Ministry of Education and Science of Ukraine

National University of "Kyiv-Mohyla Academy"

Department of Mathematics of the Faculty of Informatics

Development of a Control Algorithm for a Two-Axis Gimbal for Object Tracking on UAV

Text part to the thesis in the specialty "Applied Mathematics" - 113

Thesis Supervisor:

Senior Lecturer

Andrew Kurochkin

_____ (Signature)

"___" _____ 2026

Done by Student:

Zorian Kulyk

"___" _____ 2026

Kyiv, 2026

Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"
Department of Mathematics of the Faculty of Informatics

Approved
Head of the Department of Mathematics,
Associate Professor, Ph.D.
R. K. Chorney

_____ (Signature)

"___" _____ 2026

INDIVIDUAL TASK
for thesis
of the student Zorian Kulyk
4th year of the Faculty of Informatics
topic: Development of a Control Algorithm for a Two-Axis Gimbal for Object Tracking on UAV

The content of the PM to the thesis:
Abstract
Introduction
Related Work
Approach
Solution
Evaluation
Conclusions and Further Work
Bibliography

Date of issue: "___" _____ 2026

Supervisor: _____ (Signature)

Task received: _____ (Signature)

Анотація

У роботі представлено розробку, реалізацію та оцінку алгоритму керування для активного стеження за об'єктом на низьковартісному двовісному підвісі, призначеному для застосування на безпілотному літальному апараті (БПЛА). Платформа поєднує мікрокомп'ютер Raspberry Pi 4 для обробки відеопотоку, політний контролер SpeedyBee F405, що взаємодіє по MAVLink протоколу, камеру IMX378 та підвіс на основі 8-бітної плати керування. Програмний модуль комп'ютерного зору повністю побудований на базі бібліотеки OpenCV та порівнює два кореляційно-фільтрові трекери — MOSSE та CSRT. Похибка положення об'єкта у пікселях, отримана від трекера, безпосередньо перетворюється на ШІМ-команди для підвісу через запропоноване перетворення з коефіцієнтом просторової роздільної здатності, виведеним з внутрішніх параметрів камери. У роботі також виведено пряму кінематику двовісної конфігурації і встановлено її теоретичні обмеження. Експериментальні результати, отримані на зібраній статичній платформі, надають оцінку похибки стеження за об'єктом та підтверджують, що запропонований підхід є здійсненним з огляду на зазначені обмеження апаратного забезпечення.

Ключові слова: активне стеження за об'єктом, двовісний підвіс, БПЛА, коефіцієнт просторової роздільної здатності, низьковартісний підвіс.

Abstract

This work presents the design, implementation, and evaluation of a control algorithm for active object tracking on a low-cost, two-axis (pitch-roll) brushless gimbal intended for UAV deployment. The platform combines a Raspberry Pi 4 for vision processing, a SpeedyBee F405 flight controller communicating over MAVLink, an IMX378 camera, and a gimbal based on an 8-bit controller board. The vision pipeline is built entirely on OpenCV and compares two correlation-filter trackers, MOSSE and CSRT. Pixel error from the tracker is mapped directly to gimbal PWM commands through a proposed conversion with a spatial resolution coefficient derived from the camera intrinsics. The work also derives the forward kinematics of the two-axis pitch–roll configuration and establishes its theoretical limitations. Experimental results obtained on the assembled static platform provide an evaluation of the object tracking error and confirm that the proposed approach is feasible within the stated hardware constraints.

Keywords: active object tracking, two-axis gimbal, UAV, spatial resolution coefficient, low-cost gimbal.

Table of Contents

Анотація i
Abstract ii

Introduction 1
Background . . . 1
Motivation . . . 1
Objectives and Scope . . . 2
Thesis Structure . . . 2

1 Related Work 3
1.1 System Architecture . . . 3
1.2 Gimbal Control . . . 3
1.3 Vision-Based Tracking . . . 4
1.4 Camera Resolution . . . 5

2 Approach 6
2.1 Companion Computer . . . 6
2.2 Flight Controller . . . 6
2.3 Camera . . . 7
2.4 Gimbal . . . 7
2.5 Development Environment . . . 8
2.6 Gimbal Lock . . . 9
2.7 Power Distribution System . . . 10
2.8 Mathematical Foundations . . . 11
2.9 Motor Command Formulation . . . 14

3 Solution 18
3.1 Hardware Integration . . . 18
3.2 Parameters Definition . . . 19

4 Evaluation 22

Conclusions and Further Work 27
Conclusions . . . 27
Further Work . . . 27

Bibliography 29

Introduction

Background

Over the last decade, the demand for drones has increased rapidly. This technology is now used in many sectors, including the military, inspection and maintenance, consumer applications, agriculture, and energy, and is expected to continue growing, with the military market representing a major share [1]. Unmanned Aerial Vehicles (UAVs) have become essential for a wide range of civilian tasks, such as mapping, search and rescue, and payload transportation, as well as supporting tasks including border security, tactical surveillance, exploration, and real-time image transmission through airborne target tracking [2]. However, enabling fully autonomous tracking remains challenging. It involves many components and subsystems, leading to high overall system complexity. In addition, designing robust architectures capable of reliable real-time operation on board UAVs is one of the main obstacles to achieving the desired performance. One effective approach is to steer a lightweight gimbal toward the target, rather than reorienting the entire airframe [3]. This makes gimbal-based solutions particularly attractive for such applications.

Motivation

Although robust autonomous systems with vision-based target tracking are highly desirable, many state-of-the-art algorithms are computationally expensive [4, 5] and rely on powerful, costly hardware with high processing capabilities. This makes them unsuitable for drones with strict cost constraints and consequently limited on-board computational resources. There is currently little work that addresses a complete, low-cost gimbal tracking system that can adequately perform tasks not requiring high-end, state-of-the-art solutions, while using components that are realistic for real-world deployment under such constraints [5, 6]. This creates a clear need for computationally efficient tracking algorithms that can run on low-power microcontrollers. Studying the physical and mathematical limits of such systems is a key step toward bridging the gap between expensive high-performance platforms and scalable

tactical drones.

Objectives and Scope

This thesis aims to develop a low-cost active object tracking system using a gimbal to keep the object of interest centered in the image frame. The primary objective is to develop an algorithm for a gimbal-based solution under embedded hardware constraints, validate its capability in realistic scenarios, and demonstrate the overall feasibility of the system. In addition, this work seeks to provide a solid theoretical foundation and supporting experimental results that account for cost-budget limitations, while still enabling the construction of a robust and high-performance tracking system.

Thesis Structure

The remainder of this thesis is organized as follows. Chapter 1 reviews related work on system components, gimbal control, and vision-based tracking. Chapter 2 presents the proposed approach, covering the hardware selection and mathematical foundations. Chapter 3 describes the final system implementation. Chapter 4 reports the evaluation results. Chapter 4 concludes the thesis and outlines directions for further work.

1 Related Work

1.1 System Architecture

In the work [7], an architecture for an autonomous indoor photogrammetry drone is proposed. The design is built around the Raspberry Pi Zero W as the companion computer, chosen for its small size and very low cost. Because of this choice, the flight stack must run on a GNU/Linux system. ArduPilot and PX4 (Pixhawk) were considered as the two main candidates [7], and ArduPilot was selected due to its higher maturity and broad community support. A key limitation of this architecture is the limited computational budget of the Raspberry Pi Zero W. Clarke reports that ArduPilot alone consumes approximately 75 % of the CPU on average, leaving few resources for vision-based processing tasks. This constraint affects all subsequent design decisions and motivates a lightweight overall approach. Building on this low-cost system design, the present work adopts the same flight stack, configures the flight controller port parameters accordingly, and follows a similar approach for selecting the companion computer.

1.2 Gimbal Control

The pointing and tracking performance of a camera on a gimbal system is fundamentally constrained by the inner-loop dynamics, which are determined by the bandwidth of the gyroscopes and actuators, as well as disturbances induced by platform motion. To isolate gimbal-mounted sensors from external disturbances, many control strategies have been proposed. In [8], self-tuning PID-type fuzzy controllers are investigated. A comparison with a conventional PID controller on a UAV for real-time tracking shows reduced maximum and total error. In [9], a vision-based neuro-fuzzy controller is designed for a two-axis gimbal mounted on a small UAV. Despite the advantages of fuzzy logic for nonlinear systems, control performance can still degrade in the presence of internal and external nonlinear disturbances [2]. Since this work targets a low-cost system with minimal computational overhead and does not focus on inner-loop stabilization design or comparison, it employs a standard PID

controller with fixed P, I, and D gains.

The forward kinematics of the two-axis gimbal are defined using rotation matrices for each axis, where the combined transformation is obtained by multiplying the individual rotation matrices for each degree of freedom [2, 10]. This work adopts that formulation, simplified for a 2-axis gimbal system.

Besides, an iterative control algorithm that exploits frame ordering for a 2-axis gimbal is presented in [11]. The algorithm converts the image pixel error directly into a PWM position signal for each axis by multiplying it with the derived coefficients. In [10], the spatial resolution parameter *SR* is used to convert a pixel error into a metric displacement, thereby defining the sensitivity of the error. This work combines the former approach with the latter idea behind *SR*, with the modification that the error between the image plane and the target centers is directly mapped to the gimbal control signal, with the *SR* derived from the initial tracking frame and the camera parameters.

1.3 Vision-Based Tracking

Active tracking requires object detection followed by measuring the deviation of the detected object from its desired position. To accomplish this, various methods are used, including deep CNN-based detectors such as YOLOv3 and SSD [10, 12], as well as classical techniques such as color thresholding and contour extraction [3].

While CNNs offer robust detection performance, they are computationally expensive and typically require GPU acceleration. For resource-constrained UAVs, lightweight correlation-filter-based 2D trackers are preferable, as they operate efficiently without overloading the CPU [12]. Accordingly, this work adopts such lightweight trackers.

In [12], the problem is formulated as 3D attitude tracking on the SO(3) group, representing the orientation error as a 3D rotation matrix and designing a Lyapunov-based controller. By contrast, [3, 10] compute the visual control error directly in the image plane by subtracting the pixel coordinates of the object's center from the image center. Given that the main objective of this work is low-cost system implementation on a UAV, it adopts this simple 2D geometry-based error formulation.

State-of-the-art UAV vision systems often rely on high-end embedded GPUs such as the NVIDIA Jetson series to execute computationally demanding convolu-

tional neural networks [10, 12]. In contrast, the selected platform provides sufficient resources to run lightweight, 2D correlation-filter trackers. Specifically, this work adopts MOSSE [13] and CSRT [14], which represent two extremes of the speed–accuracy trade-off among the single-object trackers available in the OpenCV library. Empirical benchmarks on the OTB-100 dataset confirm that MOSSE is the fastest OpenCV tracker, whereas CSRT achieves the highest precision and bounding-box overlap at the cost of lower throughput [15]. Comparable results have been reproduced on Raspberry Pi hardware, confirming that these performance characteristics hold for the target platform class.

OpenCV is adopted as the vision framework not only because it provides well-tested implementations of both trackers, but also because it offers a broad ecosystem of functionality required by the overall system, including real-time video capture and frame manipulation. This tight integration within a single library simplifies the software architecture.

1.4 Camera Resolution

The choice of camera operating resolution in resource-constrained UAV platforms is driven primarily by computational and bandwidth limitations rather than by the native sensor capability.

In [7], Clarke employs the Raspberry Pi Camera Module v2.1, which houses a Sony IMX219 sensor with a native resolution of 3280×2464 pixels. The experiments show that reducing the resolution from 1920×1080 to 640×480 at 15 frames per second (FPS) rate drops the output traffic from over 1 000 KB/s to approximately 300 KB/s at a bitrate of 1.5 Mbit/s.

This work adopts a similarly reduced operating resolution for the tracking camera. A lower resolution not only conserves bandwidth but, critically, reduces the per-frame processing time for efficient object tracking.

2 Approach

Since the primary focus of this work is on developing an algorithm for a low-cost gimbal tracking system, every hardware and software choice is driven to minimize computational load within the capabilities of chosen components. The mathematical foundations presented in the subsequent sections are therefore formulated with these constraints in mind.

2.1 Companion Computer

The companion computer is responsible for the entire vision pipeline: capturing frames from the camera, running the object tracker, computing the visual error signal, and issuing gimbal commands over a serial link through the flight controller.

In [7], Clarke's system is built around the Raspberry Pi Zero W, showing that lightweight vision tasks can run on very limited hardware. The system in this work is more demanding because of the added tracking and control overhead. Accordingly, two less constrained candidate single-board computers were evaluated to fit additional computations — a Raspberry Pi 3 (2 GB RAM) and a Raspberry Pi 4 (4 GB RAM). Early experiments on the Raspberry Pi 3 revealed that the limited memory and CPU throughput could not sustain pipeline simultaneously. The Raspberry Pi 4, with its faster processor and larger memory, is therefore adopted for all subsequent work.

2.2 Flight Controller

The flight controller serves as a communication bridge between the companion computer and the gimbal actuators. In the architecture, it transmits serial commands generated by the companion computer to the gimbal's motor driver, and forwards telemetry data back to the companion computer over MAVLink. The selected board is the SpeedyBee F405 V4, a widely available and community-supported autopilot. It provides multiple UART ports for simultaneous MAVLink communication and gimbal control, as well as dedicated PWM outputs used to drive the gimbal motor channels.

2.3 Camera

The camera provides the visual input from which the tracking error is derived. This work employs the IMX378-based Waveshare 12MP USB camera module, a low-cost consumer-grade sensor that follows a similar selection consideration to the Raspberry Pi Camera Module v2.1 (Sony IMX219) used by Clarke [7]. Both are inexpensive, compact modules from the Sony IMX family designed for integration with single-board computers. However, the IMX378 offers improved specifications across key parameters. The key parameters comparison of both sensors is provided in Table 2.1.

Table 2.1: Comparison of key camera sensor parameters

Parameter	**IMX219 [7]**	**IMX378 [16]**
Sensor	Sony IMX219	Sony IMX378
Sensor resolution	3280×2464 px	3840×3032 px
Pixel size	$1.12\,\mu\text{m} \times 1.12\,\mu\text{m}$	–
Focal length	3.04 mm	4.7 mm
Interface	CSI (ribbon cable)	USB 2.0 (Type-C)

The USB 2.0 interface simplifies connectivity with the Raspberry Pi 4, avoiding the ribbon-cable routing constraints of the CSI-based camera modules and allowing more flexible physical placement on the gimbal platform.

Despite the sensor's native capability, the camera is operated at a significantly reduced resolution to fit the computational constraints of the low-cost companion computer. As with [7], the operating resolution 640×480 pixels is chosen to ensure that the tracking pipeline runs in real time on the hardware.

2.4 Gimbal

The gimbal used in this work is a low-cost, AGM-based two-axis brushless unit originally designed for action-camera stabilisation and dating from 2013 with techincal specifications listed in Table 2.2. It is driven by a proprietary 8-bit microcontroller and exposes basic PID tuning through a minimal desktop GUI.

The deliberate choice of an older, inexpensive gimbal serves as a representative worst-case hardware baseline to test of the system's architectural assumptions before

committing to more expensive components.

The gimbal operates with two control loops. The inner loop is responsible for inertial stabilisation and is driven by a PID controller, using feedback from the on-board IMU. The outer loop is the vision-based tracking loop implemented on the companion computer, which computes the visual error signal and issues angular-rate commands to the gimbal. This work focuses primarily on the design and evaluation of the outer loop. However, the inner loop must be adequately pre-set for the overall system to function correctly. In the present setup, the PID parameters of the inner loop remain at their factory defaults, as the proprietary configuration GUI is no longer compatible with current operating systems and could not be used for tuning. Consequently, the gimbal's stabilisation response is slower than what the hardware could achieve with optimised gains, representing a known limitation of the current prototype.

Table 2.2: Technical specifications of the AGM-based 2-Axis Brushless Gimbal

Parameter	**Value**
Supported receivers	DSM2 / DSMJ / DSMX
Supported remote interfaces	PPM / PWM / 2.4 GHz
Working voltage	DC 7.4 V – 14.8 V (recommended 12 V, 3S Li-Po)
Working current	100 mA – 500 mA
Operating temperature	−15 °C to 65 °C
Processor	High-speed 8-bit MCU
Sensors	3-axis MEMS gyroscope, 3-axis MEMS accelerometer
Maximum angular rate	2000 °/s
Maximum acceleration	16 G
Control frequency	2 kHz
Motor drive frequency	32 kHz
Control accuracy	0.1°
Control angle range (roll)	−45° to +45°
Control angle range (pitch)	−60° to +60°
Pose solver algorithm	Decoupled EKF for brushless gimbal drive
Compatible camera	GoPro Hero 3 (and similar form-factor)

2.5 Development Environment

Because the companion computer is a device mounted on the UAV airframe, all software development and debugging is carried out remotely from a separate work-

station. Two complementary remote-access methods are employed. For tasks that require a full graphical desktop — such as visualising tracker output in real time or inspecting camera feeds — NoMachine is used to provide a low-latency remote-desktop session over the local network. For code editing, terminal access, and general development work, a Visual Studio Code Server instance runs on the companion computer, allowing the developer to connect from the VS Code IDE on the workstation through a standard SSH tunnel. This combination offers the responsiveness of a local development experience while keeping all execution on the target hardware, ensuring that performance profiling and resource measurements reflect the true on-board conditions.

Source code is managed with Git and hosted on GitHub, providing version control, issue tracking, and a clear history of changes throughout the iterative development process. The repository is structured so that configuration files, tracker parameters, and experiment scripts are separated from the core control logic, facilitating reproducibility of experiments. The software stack is built on Python 3.12, chosen for its extensive ecosystem of scientific and computer-vision libraries.

2.6 Gimbal Lock

Gimbal lock is a problem that can arise during rotations about gimbal axes. It occurs when the system loses one degree of freedom, causing rotations about different axes to become indistinguishable.

The gimbal used in this work follows a 2-axis reduced XY (roll–pitch) rotation order. Due to the nature of gimbal lock, this problem cannot occur in the present system for two main reasons.

First, the gimbal considered here has one rotational degree of freedom and one image-stabilization degree of freedom, as proved in Section 2.8. A complete loss of degrees of freedom would render the gimbal immovable. In this configuration, however, changes in image orientation about the roll axis do not affect rotation about the pitch axis, ensuring that motion remains possible.

Second, gimbal lock arises only when one of the gimbal's rotational axes reaches a rotation of 90°. Figure 2.1 illustrates this issue for a ZXY rotation order, where Z denotes the yaw axis and circles denote rotation of the object about the corresponding axes; the yaw axis is not present in the system. According to the

technical specifications in Table 2.2, the controllable pitch angle never reaches this critical value. Consequently, the rotational space cannot be compressed to a lower dimension in this setup, and the system retains the ability to reach the desired orientations. Therefore, explicit handling of the gimbal lock problem is not required in the current work.

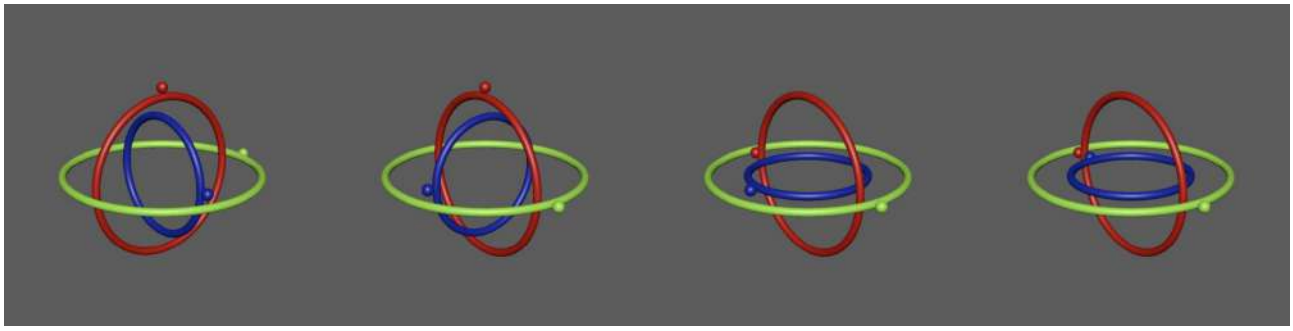

Figure 2.1: Gimbal lock under the ZXY (green-red-blue) rotation order [17]

2.7 Power Distribution System

The system components are interconnected through a distributed power architecture that relies on a single LiPo 6S battery as the unified power source. Rather than employing separate power supplies for each subsystem, the architecture routes the battery output through a central power distribution system (PDS) that partitions it into the appropriate voltage rails. This single-source design keeps the wiring minimal and makes the platform straightforward to modify or extend.

From the PDS, the flight controller is powered directly via its VCC input at the full battery voltage. The gimbal requires a regulated 12 V supply, which is provided by a dedicated Battery Eliminator Circuit (BEC) that steps down the battery voltage accordingly. A second BEC provides a stable 5 V at up to 3 A for the Raspberry Pi 4, delivered through its USB-C power input. This separation of voltage rails ensures robust electrical isolation between the system components.

2.8 Mathematical Foundations

To control the position of the gimbal, this work is adopted rotation matrices convention for the gimbal system [2, 10]:

$$\mathrm{R}(\theta_1) = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$\mathrm{R}(\theta_2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 \\ 0 & \sin\theta_2 & \cos\theta_2 \end{bmatrix} \tag{2.2}$$

$$\mathrm{R}(\theta_3) = \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 \\ 0 & 1 & 0 \\ -\sin\theta_3 & 0 & \cos\theta_3 \end{bmatrix} \tag{2.3}$$

For a 3-axis gimbal, the full rotation from the base frame to the camera frame is obtained by multiplying all three matrices in the yaw-roll-pitch order [2]:

$$\mathrm{R}_3 = \mathrm{R}(\theta_1) \cdot \mathrm{R}(\theta_2) \cdot \mathrm{R}(\theta_3) \tag{2.4}$$

Expanding (2.4):

$$\mathrm{R}_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix} \tag{2.5}$$

where the shorthand notation $c_i = \cos\theta_i$ and $s_i = \sin\theta_i$ for $i = 1, 2, 3$ is used for compactness.

For the 2-axis gimbal, the yaw angle is absent ($\theta_1 = 0$). Substituting into (2.5):

$$\mathrm{R}_2 = \mathrm{R}_2(\theta_2, \theta_3) = \mathrm{R}_3\big|_{\theta_1=0} = \mathrm{R}(\theta_2) \cdot \mathrm{R}(\theta_3) = \begin{bmatrix} c_3 & 0 & s_3 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix} \tag{2.6}$$

In [2, 10], the base frame of gimbal is differently oriented with identical rotation conventions.

In the system presented in this work, the optical axis of the camera is aligned with the roll axis at the initial position. The roll and pitch motors rotates around X and Y axes respectively as shown in Figure 2.2. The pitch axis is perpendicular to the optical axis, and the absent yaw axis (Z) points up.

Consequently, the optical axis direction $\mathbf{n}_0$ is given by the first column of R_2:

$$\mathbf{n}_0 = \mathrm{R}_2\,\mathbf{e}_1 = \begin{bmatrix} c_3 \\ s_2 s_3 \\ -c_2 s_3 \end{bmatrix} \tag{2.7}$$

At initial position $\mathbf{n}_0 = \mathbf{e}_1 = [1,\ 0,\ 0]^\top$, when $\theta_2 = \theta_3 = 0$, confirming the camera looks along X.

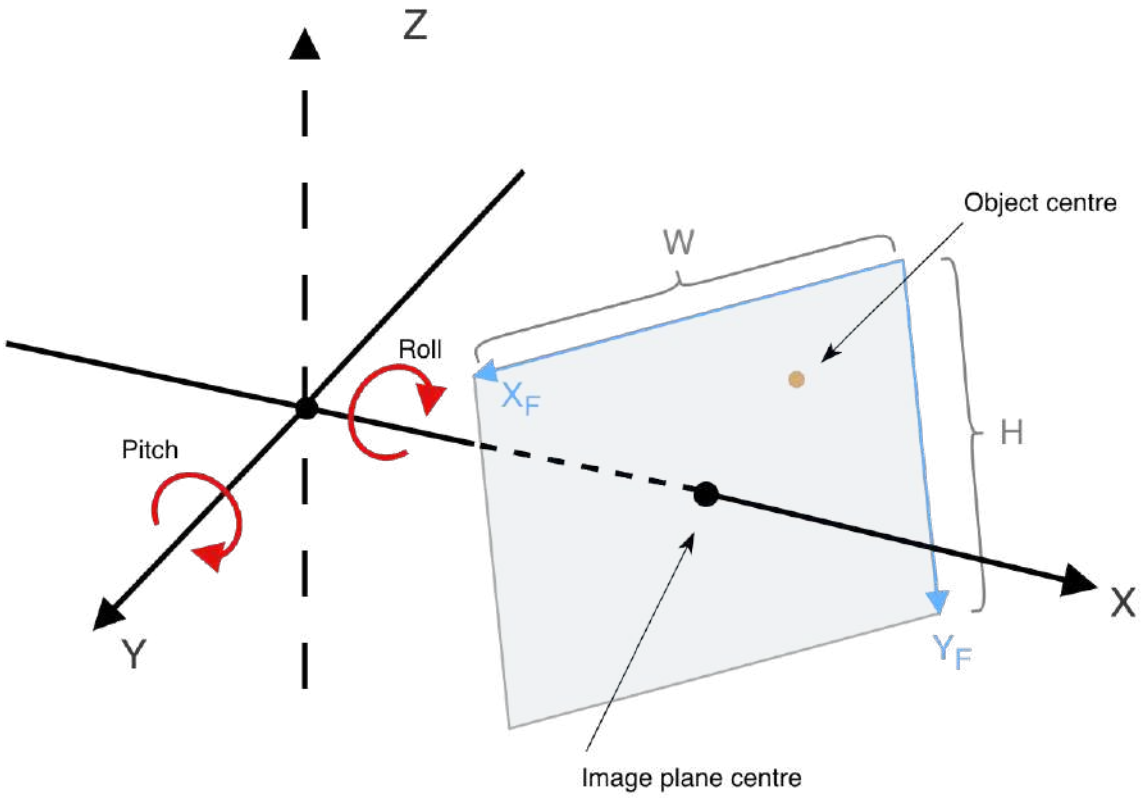

Figure 2.2: Coordinate system for gimbal

For a target at position $\mathbf{q} = [q_x\ q_y\ q_z]^\top$ relative to the gimbal base (with X forward, Y left, Z up), the pointing condition requires $\mathbf{n}_0 = k\,\mathbf{q}$ for some scalar $k \in \mathbb{R}$.

From (2.7), this gives the system of equations:

$$\begin{cases} c_3 = k\, q_x \\ s_2 s_3 = k\, q_y \\ -c_2 s_3 = k\, q_z \end{cases} \tag{2.8}$$

From (2.8), dividing the second equality by first yields:

$$\sin\theta_2 \tan\theta_3 = \frac{q_y}{q_x} \tag{2.9}$$

Assuming a practically feasible pitch angle $|\theta_3| < \pi/2$ and a target in front of the gimbal ($q_x > 0$), and noting that $|\sin\theta_2| \leq 1$, equation (2.9) can hold only if

$$|\tan\theta_3| \geq \left|\frac{q_y}{q_x}\right|. \tag{2.10}$$

Since tangent function is monotonically increasing on $(-\pi/2, \pi/2)$, this is equivalent to

$$|\theta_3| \geq \arctan\left|\frac{q_y}{q_x}\right|. \tag{2.11}$$

Taking the limit as the lateral offset grows large relative to the longitudinal one,

$$\lim_{|q_y/q_x|\to\pm\infty} \arctan\left|\frac{q_y}{q_x}\right| = \frac{\pi}{2}, \tag{2.12}$$

so the required pitch angle approaches the boundary of its feasible range. In other words, as the target moves laterally, the pointing condition demands an increasingly extreme pitch angle, which the mechanism cannot supply.

This exposes a structural limitation of the configuration. The roll axis rotates the camera about its own viewing direction, reorienting the image without redirecting it. The gimbal therefore has one pointing degree of freedom (pitch) and one image-stabilization degree of freedom (roll).

However, the roll motor can rotate the image plane around the optical axis, which shifts the projection of the target within the image. A target at position $\mathbf{q}$ in the world frame has coordinates in the camera frame [12]:

$$\mathbf{q}_C = \mathrm{R}_2^\top\, \mathbf{q} \tag{2.13}$$

Expanding using (2.6):

$$\mathbf{q}_C = \begin{bmatrix} q_x c_3 + q_y s_2 s_3 - q_z c_2 s_3 \\ q_y c_2 + q_z s_2 \\ q_x s_3 - q_y c_3 s_2 + q_z c_2 c_3 \end{bmatrix} \tag{2.14}$$

From (2.14), the lateral centering condition is:

$$q_y \cos\theta_2 + q_z \sin\theta_2 = 0 \tag{2.15}$$

Consider the target with no vertical offset $q_z = 0$ which has lateral position offset only. Equation (2.15) reduces to:

$$q_y \cos\theta_2 = 0 \tag{2.16}$$

Since $\cos\theta_2 \neq 0$ for any possible gimbal's roll angle ($|\theta_2| < \pi/2$), equation (2.16) is satisfied only when $q_y = 0$. The roll axis is therefore unable to compensate for any lateral offset, and this constitutes the fundamental limitation of the 2-axis configuration.

2.9 Motor Command Formulation

A dead zone on the pixel error is the standard approach for preventing continuous small fluctuations around the setpoint and gimbal overheating [18]. The gimbal is driven by PWM signals sent from the Raspberry Pi, via the flight controller, to the gimbal control board. The motor accepts PWM values in the range 1000–2000 μs, with 1500 μs corresponding to the neutral position.

The motor position command is defined by an iterative update of the previous PWM signal:

$$p_{i+1} = \begin{cases} p_i, & \text{if } |e_i| \leq \delta \\ \max\left\{1000,\ \min\left\{2000,\ p_i + SR \cdot (c_i - \frac{N}{2})\right\}\right\}, & \text{otherwise} \end{cases} \tag{2.17}$$

for tracking cycle iterations $i = 0, 1, 2, \ldots$, where c_i is the detected target-center coordinate along the corresponding axis in pixels (see Figure 2.3), $e_i = c_i - N/2$ is the per-axis pixel error, N is the image dimension along the same axis, and SR is the

spatial resolution coefficient (μs/px). The initial pulse width is set to any value from the acceptable range.

Before each PWM update, the magnitude of the per-axis pixel error is compared against a configurable dead zone threshold δ. While $|e_i| \leq \delta$, the PWM command is held on that axis and the corresponding motor remains at its current commanded position. Otherwise, the command is updated from the current pixel error and clipped to the operational range. A single threshold is shared between the two axes. For the expected small values of δ, the camera field of view (FOV) maps equal pixel margins to nearly equal angular margins.

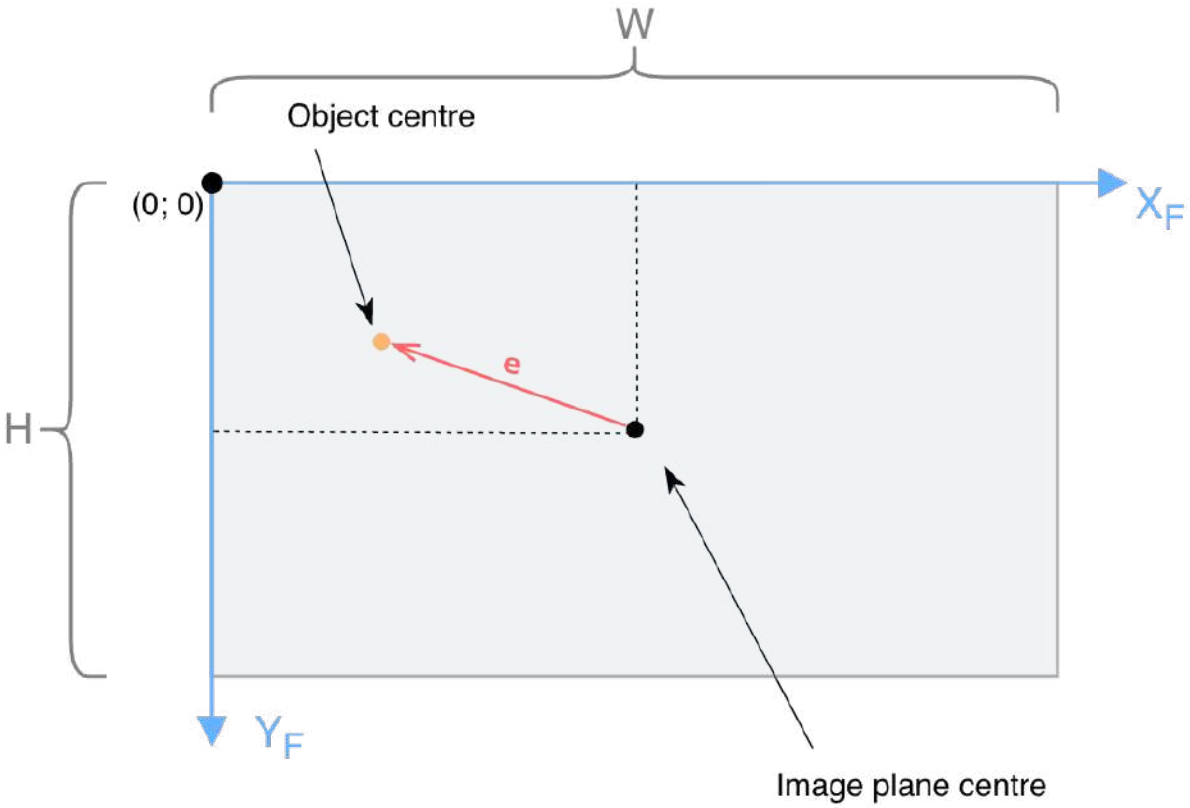

Figure 2.3: Image coordinate system

The optimal value of *SR* can be derived directly from the camera and gimbal geometry, without reference to target distance or empirical tuning. The camera employs a rectilinear lens with manufacturer-specified diagonal FOV $\gamma_d = 79.8°$ and distortion $< 0.39\%$ [16], and the object detector operates on the reduced operating image resolution of $W \times H = 640 \times 480$ pixels. Under the pinhole camera model [19, 20], the horizontal, vertical, and diagonal FOV are related to the corresponding pixel

extents by

$$\tan\left(\frac{\gamma_h}{2}\right) = \frac{W}{2f}, \qquad \tan\left(\frac{\gamma_v}{2}\right) = \frac{H}{2f}, \qquad \tan\left(\frac{\gamma_d}{2}\right) = \frac{D}{2f} \tag{2.18}$$

where γ_h, γ_v, and γ_d are the horizontal, vertical, and diagonal FOV, W and H are the image width and height in pixels, D is the image diagonal in pixels, and f is the focal length. In subsequent derivation, f is measured in pixel units rather than in millimeters regarding image frame expression in pixels.

From (2.18), f can be derived directly:

$$D = \sqrt{W^2 + H^2} = \sqrt{640^2 + 480^2} = 800 \text{ px} \tag{2.19}$$

$$f = \frac{D}{2\tan(\gamma_d/2)} = \frac{800}{2\tan 39.9^\circ} \approx 478.4 \text{ px} \tag{2.20}$$

Substituting back into the horizontal and vertical relations in (2.18):

$$\gamma_h = 2\arctan\left(\frac{W}{2f}\right) \approx 67.6^\circ, \qquad \gamma_v = 2\arctan\left(\frac{H}{2f}\right) \approx 53.3^\circ \tag{2.21}$$

Proposal 1. *Let $\Delta_{\max}$ be the full control angle range of a gimbal axis, $\Delta p_{\max}$ be the full PWM command span, and γ and N denote the FOV and pixel extent of the image along that same axis. Under the pinhole camera model (2.18), the spatial resolution coefficient that drives the pixel tracking error to zero in a single step is*

$$SR = \frac{2\tan(\gamma/2) \cdot \Delta p_{\max}}{N \cdot \Delta_{\max}} \tag{2.22}$$

where $\Delta_{\max}$ is expressed in radians.

Proof. Consider a pixel error $e = c - N/2$ along the axis, where c is the object center coordinate on that axis, and N is the image extent along that axis. The angle γ_e between the target ray and the optical axis satisfies the pinhole relation (2.18):

$$\tan(\gamma_e) = \frac{e}{f}, \qquad \tan(\gamma/2) = \frac{N/2}{f} \tag{2.23}$$

Dividing these two relations eliminates f and yields a direct geometric dependence between the two angles:

$$\frac{\tan(\gamma_e)}{\tan(\gamma/2)} = \frac{e}{N/2} = \frac{2e}{N} \tag{2.24}$$

Since the pinhole relation (2.18) is monotonic, $\gamma_e \to 0$ as $e \to 0$, so the angle γ_e becomes arbitrarily small as the tracker drives the pixel error toward the image center. The half-FOV $\gamma/2$, in contrast, is a fixed camera property. The Maclaurin expansion of the tangent function [21] is

$$\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + O(x^7) \tag{2.25}$$

so that the first-order approximation $\tan(\gamma_e) \approx \gamma_e$ holds, with residual vanishing as $e \to 0$. Solving (2.24) for γ_e:

$$\gamma_e = \frac{2e \cdot \tan(\gamma/2)}{N} \tag{2.26}$$

A PWM correction Δp rotates the gimbal axis by

$$\Delta = \Delta p \cdot \frac{\Delta_{\max}}{\Delta p_{\max}} \tag{2.27}$$

Single-step convergence requires that a pixel error e commands a PWM change $\Delta p = e \cdot SR$ rotating the gimbal by exactly θ_e that is $\Delta = \gamma_e$. Equating (2.26) and (2.27), and solving for SR, yields (2.22). □

3 Solution

3.1 Hardware Integration

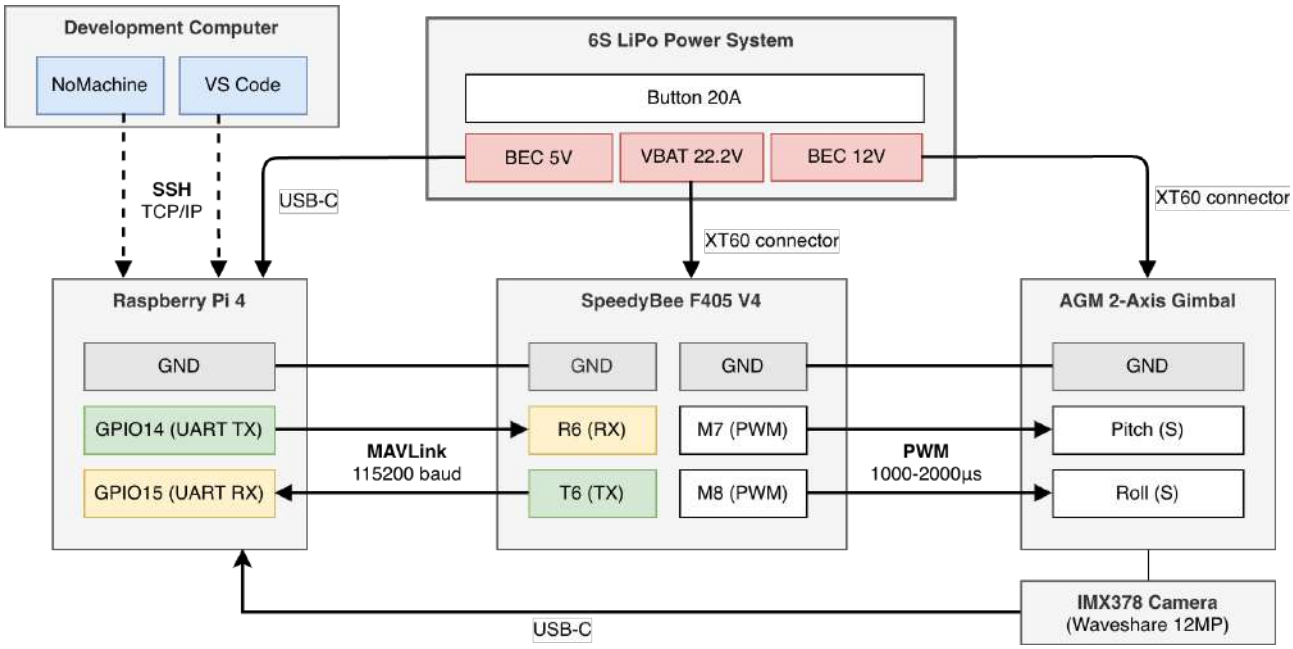

Figure 3.1: System design

As shown in Figure 3.1, the final system design consists of the following physical components:

- Raspberry Pi 4 — companion computer;
- SpeedyBee F405 V4 — flight controller;
- AGM 2-axis gimbal;
- IMX378 camera;
- Power system.

The power system uses an independent 6S lithium polymer battery as the sole energy source. A 6S LiPo provides a nominal voltage of 22.2 V. A mini power distribution hub splits this input into regulated rails for each subsystem. The hub's 5 V BEC output supplies the Raspberry Pi 4, which expects a 5 V / 3 A input. The

12 V BEC output powers the gimbal, whose specifications call for 7–14 V / 0.5 A (3S–4S equivalent). The flight controller accepts the battery voltage directly through its VBAT input, rated for 3S–6S LiPo [22]. The gimbal and flight controller are connected to the PDS via XT60 connectors, and the Raspberry Pi 4 is connected via a USB-C cable.

The companion computer transmits gimbal commands to the flight controller at 20 Hz via MAVLink. The frequency is chosen during the development as it guarantees capability of the gimbal system.

Despite default PWM update rate on the flight controller of 50 Hz [23], the higher or lower frequencies lead to lagging during remote image transmitting or to increasing one tracking cycle delay as described in more details in Chapter 4.

The companion computer communicates with the flight controller via the MAVLink protocol over a serial connection at 115 200 baud. The Raspberry Pi's UART TX and RX lines are wired to the R6 and T6 pads on the flight controller, which is configured as a MAVLink endpoint. The flight controller maps the specified channel values to servo outputs and generates the corresponding PWM signals on the assigned output pads. The gimbal's pitch and roll motors are connected to the M7 and M8 output pads respectively.

The IMX378 camera is mounted on the gimbal's camera frame using weight plates to match the payload mass expected by the gimbal's internal stabilization controller. The camera is connected directly to the Raspberry Pi via a USB-C cable.

The resulting physical layout of the components on the UAV frame is shown in Figure 3.2.

Since this work focuses on algorithm development for a low-cost gimbal-based system, the final setup is implemented as a static physical layout, without flight capability.

3.2 Parameters Definition

As motivated in Section 1.4, the vision pipeline operates on a 640×480 frame stream captured directly from the camera over USB-C.

Pretrained MOSSE and CSRT single-object trackers are integrated. Both are correlation-filter implementations that run entirely on the CPU regarding computational constraint on the Raspberry Pi 4. In the implementation, they share common

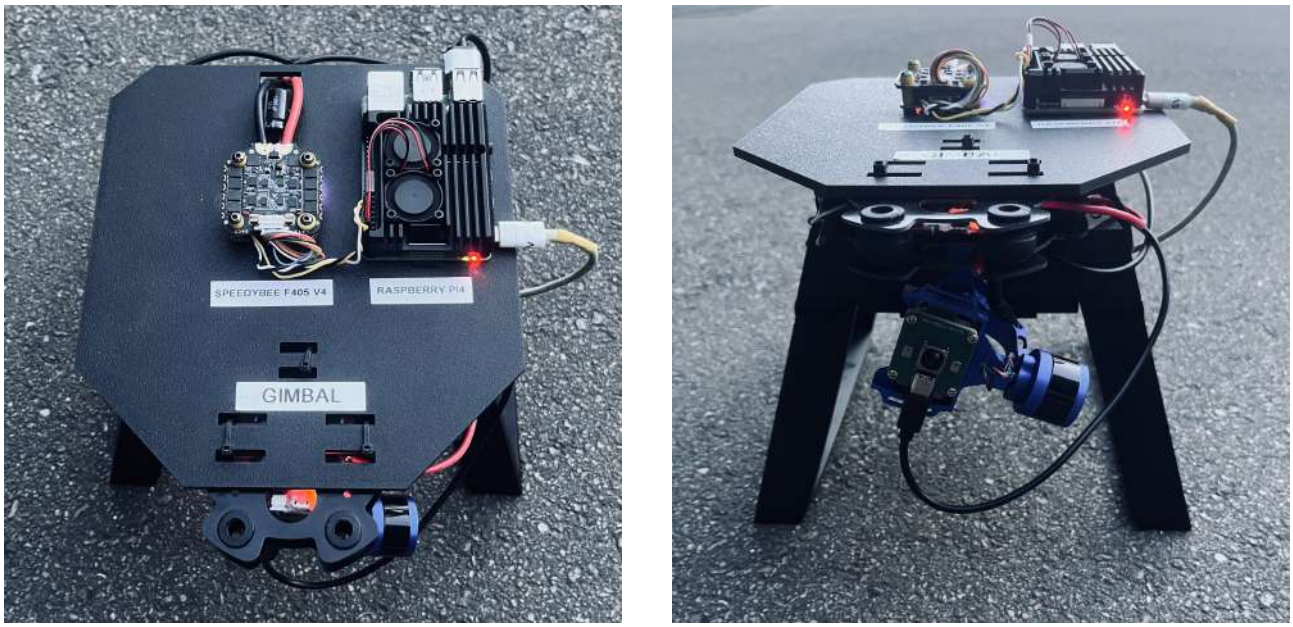

Figure 3.2: System physical layout

initialization, so the active tracker can be switched at startup through a configuration flag without changing any other component of the pipeline.

The pixel error is converted into a PWM signal for each gimbal axis through the spatial resolution coefficient introduced in Section 2.9. From Table 2.2, the gimbal's control angle ranges are ±60° and ±45° on pitch and roll axes accordingly, giving full ranges 120° ≈ 2.094 rad and 90° ≈ 1.571 rad. Both ranges have identical the full PWM command span $\Delta p_{\max} = 1000\ \mu$s. Pairing each axis with its corresponding image extent and FOV from (2.21) gives:

$$SR_{\text{roll}} = \frac{2\tan(33.8°) \cdot 1000}{640 \cdot 1.571} \approx 1.33\ \mu\text{s/px} \tag{3.1}$$

$$SR_{\text{pitch}} = \frac{2\tan(26.65°) \cdot 1000}{480 \cdot 2.094} \approx 1.00\ \mu\text{s/px} \tag{3.2}$$

These coefficients are stored as constants in the configuration file and loaded once at start. They are intentionally treated as time-invariant with no dependence on the target distance, so no continuous update is required.

Before any tracker update can run, the system must be told what to track. The target is selected interactively at start. The same selection routine can be re-triggered at any time during a run to recover from target loss or to switch to a different object without restarting the camera or rebooting the companion computer.

The dead zone is set to $\delta = 25$ px that corresponds to roughly 5% of the smaller

image dimension, a fraction which is small enough that the target still appears centered yet large enough to mask a few-pixel jitter of the tracker.

4 Evaluation

This chapter presents plot-scale testing aimed at identifying system drawbacks and validating the proposed solution with respect to its general capability and feasibility in realistic scenarios. The experiments are carried out mostly under low-wind conditions, and the reported results correspond to these conditions. All trials are performed on an experimental area of approximately 200 m^2. The system is mounted in a static position at 2 meters elevation above ground level and tested in several environmental settings. Because the default gimbal position is centered on each axis, the area directly below the gimbal lies outside the FOV at this height. To compensate, the initial motor position signal for the pitch axis is changed from 1500 μs to 1800 μs (tilting downward), while the roll axis remains at its default value, ensuring the desired initial viewing angle of the monitored area.

Figure 4.1 shows examples of camera image during live tracking. In the upper-left corner, the tracking error is displayed with respect to the X_F and Y_F image-frame coordinate system defined in Figure 2.3, together with the current object size. The gray cross marks the image-frame center, and the green region represents the object-of-interest bounds selected during the live selection step before the tracking stage.

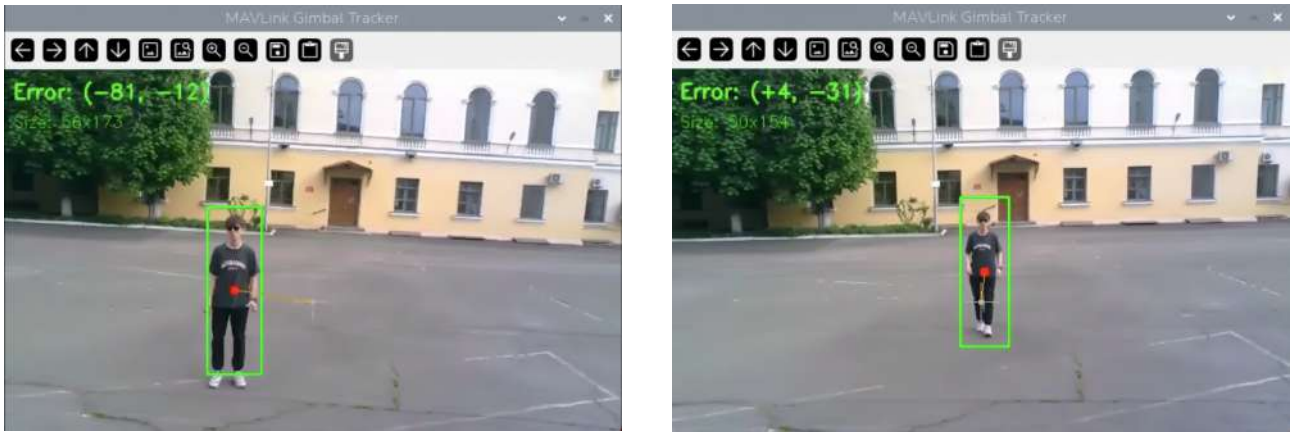

Figure 4.1: Live object tracking for CSRT

As discussed in Section 3.2, this work employs MOSSE and CSRT trackers, and both are evaluated. Overall, they demonstrate similar behavior on the control algorithm defined in Section 2.9, with good tracking performance and only occasional object loss or lag. However, MOSSE is more prone to losing distinct object boundaries when the object moves away from the camera and is particularly sensitive to rapid

motion or flicker, which can cause complete loss of the target. By contrast, CSRT tends to exhibit more lag due to the limited computational resources of the companion computer, but it typically yields more accurate and stable bounding boxes.

Each final experimental run lasts 5 minutes. The target is a human moving within the experimental area. Since both trackers exhibit similar behavior with respect to the control algorithm and share the same limitations, the final results are reported for CSRT only, as the more robust of the two. As shown in Figure 4.2, the resulting object trajectory mainly tries to affect the pitch axis with the vertical motion, which is the only rotational degree of freedom of interest in this configuration. The roll axis provides a correspondingly poor approximation of the horizontal axis error, because it is responsible for orientation change only as described in Section 2.8. The tracking error of the CSRT on each axis is plotted in Figure 4.3.

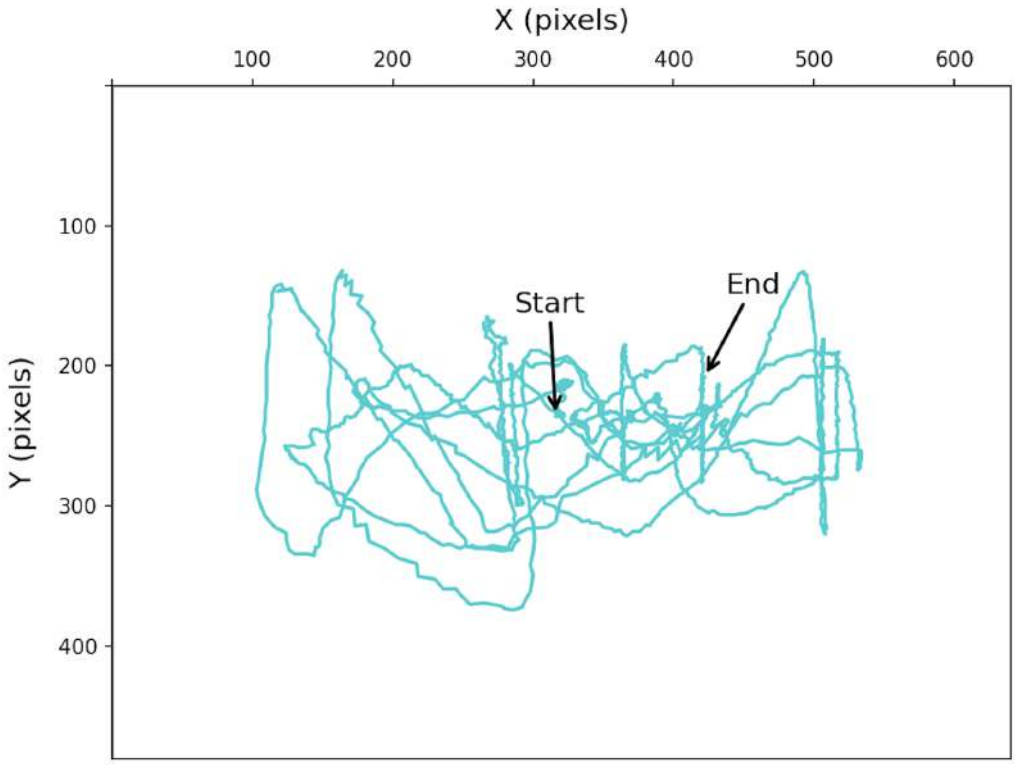

Figure 4.2: Object movement trajectory for CSRT

The effectiveness of the chosen parameters is primarily affected by the constraints described in the following paragraphs. Thus, lowering *SR* as the key parameter for signal control yields the expected centering for the pitch axis, yet with slower convergence. To validate this, in the final evaluation the derived *SR* is reduced to 0.1 based on the observed delay between the desired and actual setpoints in Figure 4.4. The experiment duration with the predefined parameters is shortened to 3 minutes to

exhibit more distinct behavior.

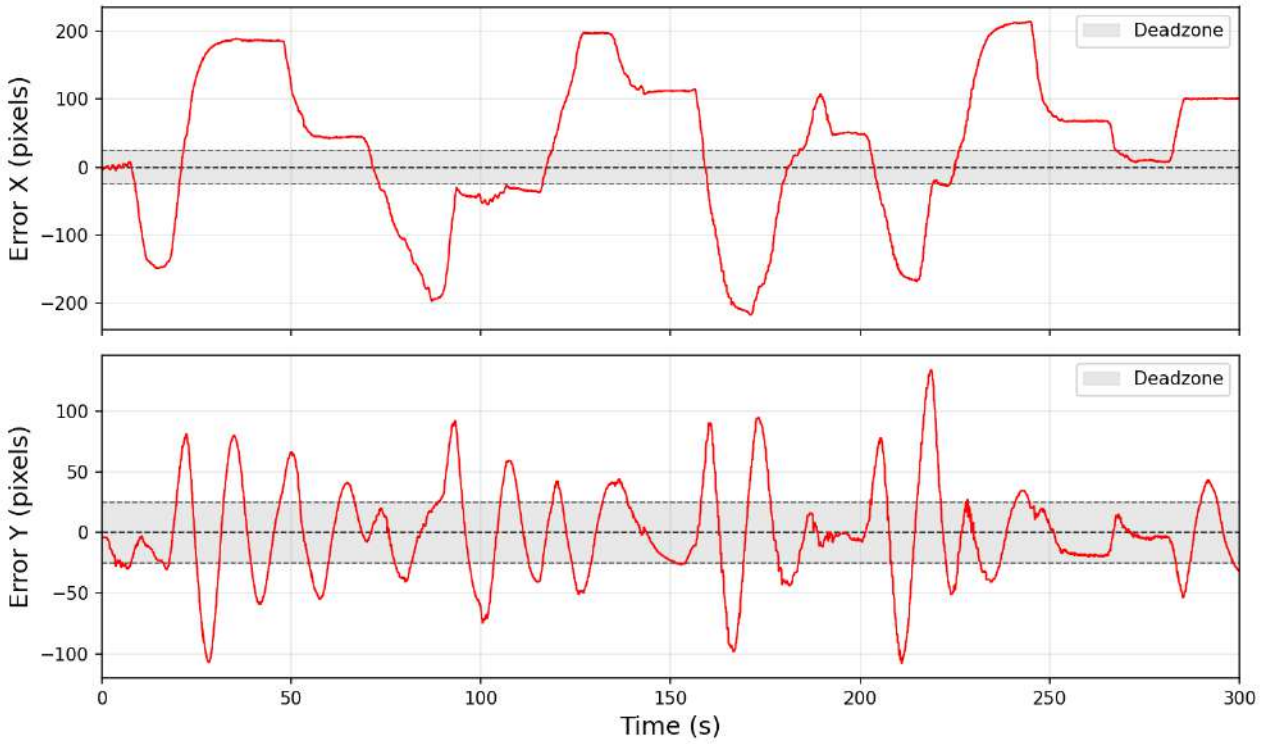

Figure 4.3: Object tracking error for CSRT on both axes with $SR_{\text{pitch}} = 0.1$

The main issue observed during testing is oscillation around the object center. In practice, the gimbal tends to remain within the upper or lower bound of the dead zone around the target but rarely points exactly at the center with predefined parameters as shown in Figure 4.4.

This behavior is primarily caused by inherent system limitations and design choices and can be attributed to two main factors.

First, the total delay of a single tracking cycle is too large to enable effective real-time tracking. As described in Section 3.1, the transmission frequency between the flight controller and the companion computer is set to 20 Hz, even though a higher frequency is theoretically possible. When combined with the time needed to process object bounds, transfer image data from the camera to the companion computer, send the processed commands from the computer to the flight controller at the selected rate, and apply the signals on the gimbal controller, the resulting end-to-end latency becomes substantial and interacts with the second issue.

Second, the gimbal speed is limited because the inner-loop PID controller of the gimbal could not be configured in the designed system, as discussed in Section 2.4. This is the case even though the manufacturer specifications claim a maximum

rotational speed of 2000°/s (see Table 2.2). Together with the mentioned latency, this limitation leads to a situation where the gimbal cannot physically reach the commanded position before a new command is issued. At each step, the control algorithm assumes that the gimbal has already reached the previously commanded position, and it updates the position signal to further reduce the error to the object center. In reality, however, the gimbal is still moving toward the previous target. As a result, the cumulative corrections in the presence of delay produce overshoot, causing the gimbal to pass beyond the object center. Once the system assumes that the calibration is complete, the gimbal has already rotated out of the dead zone, and the next control step continues the same oscillatory behavior.

A detailed analysis of the total latency and the delay introduced at each stage of the pipeline is outside the scope of this work.

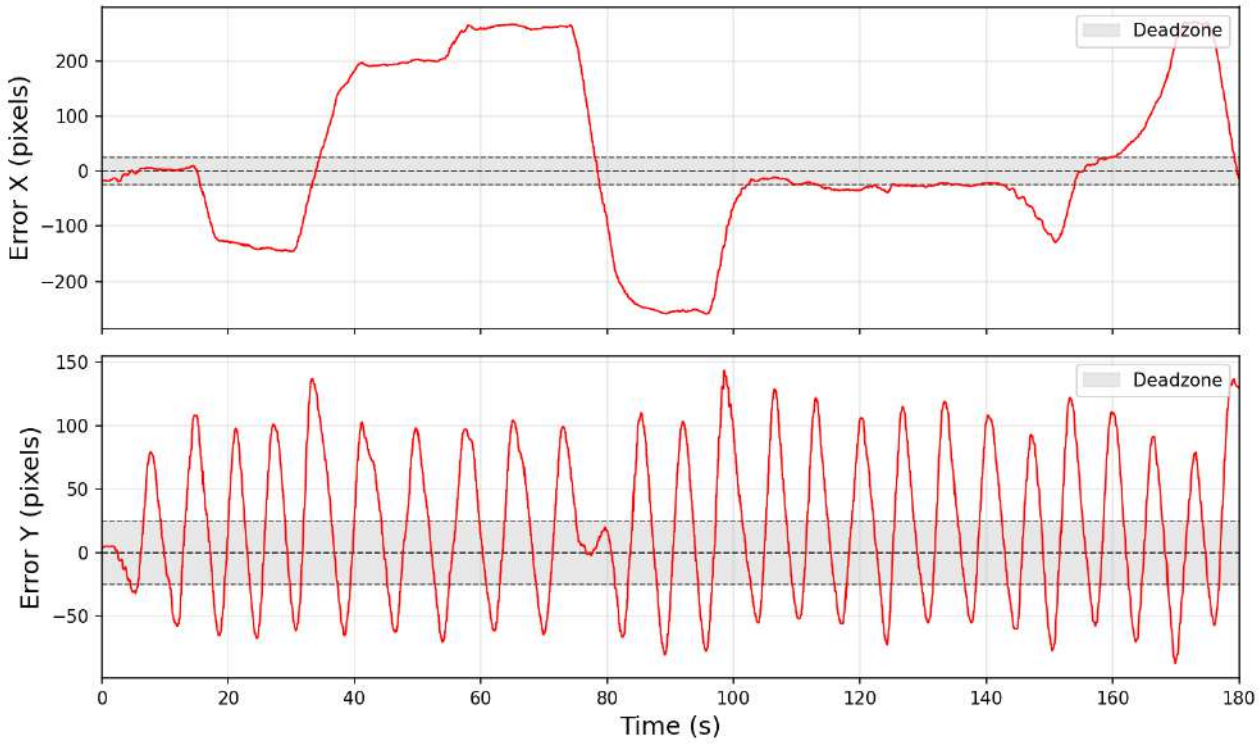

Figure 4.4: Object tracking error for CSRT on both axes with $SR_{\text{pitch}} = 1$

The final evaluation results for CSRT on the pitch axis in Table 4.1 show a clear improvement in centering as the scaling ratio is reduced. With $SR_{\text{pitch}} = 1.0$, the mean tracking error reaches 22.26 pixels with a standard deviation of 58.69, which is consistent with the oscillatory behavior observed in Figure 4.4. Reducing the spatial resolution factor to $SR_{\text{pitch}} = 0.1$ lowers the mean error to 2.32 pixels and the standard deviation to 41.80, confirming that smaller per-step corrections soften the cumulative

overshoot discussed above. The trimmed statistics, computed over the central 95% of samples to reduce the influence of brief, drastic target movements, follow the same trend and indicate that the improvement is not driven by isolated extreme values but reflects the steady-state behavior of the control loop. Although the standard deviation remains high even at $SR_{\text{pitch}} = 0.1$, this residual variability is consistent with the system limitations identified earlier and cannot be eliminated through the *SR* change alone.

Table 4.1: Pitch error statistics at different *SR*

	All samples		Trimmed (95%)	
Scale	Mean	Std	Mean	Std
0.1	2.32	41.80	1.85	34.99
1.0	22.26	58.69	21.79	55.25

Throughout development and testing, the gimbal was able to handle the applied loads without overheating or noticeable electrical spikes. During the final experiments, the FPS rate was set to 20 for both trackers. The CSRT tracker exhibited undesirably higher lag in the camera image at higher FPS rates.

To further assess computational limits and overall system performance, additional tests are performed with flight controller signal transmission frequencies of 20, 50, and 100 Hz. At 50 and 100 Hz, the Raspberry Pi 4 is unable to process the data stream in real time. After a few seconds of operation, the camera image freezes during the processing pipeline.

Conclusions and Further Work

Conclusions

This work demonstrates the feasibility of developing a low-cost gimbal-based tracking system with the proposed algorithm, despite significant embedded hardware limitations.

The experiments confirm that the system is capable of maintaining a moving human target within the camera FOV, with the CSRT tracker providing more stable bounding boxes than MOSSE at the cost of additional lag on the companion computer. The control algorithm successfully reduces the tracking error on the pitch axis, while the roll axis behaves as expected given that it controls image orientation.

The evaluation also identifies the principal drawbacks of the current implementation. The dominant issue is sustained oscillation around the object center, which arises from the interaction between the end-to-end pipeline latency and the limited achievable gimbal speed. The resulting cumulative corrections produce overshoot and prevent the gimbal from settling within the dead zone. Reducing SR partially compensates for this effect, lowering both the mean and the standard deviation of the pitch error, but it does not eliminate the underlying coupling between latency and actuator dynamics. Beyond the gimbal itself, the Raspberry Pi 4 adds a further constraint, as it cannot process the data stream at transmission frequencies above 20 Hz in real time.

These findings establish a baseline for the improvements outlined in the next section.

Further Work

The following directions are identified for future work:

Total delay of a single tracking cycle. As described in Chapter 4, the system exhibits a large total delay during tracking, which results in poor latency in gimbal movement.

Limited degrees of freedom. As described in Section 2.8, the gimbal provides only one rotational degree of freedom and one image-stabilization axis, and can therefore compensate for errors along the pitch axis only.

Gimbal speed. As described in Section 2.4, the gimbal relies on its default inner-loop calibration. Effective tracking at higher frequencies would require a newer gimbal that allows proper tuning of the PID controller parameters.

Absence of real flight testing. The final system is gimbal-based, with components selected for a real UAV platform. Building a fully flight-capable system is beyond the scope of this work — further investigation of system behavior under real flight conditions is therefore required.

Code optimization. The current implementation is written in Python and prioritizes correctness over runtime performance. General code optimization could reduce per-cycle execution time and improve tracking responsiveness.

Bibliography

[1] IDTechEx. *Drones Market Research Report*. https://www.idtechex.com/en/research-report/drones-market/1142. Accessed: 2026-03-15.

[2] A. Altan and R. Hacıoğlu. "Model predictive control of three-axis gimbal system mounted on UAV for real-time target tracking under external disturbances". In: *Mechanical Systems and Signal Processing* 138 (2020), p. 106548. DOI: 10.1016/j.ymssp.2019.106548.

[3] J. G. Hansen and R. P. de Figueiredo. "Active Object Detection and Tracking Using Gimbal Mechanisms for Autonomous Drone Applications". In: *Drones* 8.2 (2024), p. 55. DOI: 10.3390/drones8020055.

[4] A. Borne et al. *Architecture and evaluation protocol for transformer-based visual object tracking in UAV applications*. 2026. arXiv: 2603.03904 [cs.CV].

[5] X. Kang et al. "Adaptive Sampling-based Particle Filter for Visual-inertial Gimbal in the Wild". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023. DOI: 10.1109/ICRA48891.2023.10160395.

[6] X. Liu et al. "Real-Time Visual Tracking of Moving Targets Using a Low-Cost Unmanned Aerial Vehicle with a 3-Axis Stabilized Gimbal System". In: *Applied Sciences* 10.15 (2020), p. 5064. DOI: 10.3390/app10155064.

[7] M. Clarke. *An autonomous drone system, with intelligent flightplanning, to effectively undertake indoor photogrammetry*. Undergraduate dissertation, University of Nottingham, School of Computer Science. Supervised by Steven Bagley. 2018.

[8] B. E. Demir, R. Bayir, and F. Duran. "Real-time trajectory tracking of an unmanned aerial vehicle using a self-tuning fuzzy proportional integral derivative controller". In: *International Journal of Micro Air Vehicles* 8.4 (2016), pp. 252–268. DOI: 10.1177/1756829316675882.

[9] L. Edalati, A. Khaki Sedigh, M. Aliyari Shooredeli, and A. Moarefianpour. "Adaptive fuzzy dynamic surface control of nonlinear systems with input saturation and time-varying output constraints". In: *Mechanical Systems and*

Signal Processing 100 (2018), pp. 311–329. DOI: 10.1016/j.ymssp.2017.07.036.

[10] D. J. Regner et al. "Object Tracking Control Using a Gimbal Mechanism". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B1 (2021), pp. 189–196. DOI: 10.5194/isprs-archives-XLIII-B1-2021-189-2021.

[11] B. Indhu and V. P. S. Naidu. "Gimballed Camera Control for On-Point Target Tracking". In: *American Research Journal of Electronics and Communication Engineering* 1.1 (2015), pp. 1–10. DOI: 10.21694/2643-3486.19001.

[12] R. Cunha et al. "Gimbal Control for Vision-based Target Tracking". In: *Proceedings of the European Conference on Signal Processing*. 2019.

[13] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui. "Visual object tracking using adaptive correlation filters". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[14] A. Lukežic et al. "Discriminative Correlation Filter with Channel and Spatial Reliability". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4847–4856. DOI: 10.1109/CVPR.2017.515.

[15] A. Brdjanin, A. Akagic, D. Džigal, and N. Dardagan. "Single Object Trackers in OpenCV: A Benchmark". In: *International Conference on Innovations in Intelligent Systems and Applications (INISTA)*. 2020. DOI: 10.1109/INISTA49547.2020.9194647.

[16] Waveshare Electronics. *IMX378 12MP USB Camera (A): Specifications*. https://www.waveshare.com/wiki/IMX378_12MP_USB_Camera_(A). Accessed: 2026-04-20.

[17] F. Rinaldi and D. Dolci. *RBF Solver for Quaternions Interpolation*. 2020. arXiv: 2006.04686 [cs.GR].

[18] K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning*. 2nd. Research Triangle Park, NC: Instrument Society of America, 1995. ISBN: 978-1-55617-516-9.

[19] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. 2nd. Cambridge University Press, 2004. ISBN: 978-0-521-54051-3.

[20] R. Szeliski. *Computer Vision: Algorithms and Applications*. 2nd. Springer, 2022. ISBN: 978-3-030-34371-2. DOI: 10.1007/978-3-030-34372-9.

[21] M. Abramowitz and I. A. Stegun, eds. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 10th printing, with corrections; originally issued June 1964. Washington, D.C.: National Bureau of Standards, U.S. Government Printing Office, 1972.

[22] ArduPilot Dev Team. *Speedy Bee F4 V3/V4 — Copter Documentation*. https://ardupilot.org/copter/docs/common-speedybeef4-v3.html. Accessed: 2026-04-16.

[23] ArduPilot Dev Team. *RC Input and Output — ArduPilot Dev Documentation*. https://ardupilot.org/dev/docs/learning-ardupilot-rc-input-output.html. Accessed: 2026-04-16.

Declaration of generative AI use in the writing process

During the preparation of this work, the author used Claude Opus 4.6 and Grammarly only for language editing, formatting, and refining the phrasing of the text. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the content of the work.