Label 0 vs. Label 1 or Label 2 vs. Label 3—were indistinguishable, and the model could effectively recognize at most 12 unique gestures. Consequently, the classification accuracy was limited to 83%. When the input was replaced with 2D-XY images, which preserved the variations of gestures in the planar coordinate space, the model was able to extract more discriminative features, thereby improving the classification accuracy to 87%. This demonstrates that the inclusion of spatial information in the input images effectively enhances the feature extraction capability of CNNs. Finally, with 2D-XYT images as inputs, both temporal and spatial features were integrated into a more comprehensive multimodal representation. As a result, the model learned richer and more discriminative features, achieving the highest accuracy of 95% among the three configurations. Overall, these comparisons confirm that the amount and quality of information contained in the input representation directly influence CNN performance. As the dimensionality and diversity of input features increase, so too does the recognition accuracy, thereby validating the effectiveness of multimodal feature fusion for gesture recognition.

4.5.4. Support Vector Machine (SVM)

SVM is a supervised learning algorithm based on statistical learning theory, typically regarded as a binary linear classifier. Its core concept is to handle cases where samples are not linearly separable in the original low-dimensional space by mapping them into a higher-dimensional feature space through kernel functions. In this transformed space, the SVM identifies an optimal separating hyperplane that divides the samples into different categories. The construction of the optimal hyperplane must satisfy two key conditions: (a) the hyperplane should correctly separate samples of different classes on opposite sides to ensure classification accuracy, and (b) the margin between the support vectors and the hyperplane must be maximized to enhance the robustness and generalization of the model. These conditions enable SVM to maintain stable performance even when dealing with unseen or noisy data. As illustrated in Figure 16, SVM leverages high-dimensional mapping to achieve linear separability, with the green plane representing the optimal separating hyperplane.

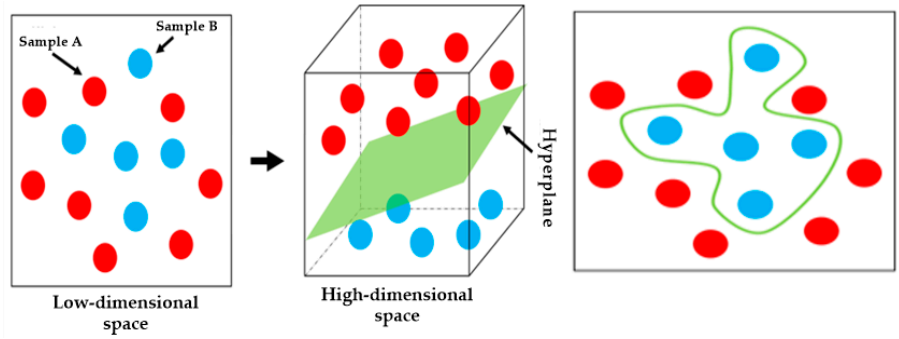


Figure 16. Conceptual illustration of SVM classification using kernel mapping.

SVM is particularly well-suited for gesture image classification problems characterized by small sample sizes, high dimensionality, and nonlinear features. When the data exhibit highly nonlinear distributions, the model employs kernel functions such as the radial basis function (RBF) to construct nonlinear decision boundaries that effectively separate different

classes. Cross-validation is used to optimize the penalty parameter (C) and kernel width (γ), ensuring the best classification performance. This design not only enhances the model's ability to handle nonlinear data but also prevents overfitting, thereby maintaining robust performance on the test dataset.

The SVM model in this study was implemented using the scikit-learn machine learning library. A linear kernel was adopted, with the penalty parameter (C) set to 0.9 and the γ parameter configured as auto. Recognition experiments were conducted on three types of gesture images: 2D-T, 2D-XY, and 2D-XYT. The overall classification performance was satisfactory, achieving accuracies of 89%, 87%, and 91%, respectively, for the three input types.

The experimental results show that the classification accuracies for the three gesture image types were 89% for 2D-T, 87% for 2D-XY, and 91% for 2D-XYT. Overall, all three configurations demonstrated strong classification performance, with differences primarily attributable to the richness of input features and their ability to represent spatial information. The 2D-XYT model achieved the highest accuracy because its input images integrated trajectory shapes, coordinate variations, and directional information, providing a more comprehensive feature representation that enabled the separating hyperplane to better distinguish between gesture classes. In contrast, 2D-T reflected only static trajectory features, while 2D-XY captured only coordinate variations, resulting in insufficient information to distinguish complex or similar gestures, and thus slightly lower accuracy. Furthermore, although SVM is inherently a linear classifier, the incorporation of kernel functions allows it to handle non-linear data distributions while maintaining robust generalization. Notably, the use of a linear kernel in this study achieved nearly 90% accuracy despite the limited dataset size, indicating strong adaptability and practical applicability to gesture image classification.

The performance of the three models on gesture recognition tasks exhibited notable differences, as summarized in Table 3. The MLP model achieved a consistent accuracy of 75% across different input representations, indicating limited capability in feature extraction and difficulty in capturing spatial relationships within gesture images. In contrast, the 2D-CNN model significantly improved classification performance by automatically learning both local and high-level features through convolution and pooling operations. Notably, the incorporation of multimodal features in the 2D-XYT input yielded an accuracy of 95%, highlighting the advantage of feature fusion in enhancing model robustness and adaptability to variations. Meanwhile, the SVM model demonstrated stable performance under small-sample and high-dimensional settings, achieving accuracies of 89%, 87%, and 91% for the three input types. Although slightly lower than CNN, SVM still provided reliable classification capability and generalization.

Table 3. Comparison of accuracy of different models in gesture recognition.

	MLP	**1D-CNN**	**2D-CNN**	**SVM**
2D_T	75%	—	83%	89%
2D_XY	75%	—	87%	87%
2D_XYT	75%	—	95%	91%
1D	—	81%	—	—

5. Experimental Results

5.1. RoboMaster TT Flight Command Testing

The ESP32 microcontroller, equipped with a built-in Wi-Fi module, was employed to establish a socket communication interface based on the User Datagram Protocol (UDP). A datagram transmission mode was adopted to enable the exchange of flight commands with the RoboMaster TT (Tello Talent) quadrotor UAV. The system architecture is illustrated in Figure 3, where the Tello UAV operates as the server and the ESP32 microcontroller functions as the client. Both sides first create communication endpoints using the socket() function, followed by binding the IP address and port number with bind(). Once the connection is successfully initialized, data exchange is conducted through the sendto() and recvfrom() functions, with each transmission explicitly specifying the target socket address to ensure correct delivery of commands and responses. After the UDP socket is established and initialized, the ESP32 sequentially transmits flight commands. The complete set of flight commands, as summarized in Table 4, serves as the foundation for subsequent flight control and experimental validation.

Table 4. ESP32 UDP and Tello SDK command set.

Description	Flight Command
ESP32 UDP	command
Takeoff/Land	takeoff/land
Forward/Backward (x = 20–500 cm)	forward/back x
Left/Right (x = 20–500 cm)	left/right x
Up/Down (x = 20–500 cm)	up/down x
Clockwise rotation angle (x =1–360°)	cw x
Counter-clockwise rotation angle (1 ~ 360°)	ccw x
Flip command (x = l/r/f/b/lb/rb/lf/rf)	flip x
Move to coordinate (x, y, z = −500 to +500 cm; v(speed) = 10–100 cm/s)	go x y z v

Note: The parameters x, y, z, and v represent the forward/backward, right/left, upward/downward directions, and linear velocity (cm/s), respectively. The command go x y z speed defines the three-dimensional displacement of the drone, where the position values range from −500 to +500 cm and the velocity ranges from 10 to 100 cm/s.

The Tello SDK provides a set of low-level flight control instructions (e.g., takeoff, land, cw x, ccw x, forward x, back x, left x, right x, up x, down x) that directly control the UAV's basic movements such as takeoff, landing, rotation, and translation. In contrast, this study defines 16 air-writing gesture commands (e.g., straight lines, diagonal lines, triangles, squares, and V-shapes) as high-level commands. Each gesture is first recognized by the system and then mapped to a corresponding flight mode, which is subsequently decomposed into a sequence of Tello SDK instructions for execution (Table 5). This hierarchical design allows users to intuitively control the UAV through gestures, without the need to manually input complex SDK commands. To further validate the system, a series of flight experiments were conducted using the RoboMaster TT quadrotor. Table 5 defines the 16 gesture commands and their corresponding drone actions and SDK command sequences. These demonstrations confirm that the proposed air-writing gesture recognition system can effectively translate human gestures into executable UAV flight commands, thereby realizing intuitive and seamless human–machine interaction.

Figure 17 illustrates the actual flight behavior of the quadrotor during vertical and horizontal maneuvers. The first and second images represent the UAV before and after ascending, respectively, while the third and fourth images depict the UAV before and after moving leftward. The flight tests confirm that the developed system can accurately execute

altitude and directional control commands through gesture recognition, maintaining stable flight performance throughout the transitions.

Table 5. Defined 16 gesture commands and corresponding drone actions.

No.	Gesture Shape	Corresponding Action	SDK Command Sequence
0	↓	Landing	***land***
1	↑	Takeoff	***takeoff***
2	→	Move Right (d)	***right d***
3	←	Move Left (d)	***left d***
4	↘	Move Backward-Right (a, b)	***go −a b 0 v***
5	↙	Move Backward-Left (a, b)	***go −a − b 0 v***
6	↗	Move Forward-Right (a, b)	***go a b 0 v***
7	↖	Move Forward-Left (a, b)	***go a − b 0 v***
8	⟀	Triangular Loop (CW)	***go a − b 0 v → go 0 2b 0 v → go − a − b 0 v***
9	⟁	Triangular Loop (CCW)	***go −a − b 0 v → go 0 2b 0 v → go a −b 0 v***
10	⟲	Rotate Clockwise (θ)	***cw θ***
11	⟳	Rotate Counterclockwise (θ)	***ccw θ***
12	⌐	Forward Rectangular Loop (a, b)	***forward a → leftb → back a***
13	⌙	Backward Rectangular Loop (a, b)	***back a → rightb → forward a***
14	⟋	Forward V-Loop (a, b)	***go − a b 0 v → go a b 0 v***
15	⟍	Backward V-Loop (a, b)	***go a b 0 v → go − a b 0 v***

Note: Bold italic formatting is used consistently to represent the SDK command sequences and their variable parameters (e.g., *a*, *b*, *v*) required by the RoboMaster TT flight controller.

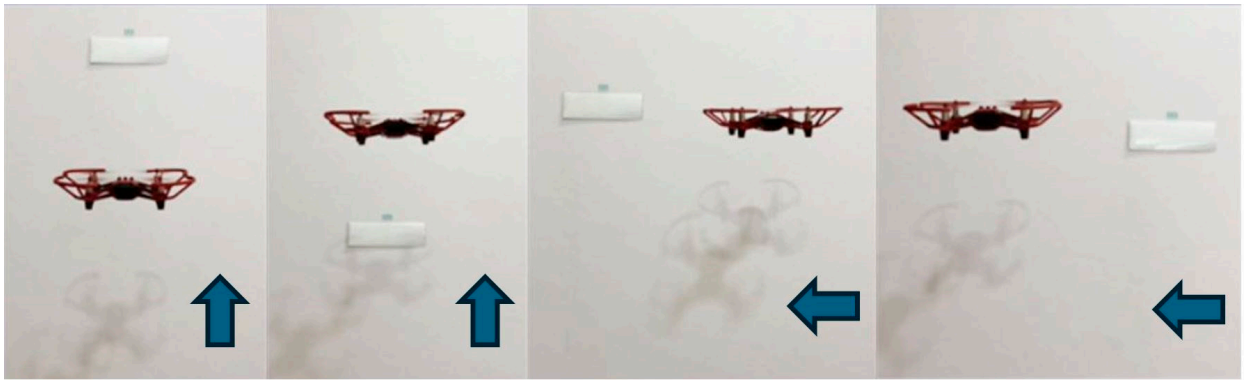


Figure 17. Quadrotor state before and after ascending and leftward motion. Note: The blue arrows indicate the movement direction of the drone.

5.2. GUI for Automated Dispatch System

The graphical user interface (GUI) for the automated dispatch system was developed in the Python environment using the built-in Tkinter package, as shown in Figure 18. The GUI integrates multiple modules for mission planning, connection management, real-time video, data monitoring, and gesture recognition, thereby providing a complete workflow for UAV dispatch and control. Tkinter functions were combined with additional libraries to implement interactive components such as control buttons, message displays, and video

windows. Through this interface, users can plan UAV dispatch routes, activate the onboard camera, and utilize folium.Map and MediaPipe on the Raspberry Pi 4 platform to perform path planning and air-writing gesture recognition. The system was validated using the 16 predefined flight command gestures.

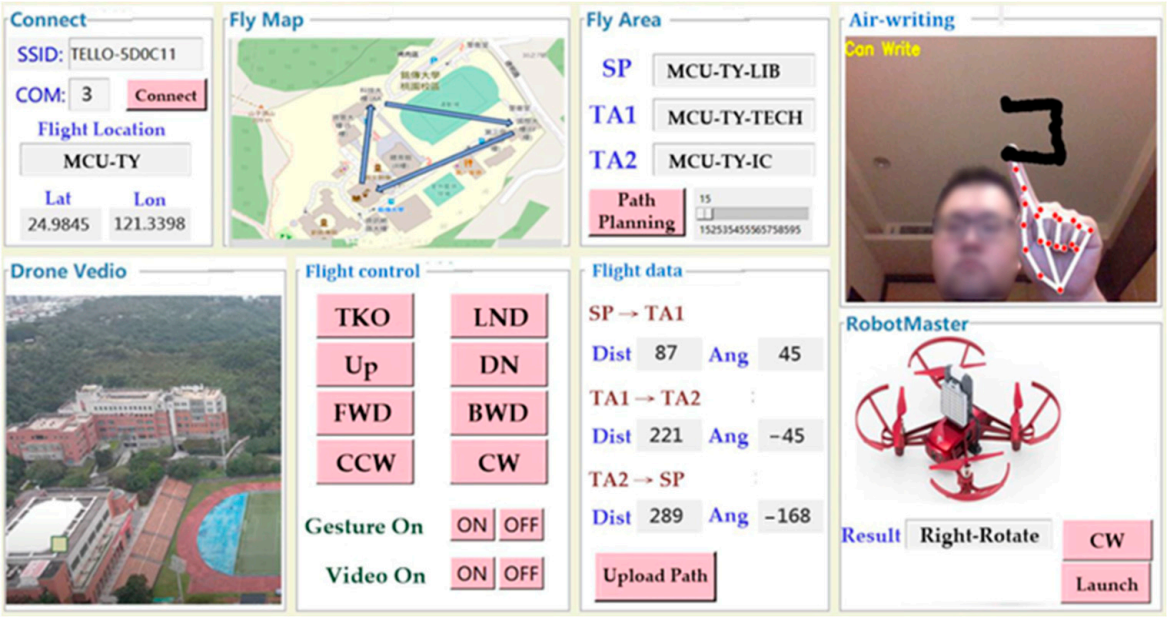


Figure 18. UAV dispatch GUI. Note: The red dots indicate detected keypoints, white lines represent skeletal connections, and the black curve shows the air-writing trajectory.

The overall GUI consists of seven functional modules: (1) RoboMaster Connection, which establishes communication with the RoboMaster platform and sets the initial location (e.g., Ming Chuan University, Taoyuan), automatically converted into geographic coordinates; (2) Mission Map, providing a map-based interface for selecting dispatch locations by clicking directly on the map; (3) Route Planning, where users specify the starting point and destinations for Mission A and Mission B, with the UAV sequentially flying from the starting point to Mission A and then to Mission B; (4) Gesture Recognition, which employs MediaPipe to detect air-writing trajectories and recognize the 16 predefined flight command gestures as control inputs; (5) Real-Time Video, displaying live video streams transmitted from the UAV during task execution; (6) Flight Control, supporting basic quadrotor operations such as takeoff, landing, rotation, and movement, serving as a baseline interface for testing and manual control; and (7) Flight Data, providing real-time UAV status and flight parameters for monitoring and analysis. This modular design not only delivers an intuitive and highly integrated operating environment but also supports mission planning, target detection, and experimental validation, forming the foundation for subsequent system demonstrations.

In the wireless communication connection section, the user is first required to input the SSID of the UAV's wireless access point, as shown in the upper-left field of Figure 18. Typically, the SSID of the RoboMaster TT access point begins with the prefix "TELLO". Once the connection is established, the current location of the UAV must also be specified. This location can be entered directly as a place name in Chinese (e.g., Ming Chuan University), which the system automatically converts into the corresponding latitude and longitude coordinates and displays in real time in the designated text window of this section.

The primary function of the map input and path planning module is to define the mission trajectory of the quadrotor UAV. In this study, the Taoyuan campus of Ming Chuan

University was selected as the experimental site, with the system's default starting position set to Ming Chuan University Library, as shown in Figure 18. Users can specify Mission A and Mission B by entering Ming Chuan University Technology Building and Ming Chuan University International Building, respectively. The system then generates a complete flight path in the order of Library → Technology Building → International Building → Library. Geographic coordinates (latitude and longitude) for each waypoint are obtained through geocoding, and further converted into a local East–North–Up (ENU) plane coordinate system using the library as the origin, as expressed in Equation (4). Figure 19 illustrates the latitude/longitude of the three waypoints, their transformed planar coordinates, and the resulting dispatch route. The consistency with the Google Maps result validates the feasibility of the proposed method. Meanwhile, the Mission Map panel is updated in real time to display the corresponding dispatch route, providing an intuitive visualization that enables the operator to clearly grasp the dispatch process and flight range, as illustrated in Figure 20.

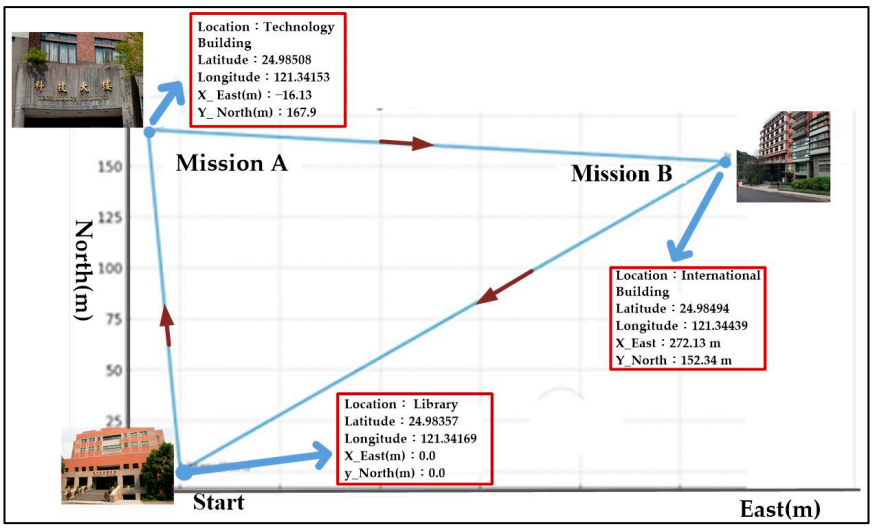


Figure 19. UAV flight path.

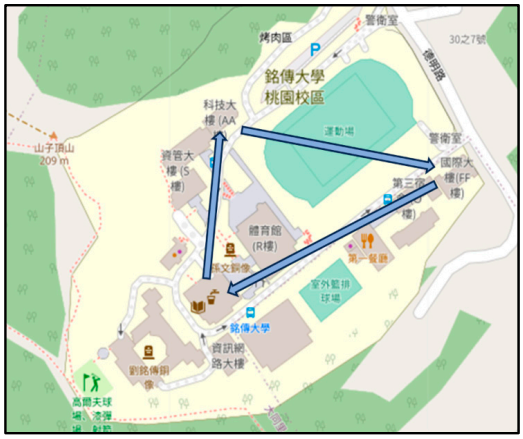


Figure 20. Dispatch map survey.

After completing the route planning, the system performs three subsequent operations. First, the interface panel is updated in real time to display the corresponding mission map, visualizing the planned flight path (Figure 20). Second, the system calculates flight parameters such as distances and azimuth angles between waypoints, which serve as references for dispatch control. Finally, to facilitate small-area testing in an indoor laboratory

environment, the actual geodetic path is proportionally scaled down to a reduced trajectory, allowing the UAV to execute a simulated dispatch mission within limited space. For instance, in the dispatch task conducted at Ming Chuan University's Taoyuan Campus, three waypoints were defined: (i) the University Library as the starting point, (ii) the Technology Building as Mission A, and (iii) the International Building as Mission B. The geodetic and local ENU coordinates of these waypoints are listed in Table 6, while the corresponding great-circle distances and bearings used between each route segment are presented in Table 7. The geodetic coordinates were obtained via geocoding and converted into local ENU coordinates using Equations (1)–(3), thereby generating the complete flight route. For demonstration purposes in the laboratory, the inter-waypoint distances were proportionally scaled while preserving the original azimuth angles of each flight segment.

Table 6. Geographic and ENU coordinates of each location.

Location	Latitude (°)	Longitude (°)	X_East (m)	Y_North (m)
Library	24.98357	121.34169	0.00	0.00
Technology Building	24.98508	121.34153	−16.13	167.90
International Building	24.98494	121.34439	272.13	152.34

Table 7. Great-circle distance and bearing between each route.

Route	Distance (m)	Initial Bearing (°)	Final Bearing (°)
Library → Technology Building	168.7	354.514	174.514
Technology Building → International Building	288.7	93.091	273.091
International Building → Library	311.9	240.761	60.761

Note: The initial bearing represents the heading direction when departing the start point along the great-circle path, whereas the final bearing corresponds to the return heading (Initial + 180°) mod 360. The arrow symbol (→) indicates the route direction from the starting location to the destination.

Although waypoints are obtained in geodetic form from Google Maps due to data-source constraints, all trajectory computations in this study are performed in the local planar ENU coordinate system after the transformation described in Section 3.3.

5.3. Gesture Recognition for Flight Commands

To validate the feasibility of the proposed method in the automated dispatch system, the gesture recognition module was integrated into the graphical control interface and tested under practical application scenarios. The training data were collected from air-writing trajectories, where keypoint sequences were first extracted using MediaPipe, converted into CSV format, and subsequently transformed into image representations. The final dataset comprised 1000 samples, with an 80:20 split between training and testing sets. Performance was evaluated using classification accuracy and confusion matrices.

Four models were examined in this study: multilayer perceptron (MLP), one-dimensional convolutional neural network (1D-CNN), two-dimensional convolutional neural network (2D-CNN), and support vector machine (SVM). Their recognition performances were compared across different input features (2D_T, 2D_XY, 2D_XYT, and 1D sequences). Table 8 summarizes the classification accuracies achieved by each model. The results indicate that MLP achieved approximately 75% accuracy, while 1D-CNN reached 81% using one-dimensional sequence inputs. In contrast, the 2D-CNN model achieved the highest accuracy of 95% when trained with combined temporal and spatial features (XYT), outperforming the other approaches. Meanwhile, SVM consistently exhibited stable

performance across different input types, achieving up to 91% accuracy. Based on these comparative results, 2D-CNN was selected as the final gesture recognition model and embedded into the dispatch system interface. Experimental validation confirmed that the model can effectively recognize 16 distinct flight command gestures and convert them in real time into executable dispatch commands for the Tello quadrotor. This integration not only demonstrates the accuracy of the proposed recognition method but also highlights its practical value in real-world UAV applications.

Table 8. Comparison of different models and feature types.

Model	Feature Source	Training Samples	Testing Samples	Accuracy
MLP	Posture, Coordinate, and Direction Feature Images (60 × 60), (60 × 120), (60 × 180)	800	100	75%
2D-CNN (2D_T)	Gesture Posture Feature Image (60 × 60)	800	100	83%
2D-CNN (2D_XY)	Coordinate Feature Image (60 × 120)	800	100	87%
2D-CNN (2D_XYT)	Time × Coordinate × Direction Feature Image (60 × 180)	800	100	95%
SVM (2D_T)	Linear Kernel	800	100	89%
SVM (2D_XY)	Linear Kernel	800	100	87%
SVM (2D_XYT)	Linear Kernel	800	100	91%
1D-CNN	One-Dimensional Time-Series	800	100	81%

Overall, CNN demonstrated the best performance under multimodal inputs, while SVM exhibited consistent robustness, and MLP served primarily as a baseline model. Although the original dataset consisted of 800 training samples and 49 test samples for the 2D_T case, the effective dataset size was substantially expanded through translation- and rotation-based augmentation. This strategy not only compensated for the limited number of collected samples but also enhanced the robustness of the model by simulating realistic variations in air-writing gestures. Similar to prior works [26–28], this study demonstrates that even with a relatively small dataset, the proposed feature fusion and preprocessing pipeline can achieve reliable recognition performance through data augmentation and lightweight CNN architectures.

6. Conclusions and Future Work

This study, based on the DJI RoboMaster TT SDK 2.0, developed a quadrotor automated dispatch system that integrates logistics route planning with air-writing gesture recognition. The system employs a Raspberry Pi as the main control unit, combining a graphical user interface (GUI), Google Maps-based route planning, ENU plane coordinate transformation, and a convolutional neural network (CNN)-based gesture recognition model. Flight control was achieved through UDP socket communication between the ESP32 microcontroller and the Tello drone. Experimental results confirmed that the proposed system can effectively plan dispatch routes, recognize 16 flight gesture commands in real time, and execute automated dispatch operations with image feedback.

The primary contribution of this work lies in the practical application of gesture recognition technology for drone mission control. By incorporating multimodal features, the CNN model achieved a recognition accuracy of 95% and was successfully embedded into the dispatch system. Overall, the results demonstrate that the multilayer perceptron (MLP) serves only as a baseline model with limited recognition capability; CNN exhibits superior performance in feature learning, particularly when combining temporal, spatial, and directional features; and the support vector machine (SVM) provides robust performance in

small-sample scenarios, serving as a reliable non-deep-learning benchmark that can in some cases achieve accuracy close to CNN. These findings suggest that CNN with multimodal features is the preferred choice when aiming for higher recognition accuracy, whereas SVM offers a simpler yet stable alternative. The implementation and validation of this system not only confirm the feasibility of the proposed approach but also highlight its practical potential in education, intelligent surveillance, and automated dispatch applications.

The proposed air-writing system achieves state-of-the-art performance through several key design innovations. First, a multi-modal fusion of spatial and temporal trajectory features—represented by T-plots, XY-plots, and XYT-plots—is employed to comprehensively capture both the geometric shape and dynamic evolution of the gestures. Second, this representation enables the model to achieve high recognition accuracy (95%) while requiring only a small number of training samples, demonstrating strong data efficiency.Third, by incorporating coordinate translation and rotation augmentations, the network exhibits robustness to variations in writing position, scale, and orientation. Finally, the overall framework is implemented with a lightweight CNN capable of real-time inference on an embedded Raspberry Pi platform, confirming its practicality for field deployment in UAV gesture control applications.

Future work may further extend the system toward multi-drone collaboration, voice-based control, and more complex mission scenarios. For example, integrating environmental sensing modules could enable applications in monitoring and disaster response, while the addition of energy modules, such as solar cells, could enhance endurance. Expanding the dataset and leveraging multimodal inputs are expected to further improve recognition accuracy and system adaptability. Moreover, the RoboMaster TT itself provides extendable functionalities, such as a dot-matrix LED for swarm performances and visualization, and a TOF ultrasonic ranging module to support collision avoidance in swarm operations.

In addition, the control layer of the UAV system could be further improved by adopting advanced fuzzy control techniques to enhance system robustness and transient performance under nonlinear or uncertain conditions. Recent studies have explored decentralized, observer-based, and sliding-mode fuzzy control strategies for nonlinear and multi-agent systems, offering valuable insights for future integration into UAV flight control frameworks [29–33]. As drone flight and imaging technologies continue to advance, combining innovative concepts with existing hardware modules will enable more diverse applications, promoting the development of UAVs in education, monitoring, disaster management, and logistics. In summary, the contributions of this study go beyond theoretical exploration, as the proposed system was fully implemented and validated, demonstrating high feasibility and strong practical applicability.

Demonstration videos of the proposed system are provided in the Supplementary Materials (Videos S1–S3).

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/pr13123984/s1, Video S1: Air-Writing Gesture-Based Quadrotor Control; Video S2: Autonomous Flight Mission via Map-Based Waypoint Planning; Video S3: Integrated Gesture–Speech Controlled Quadrotor System.

Author Contributions: Conceptualization, P.-S.T.; methodology, P.-S.T.; system design and implementation, P.-S.T.; software, Y.-C.W.; validation, T.-F.W. and Y.-C.W.; formal analysis, T.-F.W.; investigation, T.-F.W.; data curation, Y.-C.W.; writing—original draft preparation, T.-F.W.; writing—review and editing, P.-S.T. and T.-F.W.; visualization, Y.-C.W.; supervision, P.-S.T.; project administration, P.-S.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Science and Technology Council (NSTC), Taiwan, under Grant No. NSTC 114-2221-E-197-017.

Data Availability Statement: The original contributions presented in this study are included in the article/Supplementary Materials. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would also like to thank the members of the Robotics Lab for their assistance in system implementation.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kinaneva, D.; Hristov, G.; Raychev, J.; Zahariev, P. Early Forest Fire Detection Using Drones and Artificial Intelligence. In Proceedings of the 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 20–24 May 2019; pp. 1060–1065.
2. Lee, J.; Jo, H.; Oh, J. Application of Drone LiDAR Survey for Evaluation of a Long-Term Consolidation Settlement of Large Land Reclamation. *Appl. Sci.* **2023**, *13*, 8277. [CrossRef]
3. Hu, Z.; Bai, Z.; Yang, Y.; Zheng, Z.; Bian, K.; Song, L. UAV-Aided Aerial–Ground IoT for Air Quality Sensing in Smart City: Architecture, Technologies, and Implementation. *IEEE Netw.* **2019**, *33*, 14–22. [CrossRef]
4. Li, C.; Li, Z. Research on the Use of Drones for Logistics Distribution Network Terminal System Based on Computer 5G Technology. In Proceedings of the 2022 International Conference on Electronics and Devices, Computational Science (ICEDCS), Marseille, France, 25–27 May 2022; pp. 441–445.
5. Ryze Tech Co., Ltd. DJI. Tello Drone Specifications. Available online: https://www.ryzerobotics.com/tello/specs (accessed on 11 October 2025).
6. Kang, K.; Li, Y.; Liu, X.; Huang, Z. A Human-Following Drone Providing Gesture Recognition. *Multimed. Tools Appl.* **2025**, *84*, 12345–12360.
7. Pawlicki, M.; Hulek, K.; Ostrowski, A.; Możaryn, J. Implementation and Analysis of Ryze Tello Drone Vision-Based Positioning Using AprilTags. In Proceedings of the 27th International Conference on Methods and Models in Automation and Robotics (MMAR), Międzyzdroje, Poland, 22–25 August 2023; pp. 309–313.
8. Fahmizal; Afidah, D.; Istiqphara, S.; Abu, N.S. Interface Design of DJI Tello Quadcopter Flight Control. *J. Fuzzy Syst. Control* **2023**, *1*, 49–54. [CrossRef]
9. Boonsongsrikul, A.; Eamsaard, J. Real-Time Human Motion Tracking by Tello EDU Drone. *Sensors* **2023**, *23*, 897. [CrossRef] [PubMed]
10. Fakhrurroja, H.; Pramesti, D.; Hidayatullah, A.R.; Fashihullisan, A.A.; Bangkit, H.; Ismail, N. Automated License Plate Detection and Recognition Using YOLOv8 and OCR with Tello Drone Camera. In Proceedings of the 2023 International Conference on Computer, Control, Informatics and Its Applications (IC3INA), Bandung, Indonesia, 25–26 October 2023; pp. 206–211.
11. Iskandar, M.; Bingi, K.; Ibrahim, R.; Omar, M.; Mozhi Devan, P.A. Hybrid Face and Eye Gesture Tracking Algorithm for Tello EDU RoboMaster TT Quadrotor Drone. In Proceedings of the 2023 Innovations in Power and Advanced Computing Technologies (i-PACT), Kuala Lumpur, Malaysia, 23–25 November 2023; pp. 1–6.
12. Brashear, H.; Henderson, V.; Park, K.H.; Hamilton, H.; Lee, S.; Starner, T. American Sign Language Recognition in Game Development for Deaf Children. In Proceedings of the International ACM SIGACCESS Conference on Computers and Accessibility (Assets), Portland, OR, USA, 22–25 October 2006; pp. 79–86.
13. Kadous, M.W. Machine Recognition of Auslan Signs Using PowerGloves: Towards Large-Lexicon Recognition of Sign Language. In Proceedings of the Workshop on the Integration of Gesture in Language and Speech, Cambridge, UK, 5–7 April 1996; pp. 165–174.
14. Indriani; Harris, M.; Agoes, A.S. Applying Hand Gesture Recognition for User Guide Application Using MediaPipe. In Proceedings of the 2nd International Seminar of Science and Applied Technology (ISSAT), Bandung, Indonesia, 20–21 October 2021; pp. 1–6.
15. Hsieh, C.-H.; Lo, Y.-S.; Chen, J.-Y.; Tang, S.-K. Air-Writing Recognition Based on Deep Convolutional Neural Networks. *IEEE Access* **2021**, *9*, 142827–142836. [CrossRef]
16. Zan, H.; Wang, M.; Sun, M. Aerial Handwriting Recognition Based on SVM. In Proceedings of the 2018 International Symposium on Computer, Consumer and Control (IS3C), Taichung, Taiwan, 6–8 December 2018; pp. 250–253.
17. Moriya, M.; Hayashi, T.; Tominaga, H.; Yamasaki, T. Video Tablet Based on Stereo Camera: Human-Friendly Handwritten Capturing System for Educational Use. In Proceedings of the Fifth IEEE International Conference on Advanced Learning Technologies (ICALT), Kaohsiung, Taiwan, 5–8 July 2005; pp. 909–911.
18. Zhang, F.; Bazarevsky, V.; Vakunov, A.; Tkachenka, A.; Sung, G.; Chang, C.-L.; Grundmann, M. MediaPipe Hands: On-Device Real-Time Hand Tracking. *arXiv* **2020**, arXiv:2006.10214.
19. Nguyen Huu, P.; Nguyen Thi, N.; Ngoc, T.P. Proposing Posture Recognition System Combining MobileNetV2 and LSTM for Medical Surveillance. *IEEE Access* **2022**, *10*, 1839–1849. [CrossRef]

20. Chang, I.-C.; Liao, C.-S.; Yen, C.-E. The Energy-Aware Multi-UAV Dispatch and Handoff Algorithm for Maximizing the Event Communication Time in Disasters. *Appl. Sci.* **2021**, *11*, 1054. [CrossRef]
21. GeeksforGeeks. Get a Google Map Image of Specified Location Using Google Static Maps API. Available online: https://reurl.cc/Vm3y25 (accessed on 11 October 2025).
22. Sinnott, R.W. Virtues of the Haversine. *Sky Telesc.* **1984**, *68*, 159.
23. Gade, K. A Non-singular Horizontal Position Representation. *J. Navig.* **2010**, *63*, 395–417. [CrossRef]
24. Veness, C. Calculate Distance, Bearing and More Between Latitude/Longitude Points. Movable Type Scripts 2002–2023. Available online: https://www.movable-type.co.uk/scripts/latlong.html (accessed on 10 November 2025).
25. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning Representations by Back-Propagating Errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
26. Al Abir, F.; Al Siam, M.; Sayeed, A.; Hasan, M.A.M.; Shin, J. Deep Learning-Based Air-Writing Recognition with the Choice of Proper Interpolation Technique. *Sensors* **2021**, *21*, 8407. [CrossRef] [PubMed]
27. Schlüsener, N.; Bücker, M. Fast Learning of Dynamic Hand Gesture Recognition with Few-Shot Learning Models. *arXiv* **2022**, arXiv:2212.08363. [CrossRef]
28. Ewe, E.L.R.; Lee, C.P.; Kwek, L.C.; Lim, K.M. Hand Gesture Recognition via Lightweight VGG16 and Ensemble Classifier. *Appl. Sci.* **2022**, *12*, 7643. [CrossRef]
29. Chang, W.J.; Su, C.L.; Lee, Y.C. Estimated-State Feedback Fuzzy Compensator Design via a Decentralized Approach for Nonlinear-State-Unmeasured Interconnected Descriptor Systems. *Processes* **2024**, *12*, 101. [CrossRef]
30. Lin, T.A.; Lee, Y.C.; Chang, W.J.; Lin, Y.H. Robust Observer-Based Proportional Derivative Fuzzy Control Approach for Discrete-Time Nonlinear Descriptor Systems with Transient Response Requirements. *Processes* **2024**, *12*, 540. [CrossRef]
31. Chang, W.J.; Lin, Y.H.; Lee, Y.C.; Ku, C.C. Investigating Formation and Containment Problem for Nonlinear Multi-Agent Systems via Interval Type-2 Fuzzy Sliding Mode Tracking Approach. *IEEE Trans. Fuzzy Syst.* **2024**, *32*, 4163–4177. [CrossRef]
32. Lee, Y.C.; Su, C.L.; Chang, W.J. Decentralized and Performance-Constrained State-Estimated Fuzzy Control for Nonlinear Differential-Algebraic Interconnected Systems. *Inf. Sci.* **2024**, *673*, 120666. [CrossRef]
33. Chang, W.J.; Lin, Y.H.; Ku, C.C. A Comprehensive Survey on Advanced Control Techniques for T-S Fuzzy Systems Subject to Control Input and System Output Requirements. *Processes* **2025**, *13*, 792. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.