建構基於電腦視覺之全自動室內無人機系統

***黃玉君、朱育賢**

南臺科技大學電機工程系

*ych0321@stust.edu.tw

摘要

本研究旨在透過四旋翼無人機作為載具平台，建構基於電腦視覺技術之全自動室內無人機系統。本系統使用X形配置之四旋翼無人機為核心，搭配Pixhawk4飛行控制器和外部感測器、動力旋翼單元等為核心，並透過由USB網路攝影機、樹梅派4 (Raspberry PI4) 、OpenCV 函式庫等，實現電腦視覺系統。該系統可自動偵測並追蹤放置於地面之標記(線段和色塊)，並將其做為參考點，實現室內環境中之定位。此外本系統採用基於比例、微分、積分之控制演算法(PID控制演算法) 和DroneKit-Python函式庫，將視覺定位系統所提供之位置資訊轉換為飛行控制命令，完成無人載具之姿態、移動控制。在控制測試過程中，該系統展現出優秀的穩定性、可靠性、和抗干擾能力，尤在面對非理想環境時，該系統仍可保持良好的控制性能，本系統具有良好的擴展可能性，如可透過增加更多感測器以實現全自動之物體避障，透過多無人機協作以完成複雜任務…等，以進一步提高系統之靈活性和泛用性。

關鍵詞：無人機、電腦視覺、特徵追蹤、室內定位

Building an Autonomous Indoor Drone System Based on a Computer Vision System

Yu-Chun Huang* , Yu-Hsien Chu

Department of Electrical Engineering, Southern Taiwan University of Science and Technology

Abstract

This paper focuses on building an autonomous indoor drone system based on computer vision technology. The system is built on an X-configuration quadcopter, equipped with a Pixhawk 4 flight controller, external sensors, and rotor units. The computer vision system comprises a USB webcam, Raspberry Pi 4, and the OpenCV library, enabling automatic detection and tracking of floor markers to facilitate loitering and maneuvering in indoor environments. The system employs a PID control algorithm and the DroneKit-Python library to integrate location data from the computer vision system with sensor data from the flight control unit for precise movement control. The system demonstrated stability, reliability, and robust performance during testing, even under non-ideal environmental conditions. The system also has strong potential for future upgrades, including additional sensors for obstacle avoidance and multi-drone collaboration, enhancing flexibility and efficiency.

Keywords: Drone, Computer vision, Features tracking, Indoor navigation

Received: Feb. 11, 2025; first revised: Mar. 25, 2025; second revised: Apr. 1, 2025; accepted: Apr. 2025.

Corresponding author: Y.-C. Huang, Department of Electrical Engineering, Southern Taiwan University of Science and Technology, Tainan 710301, Taiwan.

I. Introduction

A drone (also known as an Unmanned Aerial Vehicle, UAV) is a pilotless, unmanned aircraft that can be controlled remotely or by an onboard computer (companion computer). Drones, particularly multicopters driven by electric motors have many benefits, including high efficiency, low cost, maneuverable, high stability etc. These advantages make drones widely used in fields such as aerial photography, delivery, search and rescue, agriculture, and entertainment.

In the past few decades, drones were developed for military purposes. Nevertheless, after several years of development and advances in small electric motors, batteries, motor controllers, and flight control units, drones—particularly multicopters—have entered the stage of practical application.

This technology is extensively used in the applications mentioned above; it is also enabling hobbyists and enthusiasts to become passionate drone operators. Most hobbyists use drone systems for flying around the yard, taking impressive aerial videos, or doing freestyle FPV (First Person View) flying.

However, the majority of civil applications, such as search and rescue, agriculture, or other innovative uses, rely on some form of computer-aided autonomous or automated flight. Modern commercial drones use Global Navigation Satellite System (GNSS) technologies like Global Positioning System (GPS) or GALILEO to perform basic autonomous flight, allowing operators to hover the drone in 3D space or set a 'flight plan' within a ground control station, so the drone can travel from one waypoint to another and also have the ability to return home [1].

However, all current autonomous or automated systems rely on GNSS for positioning and are designed primarily for outdoor use. Since GNSS can be prone to errors and may not work at all in indoor environments, we must use other indoor positioning systems for indoor autonomous flight.

After conducting research, we found several approaches for resolving indoor drone positioning. Most of these approaches utilize ultra-wideband (UWB), Bluetooth, or RF beacons to construct an indoor positioning system [2]. The use of these technologies requires the deployment of numerous beacons throughout the environment, which serve as anchor nodes to determine the actual location. Another approach uses motion-tracking cameras mounted on the ceiling to track markers on the robot, providing accurate location data. With a few exceptions, most approaches rely on additional expensive hardware and setup to implement the system, such as beacons and motion-tracking cameras. Although the methods mentioned above can provide very accurate positioning data, the required hardware is quite complex to set up and is also expensive to purchase.

For the reasons mentioned above, we decided to implement a computer vision system on board that is able to automatically detect and analyze the markers on the floor, so that we can use algorithms to obtain the accurate position. Using this system can significantly reduce the purchase cost.

In order to achieve autonomous operations for drones, we decided to use ArduPilot firmware-based flight controllers, as well as DroneKit-Python, which runs on an onboard companion computer. ArduPilot is an open-source autopilot software that can control a wide range of vehicle systems, including conventional airplanes, multirotors, helicopters, boats, and even submarines. It also enables the use of various sensors, companion computers, and communication systems. Additionally, we can use DroneKit-Python to control ArduPilot via a low-latency link from the onboard companion computer, allowing us to implement the control algorithm to achieve autonomous operation features.

II. System Architecture

The system consists of two distinct components that operate collaboratively: the drone vehicle unit and the companion computer. The block diagram of the system architecture is presented in Figure 1.

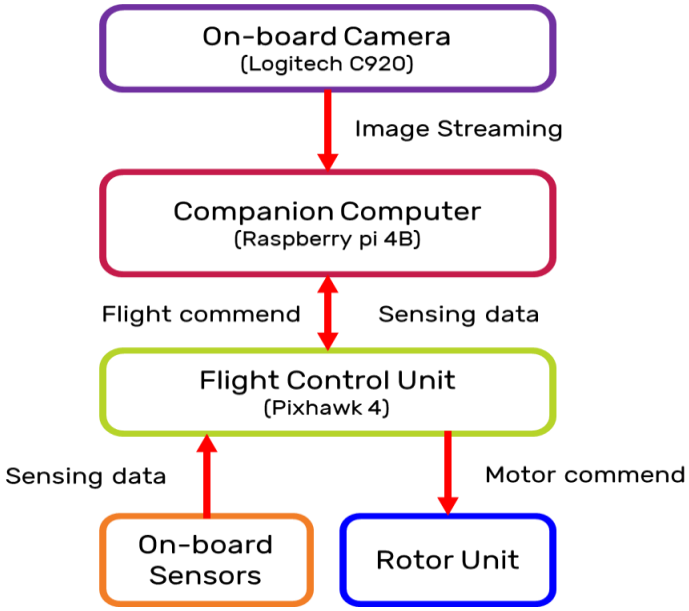


Figure1. *System Block Architecture*

The companion computer, based on the embedded system (Raspberry Pi 4B), consists of three main components. The first component is the on-board camera image streaming, which is based on a Universal Serial Bus (USB) webcam. The second component is the computer vision system, which utilizes algorithms to detect and track features in the streaming images from the webcam, such as color blocks or lines on the floor, to provide accurate location data. The third component is the flight command algorithm, which uses the location data from the computer vision system and sensor data from the flight control unit to generate commands that track the target position. These commands are then sent to the flight control unit to execute the flight actions.

The algorithms of the computer vision system will be described in detail in the 'Computer Vision System' chapter, and the flight command algorithm will be described in detail in the 'Drone Control System' chapter. This chapter will further describe the drone unit, which consists of the flight control unit, onboard sensors, rotor unit, frame, and other components. It serves as the vehicle platform for the entire system.

III. Computer Vision System

In this system, we focus on two types of features: lines and color blocks. The computer vision system is designed to detect these markers on the floor. It consists of two primary processes: image preprocessing and feature detection and tracking algorithms. Both of these features rely on the OpenCV-Python library, which runs on the companion computer (Raspberry Pi4).

1. Image Preprocess

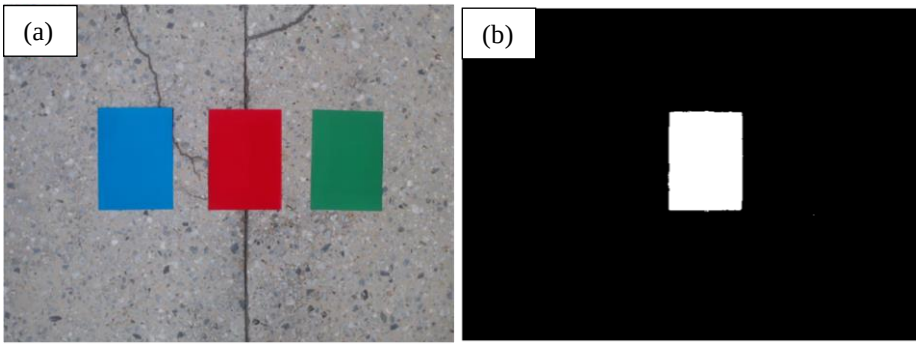
Image preprocessing involves manipulating raw frame data into a usable format. This step is crucial for reducing distortions and enhancing the features needed for effective detection and tracking. To this end, we employ several

techniques during the image preprocessing process, including grayscale conversion, Gaussian smoothing, binarization, and color masking, among others [3]. This section will provide a description of the image preprocessing process, focusing on two main features: color blocks and lines.

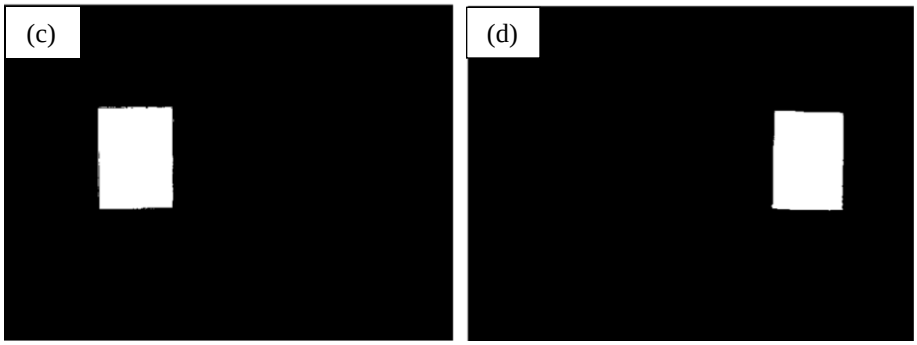
(1) Color blocks

In this system, color blocks serve as anchors for loiter control, aiding in the stabilization of the vehicle at a specific position, replacing the GNSS positioning system. The main technique used in image preprocessing is color masking, to isolate specific color areas in an image.

The original frame is passed through a color mask defined by the RGB threshold, where pixels not belonging to the target color area are blocked, and only the pixels corresponding to the desired color pass through, thereby highlighting the specific color region [4]. The following figure illustrates how color masking works.



(a) original frame: directly captured from the webcam. (b) Red color mask



(c) Blue color mask (d) Green color mask

Figure 2. *The processing steps of color block image preprocessing*

(2) Lines

Lines are used to create patterns that guide the movement of the vehicle, similar to how trains follow railway tracks. In this context, the vehicle maneuvers between color blocks. During the image preprocessing of line markers, several techniques are utilized, including grayscale conversion, Gaussian smoothing, and binarization.

The original frame is first processed through grayscale conversion. Since the line markers consist of black tape on the floor, color information is not essential for the analysis. The resulting grayscale image is then subjected to Gaussian smoothing to reduce noise and eliminate irrelevant details [5]. Subsequently, the image undergoes binarization, converting it into a binary format for subsequent processing. Figure 3 illustrates these processing steps. Each subfigure (a)–(d) corresponds to a specific image processing stage: (a)The original frame is directly captured

from the webcam. (b) The cv2.cvtColor() function from the OpenCV library is used for color space conversion. (c) The grayscale image is processed through a 9×9 Gaussian smoothing filter to reduce image noise. (d) The Gaussian-smoothed image is binarized with a threshold value set to 105 to produce a binary image suitable for the line tracking system.

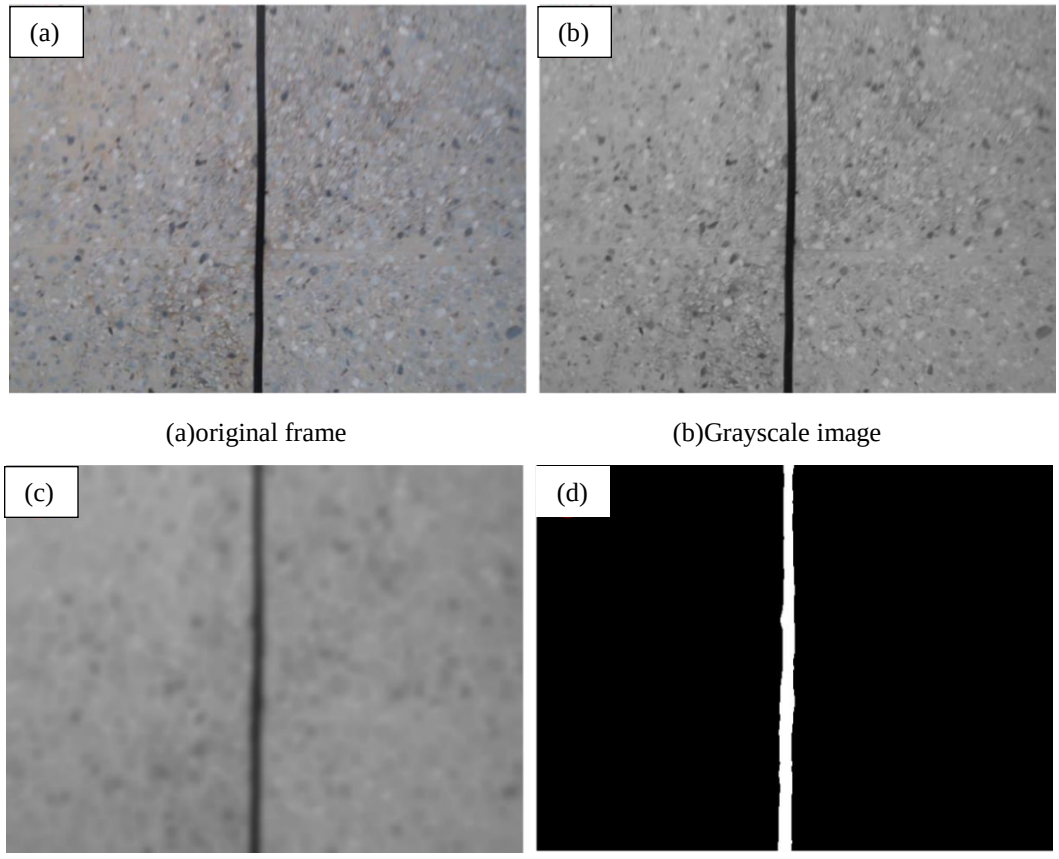


Figure 3. *The processing steps of lines image preprocessing*

2. Feature Detection and Tracking

Feature detection and tracking system uses the processed frames, which have passed through image preprocessing, to provide error data for the drone control system, enabling it to adjust the vehicle's movement. During this process, we employ several techniques provided by OpenCV, including contour detection, contour area calculation, and moment analysis. Including contour detection, contour area calculation, and moment analysis, among others. In this process, we are still primarily focused on two key features: color blocks and lines.

(1) Color blocks

To detect and track the features of the color blocks, the binary image obtained from the image preprocessing step is first input into the cv2.findContours() function to identify the contours of the color blocks. Subsequently, the cv2.contourArea() function is employed to calculate the area of each contour. The largest contour is processed using the cv2.moments() function to compute its centroid. This enables the comparison of which facilitates the calculation of positional errors along the x and y axes.

The subsequent figure illustrates the outcome of the color block detection and tracking process. The green box highlights the region of the color block (the red block in the figure), the green dot represents the centroid of the block, and the white line denotes the center coordinates of the image. The yellow text displays the calculated error values along the x and y axes.

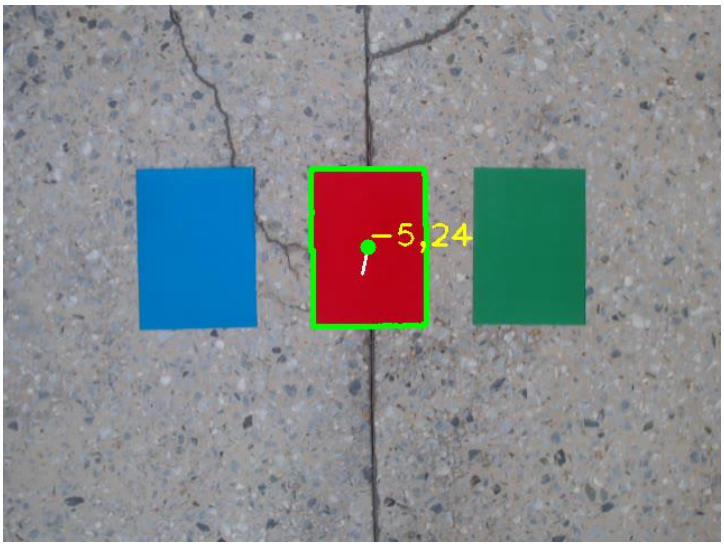


Figure 4. *Results of the color block detection and tracking system*

(2) Lines

In the line detection and tracking process, the binary image obtained from the image preprocessing step is initially divided into three equally sized horizontal sections, as illustrated in Figure 5.

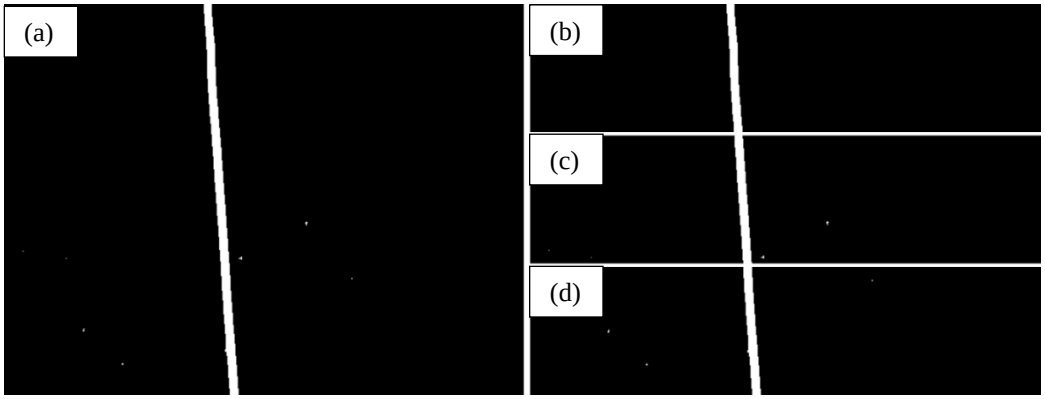


Figure 5. *The binary image divided into three equal horizontal sections*

Subsequent to the slicing process, the image is processed using the functions cv2.findContours(), cv2.contourArea(), and cv2.moments(), similar to the methods used in the color block detection procedure. These functions facilitate the computation of error along the x-axis. Following this, three distinct error values are derived and compared with the image's center coordinates. The central error value is utilized as the roll axis error correction, while the top and bottom error values are compared to calculate the yaw axis error for the control system. The following figure illustrates the processes and outcomes of the line detection and tracking system. In Figure 6, the white dot represents the centroid of the detected lines, the green line corresponds to the image's center, the purple line indicates the error in line detection, and the red text denotes the error values (in pixels).

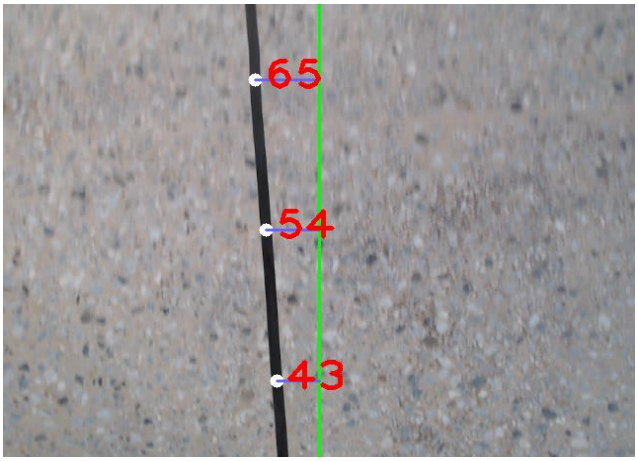


Figure 6. *The results of the line detection and tracking system*

IV. Drone Control System

The drone control system consists of two main components: the autonomous system software and the drone platform. The drone platform utilizes a multicopter, specifically an X-configuration quadcopter, and integrates autonomous control software. The primary objective of this system is to utilize error data provided by the computer vision system to enable autonomous control of the drone. The following section explains the functionality of these two systems and presents the testing results of the entire system.

1. Drone system architecture

The core of the drone system is the Flight Control Unit (Pixhawk4 in this case), which contains several critical sensors including a compass, gyroscope, barometer, and accelerometer. The primary function of the Flight Control Unit is to utilize these sensors, along with external data sources (such as an external rangefinder and companion computer in this case), to control the vehicle's posture and movement. This system also requires output devices, such as the rotor unit (which comprise electronic speed controllers and brushless DC motors) as actuators, a buzzer and OLED screen for status indication, and a telemetry radio for vehicle state monitoring. For safety purposes, a radio remote control receiver is also employed for emergency control. The following figure illustrates the architecture of the drone system.

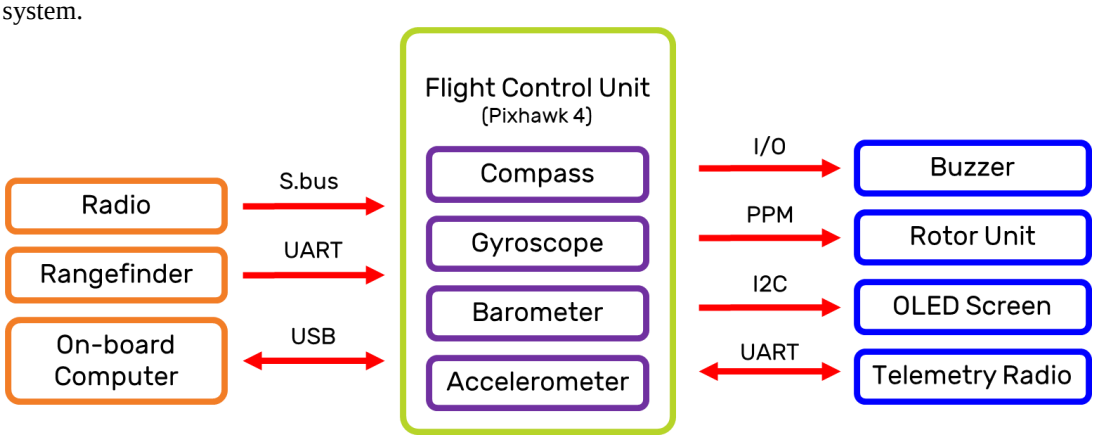


Figure 7. *The architecture of the drone system.*

Figure 8. *The physical components of the drone system*.

2. Autonomous system software

The autonomous system software is based on the PID (Proportional-Integral-Derivative) control algorithm and the DroneKit-Python library [6], [7]. The PID control algorithm utilizes the error values provided by the Computer Vision System to compute the control output. The DroneKit-Python library is employed to send flight commands to the drone system for movement control. The following section explains how the PID control algorithm and the DroneKit library function.

(1) PID control algorithm

The PID control algorithm operates as a closed-loop controller. The system takes the error value from the computer vision system as input and outputs control commands to the DroneKit library to maneuver the vehicle.

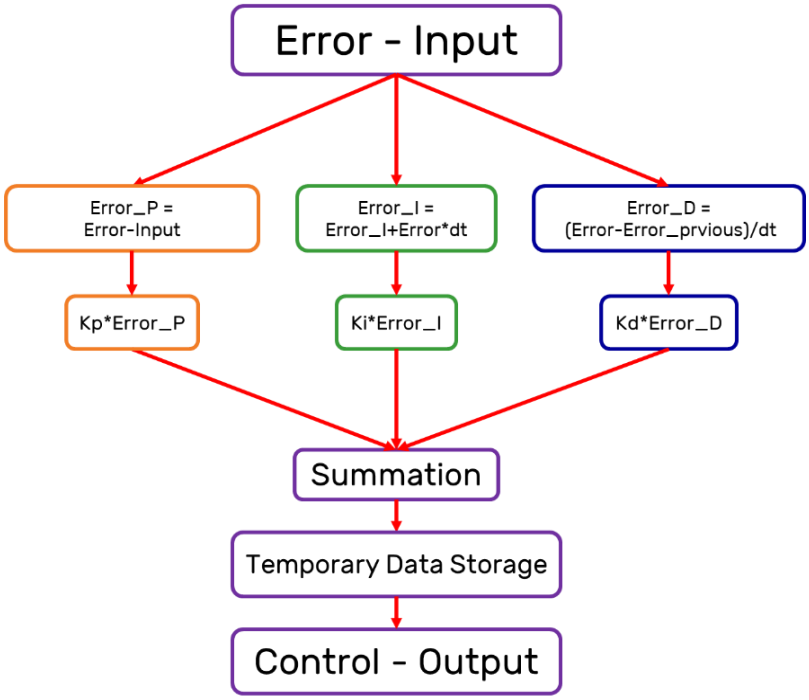


Figure 9. *The architecture of the PID control algorithm*

This algorithm operates in a continuous loop. During each iteration, the values for Error_P, Error_I, and Error_D are calculated. The Error_P value is equal to the input error, the Error_I value is calculated by adding the previous integral error to the product of the input error and the frame's duty time, and the Error_D value is the difference between the input error and the previous error, divided by the frame's duty time. After calculating the error values, they are multiplied by their respective weights, Kp, Ki, and Kd, and then summed. Before outputting the control command, the integral error, previous error, and the frame's duty time are temporarily stored for the next iteration. The following figure illustrates the architecture of the PID control algorithm.

To effectively maintain steady control of the vehicle, tuning the parameters of the PID control algorithm is essential. The Kp parameter determines the system's responsiveness, the Ki parameter addresses steady-state error, and the Kd parameter helps to smooth the control output. The table below illustrates the relationship between parameter adjustments and system response, along with the final selected values for each parameter.

Table 1. *PID Parameter Adjustment Impact and Final Selected Value*

Parameter	Adjustment	Adjust the impact of the results	First Selected value
Kp	Increase	Increase control sensitivity. Excessive sensitivity may lead to instability.	1.2
	Decrease	Excessively low gain may lead to control failure.	
Ki	Increase	Reduce the steady-state error. Excessive gain may lead to overshoot.	0.2
	Decrease	Excessively low gain may lead to steady-state error cannot be completely eliminated.	
Kd	Increase	Smoothing the control output, excessive gain may result in a slow-response system.	0.38
	Decrease	Unstable control output.	

(2) DroneKit-Python Library

DroneKit-Python is an open-source SDK (Software Development Kit) based on the MAVLink (Micro Air Vehicle Link) protocol [8], designed to interface with ArduPilot-based flight control systems through a companion computer. It provides a Python library that enables interaction with the flight control unit and enables control of the vehicle's behavior.

In our system, we utilize the channel_overrides (also referred to as "RC overrides") function provided by DroneKit [9]. This function is employed to transmit flight commands, including the values for Roll, Pitch, and Yaw, to the flight control unit, thereby controlling the vehicle's movement and completing the closed-loop system.

3. System Testing Results

The performance of the Autonomous Indoor Drone System, which includes the Computer Vision System, Drone Control System, and other subsystems, shows impressive results during the Color Block and Line Tracking tests. It demonstrates precise target tracking and exhibits good reliability and performance, even when facing environmental interference (such as wind or different illumination conditions).

The following figure shows the System Response of the Autonomous Indoor Drone, where it is evident that the error (X-axis in pixels) is significantly controlled. The figure shows that when the PID control algorithm is activated at 2000 ms, the error begins to be corrected. After 4.5 seconds, the vehicle achieves stable control, with the error remaining within a range of ±25 pixels. After 11,500 ms, the PID control algorithm is deactivated, and the vehicle begins to drift again.

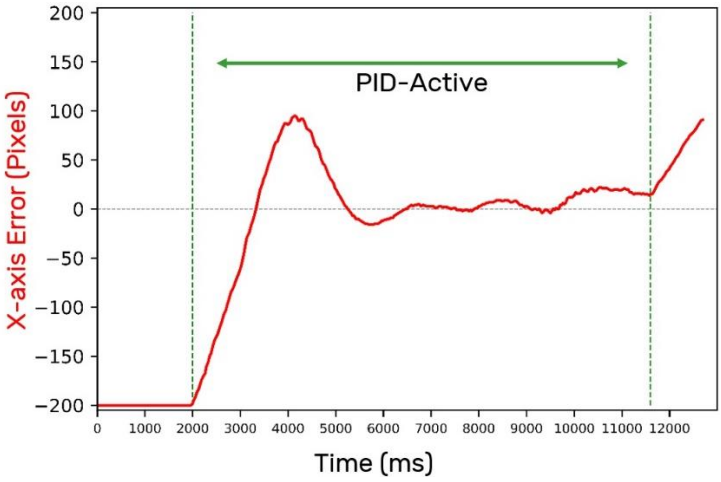


Figure 10. *Response of the Autonomous Indoor Drone System*

To assess the effectiveness of the autonomous flight control algorithm, we conducted tests in an outdoor environment to replicate non-ideal conditions, including wind and varying illumination. The vehicle was evaluated in the shadows of buildings and on an unshaded asphalt road, experiencing wind speeds between 3.5 and 5 m/s, as reported by the Central Weather Administration (CWA). The figure below illustrates the testing environment and conditions under which the vehicle was evaluated. Specifically, subfigure (a) shows the flight test conducted under the shadow of buildings, representing a scenario with partial GNSS signal obstruction due to surrounding structures. In contrast, subfigure (b) presents the test conducted on an open asphalt road with no shade, allowing for optimal GNSS reception and ideal conditions for autonomous flight operations.

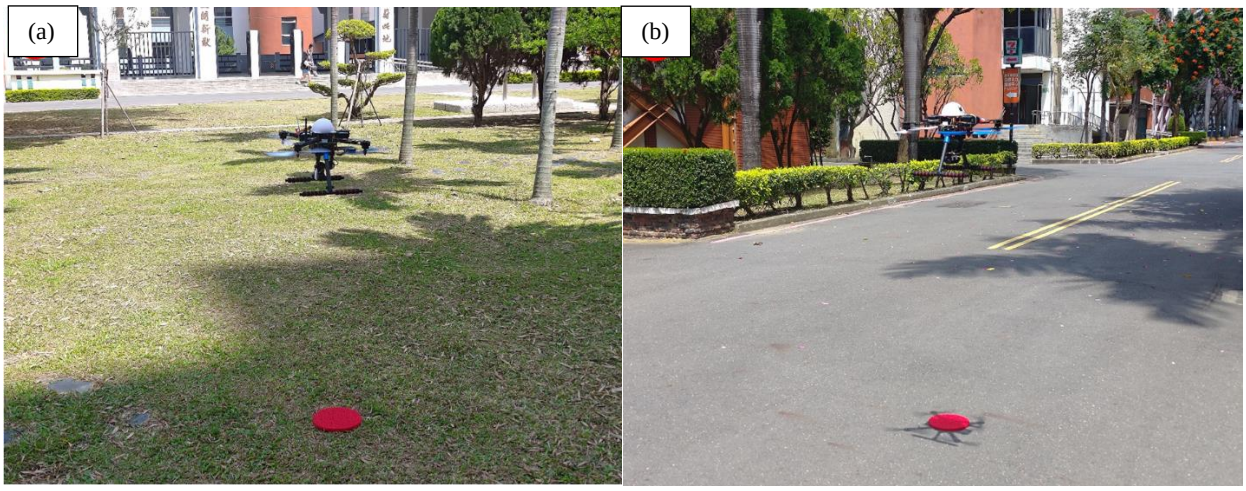


Figure 11. *Testing environment and the vehicle's testing conditions*

During the outdoor test, we applied the same PID control algorithm used in the indoor testing. After the test, we observed that the system's stability was clearly robust, even under non-ideal flight conditions. Although the vehicle's response was not as accurate or fast as in the indoor environment, it remained stable and usable despite these challenging conditions. The figure below illustrates the testing environment and the vehicle's performance during the test. Figure 12. (a) Vehicle response during flight test under building shadows. (b) Vehicle response during flight test on an asphalt road without shade.

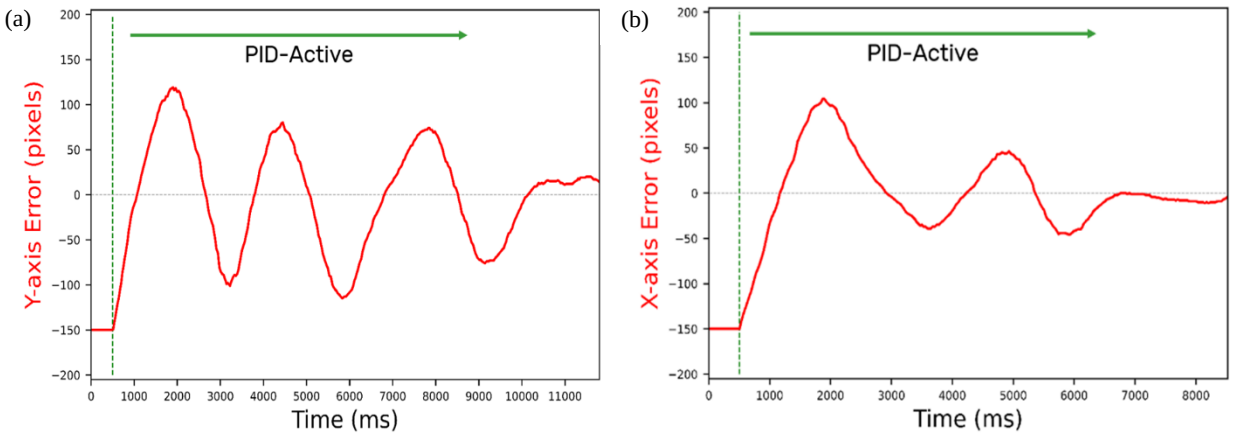


Figure 12. *Testing environment and the vehicle's testing conditions:*

V. Conclusion

This paper focuses on the use of computer vision technology to build an autonomously controlled indoor drone system. The system employs an X-configuration quadcopter as the base platform, utilizing a Pixhawk 4 as the primary flight controller, along with external sensors and a rotor unit. The computer vision system is based on a USB webcam, Raspberry Pi 4, and the OpenCV library, enabling the system to automatically detect and track markers placed on the floor, which are used to facilitate loitering and maneuvering within indoor environments. During the testing process, the system demonstrated stability, reliability, and robust performance even under non-ideal environmental conditions.

Looking ahead, the system has considerable potential for future upgrades. For instance, additional sensors could be integrated to enable automatic obstacle avoidance, enhancing the drone's navigation capabilities in complex environments. Additionally, the system could be expanded to support multi-drone operations for collaborative missions, which would enhance the overall efficiency and flexibility of the autonomous drone network. In future research, we plan to develop an automatic PID tuning algorithm and use advanced visual fiducial markers, such as AprilTag, to replace the simple color blocks and lines currently in use. By incorporating these technologies, the autonomous drone system will achieve more accurate and precise flight movements, significantly improving its performance and navigation capabilities in three-dimensional space.

Reference

[1] M, Kan, S. Okamoto, and J. H. Lee, "Development of drone capable of autonomous flight using GPS," *International MultiConference of Engineers and Computer Scientists (IMECS 2018)*, Hong Kong, China, March 14-16, 2018, pp.665–669.

[2] S. Lee, S. Yoo, J. Y. Lee, S. Park, and H. Kim, "Drone positioning system using UWB sensing and out-of-band control," *IEEE Sens. J.*, vol.22, no. 6, pp. 51570–51589, 2021.

[3] S. E. Umbaugh, *Computer vision and image processing: A practical approach using CVIPtools*. Prentice Hall, Upper Saddle River, NJ, USA, 1998.

[4] Y. Zhang, Y., Li, Y., Wang, and Y. Zhang, “Research on target recognition algorithm based on OpenCV,”. *3rd International Conference on Computing, Networks and Internet of Things (CNIOT 2022)*, Sanya, China, July, 2022, pp.1–5.

[5] T. Lindeberg, “Discrete approximations of Gaussian smoothing and Gaussian derivatives,” *J. Math. Imaging Vis.*, vol. 66, pp.759–800, 2024.

[6] V. M. Babu, K. Das, and S. Kumar, “Designing of self-tuning PID controller for AR drone quadrotor,” *2017 International Conference on Advanced Robotics (ICAR 2017)*, Hong Kong, China, July 2017, pp.1–6.

[7] A. B. Pulungan, Z. Y. Putra, A. R. Sidiqi, H. Hamdani, and K. E. Parigalan, ”Drone Kit-Python for autonomous quadcopter navigation,” *Int. J. Inform. Vis.*, vol. 8, no. 3, pp. 1287–1294, Sep. 2024.

[8] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui, “ Micro air vehicle link (MAVLink) in a nutshell: A survey,”. *IEEE Access*, vol. 7, pp. 87658–87680, 2019.

[9] L. W. Graves, “Infrared video tracking of UAVs: Guided landing in the absence of GPS signals*”* M.S. thesis, dept. Elect. Eng., Univ. Alaska, Fairbanks, 2019.