



TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

HYBRID VISUAL NAVIGATION IN INDOOR ENVIRONMENT USING A QUADCOPTER

By:

Bhuwan Shrestha (078BAS012)
Dipesh Paudel (078BAS021)
Madhusudan Pun (078BAS026)
Prakash Poudel (078BAS030)

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF MECHANICAL
AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF BACHELOR IN AEROSPACE
ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

March, 2026

COPYRIGHT

The authors have agreed that the Library, Department of Mechanical and Aerospace Engineering, Institute of Engineering, Pulchowk Campus, may make this report freely available for inspection.

Moreover, the authors have agreed that permission for extensive copying of this project report for scholarly purposes may be granted by the supervisors who supervised the project work recorded herein or, in their absence, by the Head of the Department wherein the project report was done.

It is understood that recognition will be given to the authors of this project report and to the Department of Mechanical and Aerospace Engineering, Pulchowk Campus, Institute of Engineering, in any use of the material of this project report.

Copying or publication or other use of this project report for financial gain without written permission of the authors and approval of the Department of Mechanical and Aerospace Engineering, Institute of Engineering, Pulchowk Campus is strictly prohibited.

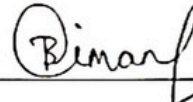
Requests for permission to copy or to make any other use of this project report in whole or in part should be addressed to:

Head of Department

Department of Mechanical and Aerospace Engineering
Institute of Engineering, Pulchowk Campus
Lalitpur, Nepal

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING, PULCHOWK
CAMPUS DEPARTMENT OF MECHANICAL AND
AEROSPACE ENGINEERING

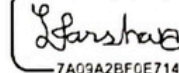
The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled "HYBRID VISUAL NAVIGATION IN INDOOR ENVIRONMENT USING A QUADCOPTER" submitted by Bhuwan Shrestha, Dipesh Paudel, Madhusudan Pun and Prakash Poudel in partial fulfillment of the requirements for the Degree of Bachelor in Aerospace Engineering.



Supervisor Asst. Prof Biman Rimal

Department of Mechanical and Aerospace
Engineering

DocuSigned by:

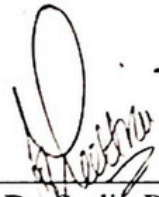


6/1/2026

7A09A2BF0E714F4

External Examiner Asst.Prof. Dr.Sarojini Darshan

Kevin T. Crofton Department of Aerospace and
Ocean Engineering



Asst. Prof. Dr. Sudip Bhattarai

Head of Department

Department of Mechanical and Aerospace
Engineering

Date: 2083/2/21

ABSTRACT

Autonomous navigation in GNSS-denied environments remains a critical challenge for aerial robots, requiring accurate localization, reliable mapping, and real-time obstacle avoidance without access to satellite positioning. This project presents the design, simulation, and hardware implementation of a hybrid visual navigation system on a custom quadcopter. The system combines VINS-Fusion for visual-inertial odometry, EGO-Planner for real-time trajectory generation and obstacle avoidance, and a non-linear geometric controller for flight stabilization. The complete navigation stack was deployed on a Raspberry Pi 4 Model B companion computer using an Intel RealSense D435 camera as the primary exteroceptive sensor. Simulation in Gazebo confirmed stable VIO-based control and collision-free flight before hardware deployment. Four real-world flight tests were conducted across indoor and outdoor environments, including runs with manually placed obstacles to test real-time replanning. The system achieved a mean 3D tracking error of 0.276 m, within the project target of 0.3 m, with an RMSE 3D of 0.358 m across all tests. Obstacle avoidance introduced negligible additional tracking error, with near-identical performance between baseline and obstacle runs. These results confirm that a complete VIO-based navigation stack with obstacle avoidance can run on a Raspberry Pi 4B and maintain sub-0.3 m mean accuracy in unstructured real-world settings.

Keywords: *Visual-Inertial Odometry, GNSS-denied Navigation, Autonomous Flight, Stereo Vision, Path Planning, Obstacle Avoidance, Embedded Systems*

ACKNOWLEDGEMENT

This project is prepared in partial fulfillment of the requirement for the Bachelor's degree in Aerospace Engineering. First and foremost, we would like to express our sincere gratitude towards Assistant Prof. Biman Rimal, our project supervisor, for his constant guidance, inspiring talks and precious encouragement. Without his invaluable supervision and suggestions, it would have been a difficult journey for us. His useful suggestions for this whole work and cooperative behaviour are sincerely acknowledged.

Special thanks to Assistant Prof. Darshan Sarojini, of Virginia Tech, whose guidance, supervision and support provided invaluable direction through regular online meetings throughout this project.

We would like to thank Assistant Prof. Dr. Sudip Bhattachai, Head of Department of Mechanical and Aerospace Engineering, and Assistant Prof. Kamal Darlami, Deputy Head of Department of Mechanical and Aerospace Engineering, for their invaluable help and support during this duration.

We would like to thank the Department of Mechanical and Aerospace Engineering, Institute of Engineering, Pulchowk Campus for providing us the opportunity of a collaborative undertaking which has helped us to implement the knowledge gained over these years as a major project for fourth year, and develop a major project of our own that has greatly enhanced our knowledge and provided us a new experience of teamwork. We would also like to thank IIEC for providing us the space to create indoor environment for flight testing.

We would also like to thank the Research Center of Low Altitude Economy (RCLAE) for providing project resources. We are equally grateful to our friends, seniors, and juniors who have directly and indirectly supported us in completing this project.

TABLE OF CONTENTS

TITLE	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	x
1 INTRODUCTION	1
1.1 Background	1
1.2 Problem Statement	1
1.3 Objectives	2
1.3.1 Main Objective:	2
1.3.2 Specific Objectives:	2
1.4 System Requirements	2
1.4.1 Hardware Requirements	2
1.4.2 Software Requirements	4
2 LITERATURE REVIEW	5
2.1 Evolution of Visual Navigation Systems	5
2.2 Research Gap	8
3 THEORETICAL BACKGROUND	10
3.1 Quadcopter Dynamics	10
3.1.1 Rotor Dynamics	11
3.1.2 Kinematics	12
3.1.3 Equations of Motion	12
3.2 Visual-Inertial Odometry (VIO)	13
3.2.1 Filter-Based VIO	14
3.2.2 Optimization-Based VIO	14
3.3 Path Planning	15
3.3.1 Global Path Planning	15
3.3.2 Local Path Planning	16
4 METHODOLOGY	18
4.1 Simulation	20
4.1.1 Quadcopter model and Indoor Environment in Gazebo	20
4.1.2 Selection and Modification of Visual Inertial Odometry Algorithm	22

4.1.3	VINS-Fusion	23
4.1.4	Simulation of VIO in Gazebo and RViZ	24
4.1.5	Selection and Modification of Path Planning Algorithm	26
4.1.6	Integration of VIO and Path Planning Algorithm	28
4.1.7	Simulation of Autonomous Navigation in Gazebo and RViz	29
4.2	Hardware and System Integration	32
4.2.1	Quadcopter Assembly and Sensor Integration	32
4.2.2	Firmware Configuration and Parameter Tuning	34
4.2.3	Sensor Calibration	35
4.2.4	Pre-Flight Testing and Validation Protocol	36
4.3	Post-Flight Data Processing and Error Computation	37
4.3.1	Data Recording and Extraction	37
4.3.2	Temporal Alignment and Frame Correction	37
4.3.3	Error Metrics	38
5	RESULTS AND DISCUSSION	40
5.1	Simulations	40
5.1.1	Open Feedback Loop Odometry Analysis	40
5.1.2	Closed Feedback Loop Odometry Analysis	41
5.1.3	Simulation of Autonomous Navigation	41
5.2	System Integration, Calibration and Initial Tests	44
5.2.1	Initial Flight Testing Using IMU	44
5.2.2	VINS Calibration Test	46
5.2.3	Hover Test	46
5.2.4	Obstacle Avoidance Test	47
5.3	Indoor Autonomous Navigation Tests	49
5.3.1	Test Environment 1: IIEC, Pulchowk Campus	49
5.3.2	Test Environment 2: Manufacturing Lab, Aerospace D2 Block	53
5.4	Outdoor Autonomous Navigation Tests: Aerospace Garden	57
5.5	Onboard System Performance	60
5.6	Summary of Navigation Accuracy	62
5.7	DISCUSSION	62
5.7.1	VIO Performance Analysis	62
5.7.2	Trajectory Tracking and Obstacle Avoidance Performance	63
5.7.3	Computational Performance	65
5.7.4	Limitations	66
5.8	Problems Faced	66
5.8.1	Challenges Encountered During Simulation	67
5.8.2	Challenges Encountered in Hardware Implementation	67

5.9	Work Timeline	68
5.10	Budget Estimation	68
6	CONCLUSION AND FUTURE ENHANCEMENT	69
6.1	Conclusion	69
6.2	Future Work	69
	REFERENCES	71
A	Python Code	72
B	Test Failures and Results	73

LIST OF FIGURES

4.1	Detailed Methodology of Project- Part I	18
4.2	Detailed Methodology of Project- Part II	19
4.3	Quadcopter with camera and its published topics	21
4.4	Virtual Environments in Gazebo	22
4.5	Open-Loop System Architecture	25
4.6	Closed-Loop System Architecture	26
4.7	Autonomous Navigation system architecture	29
4.8	Voxel Map of surrounding	30
4.9	Assembled quadcopter platform highlighting component placement . . .	32
4.10	Downward-facing attachment of the VL53L1X distance sensor	33
4.11	Forward-facing integration of the Intel RealSense D435 depth camera . .	33
5.1	Open-loop VIO estimation: (a) VINS-Fusion estimated trajectory and (b) QGroundControl mission interface.	40
5.2	RViz visualization of closed-loop VINS-Fusion odometry estimation. . .	41
5.3	Closed-loop mission results in SITL: (a) estimated trajectory in RViz and (b) VINS-Fusion odometry output	41
5.4	Simulation test environment setup of Manufacturing Lab in Gazebo . .	42
5.5	Quadcopter trajectory during mission(a) 2D top-down view with actual flight path and (b) 3D spatial trajectory commanded by the planner . . .	42
5.6	2D EGO-Planner occupancy grid: (a) raw occupancy map of obstacles detected (b) inflated occupancy map and (c) navigation overlay	43
5.7	Per-axis path tracking error during Takeoff and Navigation phase	43
5.8	Velocity tracking performance during the Navigation phase	44
5.9	Vertical state estimation during hover using internal IMU and barometer	45
5.10	Horizontal (XY) divergence during OFFBOARD-commanded flight us- ing IMU-only localization	45
5.11	Effect of IMU-camera extrinsic calibration on VINS-Fusion: (a) Before calibration and (b) After calibration	46
5.12	Altitude time series during autonomous hover at a commanded height of 1.0 m	47
5.13	Test setup at Aerospace Garden: (a) baseline configuration with no ob- stacles and (b) With obstacle configuration.	47
5.14	Comparison of 2D top-down quadcopter trajectories: (a) baseline run without obstacles and (b) run with obstacles	48
5.15	Per-axis path tracking error during Navigation phase: (a) without obsta- cles and (b) with obstacles	49

5.16	Test environment at IIEC, Pulchowk Campus	50
5.17	Quadcopter trajectory during mission at IIEC: (a) 2D top-down view with actual flight path (b) 3D spatial trajectory with altitude variations commanded by the planner	50
5.18	2D EGO-Planner occupancy grid of IIEC: (a) raw occupancy map of obstacles detected (b) inflated occupancy map and (c) navigation overlay	51
5.19	Per-axis path tracking error over the complete takeoff, navigation, and landing sequence at IIEC	51
5.20	Altitude time series during the complete autonomous flight at IIEC . . .	52
5.21	3D Euclidean path tracking error over the navigation phase at IIEC . . .	52
5.22	Velocity tracking performance over the complete flight at IIEC	53
5.23	Test environment at the Manufacturing Lab, Aerospace D2 Block	53
5.24	Quadcopter trajectory during mission at Manufacturing Lab: (a) 2D top- down trajectory (b) 3D spatial trajectory of the mission.	54
5.25	2D EGO-Planner occupancy grid of Manufacturing Lab: (a) raw occu- pancy map (b) inflated occupancy map and (c) navigation overlay	54
5.26	Per-axis path tracking error over the takeoff and navigation phases at the Manufacturing Lab	55
5.27	Altitude time series during the Navigation Phase at Manufacturing Lab .	55
5.28	3D Euclidean path tracking error over the navigation phase at the Man- ufacturing Lab	56
5.29	Velocity tracking performance during the autonomous navigation phase at the Manufacturing Lab	56
5.30	Test setup at Aerospace Garden, Pulchowk Campus	57
5.31	Quadcopter trajectory during mission at Aerospace Garden (a) 2D top- down view with actual flight path and (b) 3D spatial trajectory com- manded by the planner	58
5.32	2D EGO-Planner occupancy grid of Aerospace Garden: (a) raw occu- pancy map (b) inflated occupancy map and (c) navigation overlay	58
5.33	Per-axis path tracking error over the takeoff, navigation and landing phases at the Aerospace Garden	59
5.34	Altitude time series during the outdoor flight at Aerospace Garden . . .	59
5.35	3D Euclidean path tracking error over the navigation phase at Aerospace Garden	60
5.36	Velocity tracking performance over the complete flight at Aerospace Garden	60
5.37	System health over the whole flight duration for Triangle with obstacle .	61
B.2	Qr code for google drive results	73

LIST OF TABLES

4.1	Comparison of Visual-Inertial Odometry Algorithms	22
4.2	VINS Parameter Setup	34
4.3	Parameters Tuning for Ego-Planner	34
4.4	Pixhawk and Camera Parameter Setup	36
5.1	Onboard Computational Load During Navigation Tests	61
5.2	Summary of Navigation Tracking Performance Across All Tests	62
5.3	Project Timeline	68
5.4	Budget Estimation	68

LIST OF ABBREVIATIONS

ARM	Advanced RISC Machines
API	Application Programming Interface
BLDC	Brushless DC Electric Motor
CFD	Computational Fluid Dynamics
CoG	Center of Gravity
CPU	Central Processing Unit
DBoW	Distributed Bag of Words
DIFODO	Differential Odometry
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
ESC	Electronic Speed Controller
FPS	Frames Per Second
GICP	Generalized Iterative Closest Point
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input/Output
GPS	Global Positioning System
HybVIO	Hybrid Visual-Inertial Odometry
IMU	Inertial Measurement Unit
KLT	Kanade-Lucas-Tomasi
LiDAR	Light Detection and Ranging
LiPo	Lithium Polymer
MAV	Micro Aerial Vehicle
MEMS	Micro-Electro-Mechanical Systems
MSCKF	Multi-State Constraint Kalman Filter
NED	North-East-Down
ORB-SLAM	Oriented FAST and Rotated BRIEF SLAM
PID	Proportional-Integral-Derivative
RAM	Random Access Memory
RDVO	Robust Dense Visual Odometry
RGB-D	Red-Green-Blue Depth
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
RViZ	ROS Visualization Tool
SDF	Simulation Description Format
SITL	Software-in-the-Loop
SLAM	Simultaneous Localization and Mapping

S-MSCKF	Stereo Multi-State Constraint Kalman Filter
SPAD	Single Photon Avalanche Diodes
SPKF	Sigma Point Kalman Filter
SVO	Semi-Direct Visual Odometry
ToF	Time of Flight
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter
URI	Uniform Resource Identifier
URDF	Unified Robot Description Format
USB	Universal Serial Bus
VIO	Visual-Inertial Odometry
VINS	Visual-Inertial Navigation System
VO	Visual Odometry
V-SLAM	Visual SLAM
VTOL	Vertical Take-Off and Landing

Chapter 1

INTRODUCTION

1.1 Background

Quadcopter UAVs established themselves as foundational systems due to stable hovering. A quadcopter, also known as a quadrotor UAV, is a common type of UAV system. Quadrotors are capable of vertical take off and landing and hovering, offering more stable and flexible control compared to fixed-wing drones.

Quadrotor UAVs have become widespread in applications such as aerial photography, firefighting, and environmental monitoring due to their VTOL capability and stable hovering. However, their autonomous operation in GNSS denied environments (e.g., indoor facilities, urban canyons, or dense forests) remains a critical challenge [1]. While GNSS provides reliable outdoor positioning, its susceptibility to signal blockage and multipath errors renders it ineffective in constrained settings [2].

Standalone visual navigation techniques such as Visual Odometry and Visual SLAM leverage cameras for motion estimation but suffer from drift accumulation, sensitivity to lighting/texture changes and performance degradation in motion blur[3], [4].

To overcome these limitations, hybrid visual navigation has emerged. This approach fuses camera data with complementary sensors using probabilistic estimation frameworks like Kalman filters or factor graph optimization.

1.2 Problem Statement

Autonomous navigation in GNSS-denied environments presents persistent challenges for aerial robotics, especially for quadcopters requiring uninterrupted high-precision localization and mapping. As applications increase in GNSS obstructed settings including indoor inspection, urban search/rescue, and sub-terranean exploration existing navigation methodologies reveal critical limitations. GPS-reliant systems become inoperative where alternative positioning is most needed, while monocular visual inertial solutions struggle with computational loads and environmental sensitivity. This operational reality drives demand for hybrid navigation architectures that reconcile essential yet competing objectives: sub-meter localization accuracy, real-time mapping, and power-efficient computation compatible with the resource budgets of small UAVs.

1.3 Objectives

1.3.1 Main Objective:

To design and experimentally validate a hybrid visual-inertial navigation system on a custom quadcopter, demonstrating autonomous, collision-free flight in GPS-denied indoor environments.

1.3.2 Specific Objectives:

- i. To achieve real time Visual-Inertial Odometry (VIO) using Intel RealSense D435 and Pixhawk 4, IMU data with closed-loop positional drift below 0.3 m.
- ii. To implement a low compute 3D path planner on a Raspberry Pi 4B, enabling obstacle avoidance at flight speeds up to 0.5 m/s.
- iii. To optimize the onboard ROS architecture for synchronized sensor processing, ensuring the complete navigation stack runs consistently without exceeding on-board computational limits.
- iv. To benchmark the navigation logic in high fidelity Gazebo SITL simulations and validate the system's obstacle avoidance success rate in physical real world indoor test environments.

1.4 System Requirements

The development of an autonomous aerial vehicle capable of navigating GPS-denied environments imposes strict operational constraints regarding weight, power consumption, and computational latency. This section outlines the critical specifications and component selections required to achieve stable, real-time autonomous flight.

1.4.1 Hardware Requirements

i. Pixhawk Flight Controller

The Pixhawk 4 flight controller ensures stable quadcopter control by processing sensor data from accelerometers, gyroscopes, magnetometers, and barometers. It supports visual sensors and interfaces with the Raspberry Pi 4B for real-time odometry and SLAM. Integrated with ultrasonic or distance sensor via MAVLink, it enhances obstacle avoidance.

ii. **Raspberry Pi**

The Raspberry Pi 4B is a compact, powerful computer with a quad-core ARM Cortex-A72 and up to 8 GB RAM. It runs Raspberry Pi OS or Ubuntu to support visual odometry and SLAM. Its GPIO pins connect to visual and auxiliary sensors for mapping and obstacle detection. It interfaces with the Pixhawk 4 via MAVLink for flight coordination.

iii. **Intel RealSense Depth Camera**

The Intel RealSense D435 Depth Camera is a compact stereo sensor essential for navigation. Using dual infrared stereo sensors, it computes depth and improves motion tracking. Connected via USB to the Raspberry Pi 4B, it provides real-time data for visual odometry and SLAM, working with the Pixhawk 4 and sensors like ultrasonic or distance sensor for obstacle avoidance.

iv. **Airframe**

The airframe forms the structural backbone of the quadcopter, offering a lightweight yet durable platform for autonomous navigation. Made from materials like carbon fiber or reinforced plastic, it houses key components such as the Pixhawk 4, Raspberry Pi 4B, Intel RealSense D435, and auxiliary sensors like ultrasonic or distance sensor. Its modular design ensures balanced weight distribution and supports four efficient brushless motors for agile flight.

v. **Brushless Motor and Propeller**

Brushless motors and propellers form the quadcopter's propulsion system, enabling agile flight. Four efficient brushless DC motors, controlled by ESCs linked to the Pixhawk 4, provide precise thrust. Each is paired with lightweight, durable carbon fiber or reinforced plastic propellers optimized for lift.

vi. **ESC and LiPo Battery**

An Electronic Speed Controller (ESC) regulates brushless motor speed and direction via flight controller signals. Paired with a LiPo battery, it offers a lightweight, high-energy power source with high discharge rates, ideal for UAV electronics.

vii. **Distance Sensor (VL53L1X)**

A Time of Flight (ToF) ranging sensor that uses laser technology to detect distances precisely is the VL53L1X module. It is a multisensor module that combines a microcontroller, detector, and laser emitter. It incorporates ST's second generation FlightSense proprietary technology and a state-of-the-art SPAD array (single photon avalanche diodes). Its dimension is $(4.9 \times 2.9 \times 1.56)$ mm. It is fast and accurate long distance ranging up to 400 cm distance measurement and

up to 50 Hz ranging frequency. Its typical full field of view is 27° . It uses a single axis to measure distance in a straight line.

1.4.2 Software Requirements

i. ROS

ROS or Robot Operating System is a collection of tools and software libraries for creating robot applications. The ROS environment and its supporting libraries make it simple and quick to construct autonomous robotic systems. One of the main advantages of building robots with ROS is the collection of drivers and already existing algorithms within the system.

ii. Gazebo

Gazebo is an open source 3D robotics simulator widely used in the robotics community to simulate robots. It is a collection of open source software libraries designed to simplify the development of high performance real world simulations.

iii. MATLAB

MATLAB is a matrix based programming environment for scientific computing. It is a technical computing language designed for data visualization and high performance data analysis.

iv. Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft. It can be run on a variety of operating systems and provides support for multiple languages like Python and C/C++. It provides efficient and comprehensive tools for code editing.

Chapter 2

LITERATURE REVIEW

2.1 Evolution of Visual Navigation Systems

Accurate state estimation is fundamental to autonomous quadcopter flight. In open-sky conditions, GPS provides reliable positioning, but its signals degrade rapidly in urban canyons, under dense canopies, and indoors [5], [6]. For MAVs operating in these GPS-denied settings, onboard visual-inertial sensing has become the primary alternative. This section reviews the key developments in visual-inertial odometry, lightweight visual odometry, path planning, and integrated autonomous systems that informed the design choices made in this project.

A foundational contribution to filter-based VIO is the Multi-State Constraint Kalman Filter (MSCKF), introduced by Mourikis and Roumeliotis [5]. The algorithm derives a measurement model that expresses geometric constraints from feature observations across multiple camera poses, without including 3D feature positions in the EKF state vector. This marginalization keeps computational complexity linear in the number of features, which is a significant advantage for real-time operation on embedded hardware.

Early VIO work was predominantly monocular. The demand for greater robustness on fast-moving platforms drove the development of stereo extensions. Sun et al. [7] extended the MSCKF to stereo cameras, showing that a filter-based stereo VIO could match the efficiency of monocular methods while providing better resistance to initialization failures and scale drift. They validated the system on aggressive flight maneuvers at speeds above 4 m/s. Bloesch et al. [8] took a different approach with a direct EKF-based VIO that uses pixel intensity errors from multilevel patch features directly in the filter update. Processing takes roughly 30 ms per frame on a standard CPU, and the system requires no explicit initialization. This is an attractive property for field deployment where startup time matters.

Optimization-based methods approach the problem differently. Instead of processing measurements one at a time, they minimize a joint cost function over a sliding window of recent keyframes. The accuracy gains come from jointly refining multiple states simultaneously, at the cost of higher computation.

VINS-Mono [9] is among the most widely adopted systems in this category. It implements a tightly-coupled nonlinear optimizer that fuses preintegrated IMU measurements with KLT-tracked visual features over a window of 10–15 keyframes. The system includes an automatic initialization procedure, online temporal calibration, and a par-

allel loop-closure thread using DBoW2 descriptors. On the EuRoC benchmark, VINS-Mono achieved state-of-the-art accuracy while remaining computationally tractable.

VINS-Fusion [10] extends this framework to support multiple camera configurations, including monocular, stereo, and stereo-inertial modes within a unified optimization backend. The stereo mode eliminates the scale ambiguity inherent to monocular systems and provides more reliable depth triangulation for nearby features. A key architectural decision in VINS-Fusion is the separation of the core odometry thread from the loop-closure module. Each runs on independent threads with different priorities. If loop closure falls behind due to CPU pressure, the high-frequency odometry needed for flight stability is preserved. This modularity makes VINS-Fusion particularly relevant for deployment on resource-constrained platforms where all threads cannot run at full speed simultaneously.

Kong et al. [11] explored a complementary direction by incorporating both point and line features into a tightly-coupled stereo VINS. Using trifocal geometry with a combined EKF-SPKF framework, their system showed improved estimation accuracy in structured, low-texture environments, conditions frequently encountered in indoor corridors and industrial spaces where point features alone may be insufficient.

For platforms with severe computational limits, lightweight visual odometry methods trade accuracy for speed. DIFODO (DIFferential ODOmetry), proposed by Jaimez and Gonzalez-Jimenez [12], estimates camera ego-motion using only depth data at 60 Hz on a single CPU core. It is roughly 30 times faster than GICP and comparable in accuracy to RDVO. The catch is obvious: without inertial fusion, drift accumulates steadily over time.

The Semi-direct Visual Odometry (SVO) algorithm by Forster et al. [13] operates directly on pixel intensities, avoiding explicit feature extraction. This yields frame rates of 55 FPS on embedded computers and over 300 FPS on laptop hardware. SVO uses a probabilistic depth filter that handles outlier measurements well. However, it is vulnerable to motion blur and struggles in low-texture environments, both of which are common in indoor flight scenarios.

Full SLAM systems produce globally consistent maps but demand substantial processing power. Lightweight VO methods are fast but drift without correction. Bridging this gap, hybrid approaches fuse complementary sensors and combine different algorithmic strategies.

James et al. [14] built a multi-sensor fusion system on a Jetson Nano, combining a ZED 2i depth camera, IMU, and LiDAR. The system achieved 0.4 m navigation accuracy and 0.13 m mapping RMSE, which are reasonable numbers for a low-power

embedded platform. Nur [15] reported that IMU-LiDAR-visual fusion compensates for individual sensor weaknesses, yielding localization accuracies of 1.2 m in urban environments and 1.5 m in forests.

HybVIO [16] takes a different hybrid strategy, pairing a filtering-based VIO frontend with a loosely-coupled optimization-based SLAM backend. The system incorporates improved IMU bias modeling, outlier rejection, and stationary detection. These features directly benefit indoor quadcopter operations, where hover phases and low-speed maneuvers are common and where IMU bias stability is critical for maintaining accurate state estimates between visual updates.

Autonomous navigation demands more than accurate localization. The quadcopter must also generate collision-free trajectories in real time. Traditional gradient-based planners compute a Euclidean Signed Distance Field (ESDF) from the occupancy map to evaluate obstacle gradients, a computationally expensive step, especially when the trajectory being optimized passes through only a small fraction of the mapped space.

Zhou et al. [17] addressed this bottleneck with EGO-Planner, an ESDF-free local planner. Instead of precomputing distances for the entire map, EGO-Planner checks collisions only along the current trajectory. When a segment intersects an obstacle, the algorithm identifies a surface anchor point and a repulsive direction, then pushes the B-spline control points away from the obstacle using local gradient information. This reduces planning computation by roughly an order of magnitude compared to ESDF-based planners, making it feasible for real-time operation on embedded hardware. Liu et al. [18] earlier highlighted the importance of low-latency planning for quadrotors with limited sensing, showing that planning delays as small as 50 ms can degrade tracking performance at flight speeds above 2 m/s.

Deploying individual algorithms is one thing. Integrating them into a flight-ready system introduces additional challenges: sensor synchronization, communication latency between the companion computer and flight controller, and computational scheduling across multiple real-time threads.

Mohta et al. [19] demonstrated fast autonomous flight in GPS-denied, cluttered warehouse environments at speeds up to 8 m/s. Their system combined a stereo VIO frontend with a local planning backend on a custom quadcopter. However, their companion computer was an Intel NUC with an i7 processor, which offers roughly ten times the single-thread performance of a Raspberry Pi 4B. Whether the same algorithmic stack can run on significantly weaker hardware without compromising real-time control remains an open question.

The geometric tracking controller on $SE(3)$, formulated by Lee et al. [20], has become

a standard choice for agile quadrotor control in such integrated systems. Unlike Euler-angle-based PID controllers, it operates directly on the rotation matrix, avoiding gimbal lock singularities and providing almost-global asymptotic tracking. Computationally, it requires only matrix multiplications and cross products, making it well-suited for the tight control loops needed when tracking trajectories from reactive planners.

Taken together, these works have advanced the individual components of GPS-denied quadcopter navigation: state estimation through VIO [5], [9], [10], lightweight alternatives for constrained hardware [12], [13], multi-sensor fusion for drift reduction [14], [16], efficient planning [17], and integrated flight systems [19]. However, significant gaps remain in deploying these components as a unified system on low-cost embedded hardware, as discussed below.

2.2 Research Gap

Although the algorithms reviewed above have individually shown strong results, their integration into a complete, deployable navigation system on truly resource-constrained hardware remains underexplored.

The first gap concerns computational feasibility. Most VIO systems are benchmarked on desktop-class hardware. VINS-Fusion was evaluated on an Intel i7 processor [10], and Mohta et al. [19] used an Intel NUC for their autonomous flight experiments. Deploying these same algorithms on an ARM-based single-board computer like the Raspberry Pi 4B, a board that lacks a GPU and offers roughly one-tenth the single-thread performance of an i7, introduces computational bottlenecks that the existing literature does not characterize. Filter-based methods like MSCKF [5] are computationally lighter but suffer from unbounded drift accumulation over extended trajectories, requiring frequent recalibration or mission segmentation to maintain acceptable accuracy.

The second gap involves the separation between state estimation and path planning. Most planning work, including EGO-Planner [17], is validated using ground-truth localization from motion capture systems or external position sources. The interaction between VIO drift and planner behavior, where accumulated estimation error can cause the planner to generate trajectories through space it believes is free but is actually occupied, is rarely studied in an end-to-end context. Shah et al. [21], in a prior undergraduate project at Pulchowk Campus, demonstrated Visual SLAM on a quadcopter in indoor environments. Their work validated the feasibility of vision-based localization on similar hardware but employed a full SLAM pipeline that constrained flight speeds to approximately 0.1 m/s due to computational overhead. The integration of a reactive path planner for real-time obstacle avoidance at higher flight speeds was identified as a direction for future work in their conclusions.

The third gap is in validation methodology. Testing of complete systems remains predominantly simulation-based or confined to structured laboratories with motion capture ground truth [22], [23]. Evaluation in genuinely unstructured indoor spaces, such as manufacturing floors, storage areas, and construction sites, where lighting is uncontrolled, visual textures are repetitive, and obstacles are irregularly distributed, is notably absent from the current literature.

This project addresses these three gaps directly. We deploy a complete navigation stack consisting of VINS-Fusion for state estimation, EGO-Planner for trajectory generation and obstacle avoidance, and a geometric SE(3) controller [20] for flight control, entirely on a Raspberry Pi 4B companion computer. The system is validated through real-world flights across multiple unstructured indoor and outdoor environments at Pulchowk Campus, without reliance on GPS, motion capture, or any external positioning system.

Chapter 3

THEORETICAL BACKGROUND

3.1 Quadcopter Dynamics

Quadcopter dynamics involves numerous coupled parameters requiring systematic modeling. To model the dynamics of quadcopter some assumptions are needed to simplify the process. The assumptions are made for the forces, torques, motors, propellers, and structural components of the quadcopter. Those assumptions are:

- i. The resistance of the motors is negligible.
- ii. The quadcopter structure is rigid and symmetric.
- iii. The blade flapping, or the deformation of propeller blades brought on by high velocities and flexible materials, is neglected.
- iv. The surrounding fluid velocities, such as wind and freestream velocity, are negligible.
- v. The quadcopter's center of gravity (CoG) aligns with the origin of the body-fixed frame.

With these assumptions set, the next thing is the frame of reference. Frame of reference is a set of coordinates, such as positions and velocities of a quadcopter, in the frame being observed. There are mainly two different frame of references called the inertial and non-inertial frame of references. The inertial frame of reference is also commonly denoted as inertial frame or the Earth frame (NED: North-East-Down or ECEF: Earth-Centered, Earth-Fixed). The non inertial frame of reference is the body fixed frame, with the origin at the CoG of the quadcopter. Wind frame, Stability frame are some other frame of references. Each reference frame serves a specific purpose in dynamics and modelling of quadcopter.

The quadcopter moves by changing its thrust, where thrust is being generated by the rotation speed of the motor. In the current configuration, pitch which is the lateral motion, is achieved by changing the rotation speed of the motors 2 and 4, which causes variation in thrust forces 2 and 4. The longitudinal motion, roll is achieved by changing the rotation speed of motors 1 and 4, which causes variation in thrust forces 1 and 4. Yaw motion is generated by the difference in counter-torque between the pairs of propellers (F1, F2) and (F3, F4). This occurs when the clockwise-rotating rotors speed up while the counter-clockwise ones slow down, or the reverse. The overall thrust required for hovering or ascending is achieved by summing the individual forces produced by all the rotating propellers.

3.1.1 Rotor Dynamics

The modelling of the thrust forces and torques generated by the rotors can be represented by a simplified model. Each rotor with angular speed ω generates a force F in a direction perpendicular to the plane of rotation of the rotor given by,

$$F_i = \left(\frac{K_v \sqrt{K_\tau \sqrt{2\rho A}}}{b_r} \right) \omega_i = k \omega_i^2 \quad (3.1)$$

where K_v and K_τ are constants related to the motor properties, ρ is the density of the air, A is the area the rotor sweeps out, K_τ is a constant determined by the blade configuration and parameters, and k is the thrust factor[43].

The quadcopter's total thrust (in the body frame) is determined by adding up all of the forces which is given by

$$T_B = \sum_{i=1}^4 F_i = k \begin{bmatrix} 0 \\ 0 \\ \sum \omega_i^2 \end{bmatrix} \quad (3.2)$$

The torque contributed by the rotor about the body frame z -axis, which is required to keep the propeller spinning and providing thrust, generated due to drag is given by[44]

$$\tau_i = \frac{1}{2} \rho \Omega C_D A (\omega R)^2 \simeq b \omega_i^2 \quad (3.3)$$

where R represents the propeller radius, C_D is a drag coefficient, and b is the drag constant.

The total torque around the z -axis is calculated by summing the individual torques produced by each propeller

$$\tau_\psi = b(\omega_1 - \omega_3 + \omega_2 - \omega_4) \quad (3.4)$$

The roll and pitch torques is obtained from standard mechanics. The motors on the roll axis are arbitrarily chosen to be $i = 1$ and $i = 3$, thus

$$\tau_\phi = \sum r \times T = L_i k (\omega_1^2 - k \omega_3^2) = Lk(\omega_1^2 - \omega_3^2) \quad (3.5)$$

The pitch torque is given by the expression below:

$$\tau_\theta = Lk(\omega_2^2 - \omega_4^2) \quad (3.6)$$

where L is the center distance of the propellers. Summing all, the torques in the body frame are found to be:

$$\tau_B = \begin{bmatrix} Lk(\omega_1^2 - \omega_3^2) \\ Lk(\omega_2^2 - \omega_4^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \quad (3.7)$$

3.1.2 Kinematics

The quadcopter in this model is considered a rigid body with body fixed frame attached to it. The inertial frame is stationary with respect to the ground, with gravity acting along the negative z -axis. In contrast, the body frame is attached to the quadcopter and moves with it in the inertial frame, the rotor axes of the positive z -direction, while the quadcopter's arms extend along the x and y axes. Let (x_B, y_B, z_B) be the body fixed frame of reference, where x_B and y_B are parallel to the arms of the quadcopter and orthogonal to each other, whereas z_B is oriented orthogonally to the plane $Oxyz$. The inertial frame coordinate system (x_I, y_I, z_I) is fixed with respect to the ground.

The position of the quadcopter in the inertial frame is $x = (x, y, z)^T$ and velocity is $\dot{x} = (\dot{x}, \dot{y}, \dot{z})^T$. In the body frame, roll, pitch, and yaw is defined as $\theta = (\phi, \theta, \psi)^T$ and angular velocity as $\dot{\theta} = (\dot{\phi}, \dot{\theta}, \dot{\psi})^T$.

The data and calculation make more sense when they are in the inertial frame. But the flight computer unit (FCU) calculates all data in the body fixed frame. So, to transform these data, a rotation matrix is required. The rotation matrix is applied by applying three successive rotations around the axes of the body-fixed frame. Quadcopter orientation relative to inertial frame is bounded using XYZ Euler angles. The angles are defined as the following: the roll angle ϕ is bounded by $-\frac{\pi}{2} < \phi < \frac{\pi}{2}$, the pitch angle θ is within $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$, and the yaw angle ψ ranges from $-\pi < \psi < \pi$ [45].

To convert from body fixed to inertial frame, the rotation matrix is given in Equation 3.1.8,

$$R_I = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \quad (3.8)$$

Here $C = \cos(\cdot)$ and $S = \sin(\cdot)$.

3.1.3 Equations of Motion

There are two types of equation to define the quadcopter dynamics. The first set of equations is due to the Principle of Action, called the Euler-Lagrange or Lagrangian formulation. The second equation which is needed to derive the equations. The Newton-Euler method is used to derive the equations. The Newton-Euler method describes the quadcopter's motion by integrating Newton's Second Law of Motion with Euler's Equations

for rotational dynamics. The Newton-Euler approach allows us to model both rotational and translational movements simultaneously.

The forces acting on the quadcopter in the inertial frame result from thrust, gravity, and friction. The thrust forces in the body frame are calculated by applying the rotation matrix to the thrust in the body frame. Thus the translational motion is described as:

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + F_D + RT_B \quad (3.9)$$

Here x is quadcopter's position, g is the acceleration due to gravity, F_D is the drag force, and T_B is the body frame thrust.

Since, the quadcopter rotates as well, an another equation is needed to describe its angular motion, just like how Newton's second law describes linear motion. This is given by Euler's rotational equation,

$$I\dot{\omega} + \omega \times (I\omega) = \tau \quad (3.10)$$

where, ω is the angular velocity (how fast it is rotating). I is the inertia matrix (describes how mass is distributed). τ is the external torque. The term $\omega \times (I\omega)$ accounts for the rotational effects due to angular velocity. This can be rewritten as

$$\dot{\omega} = I^{-1}(\tau - \omega \times (I\omega)). \quad (3.11)$$

The inertia matrix I is symmetrical, based on above mentioned assumptions and is given by

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (3.12)$$

The final expression for the rotational equations of motion is

$$\dot{\omega} = \begin{bmatrix} \frac{\tau_\phi}{I_{xx}} \\ \frac{\tau_\theta}{I_{yy}} \\ \frac{\tau_\psi}{I_{zz}} \end{bmatrix} + \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix} \quad (3.13)$$

3.2 Visual-Inertial Odometry (VIO)

Visual-Inertial Odometry (VIO) is a state estimation technique that fuses measurements from a camera (visual constraints) and an Inertial Measurement Unit (IMU) to estimate

the six-degree-of-freedom (6-DoF) pose and velocity of the quadcopter. By combining the long-term consistency of visual data with the short-term accuracy and rapid response of inertial sensors, VIO achieves robust localization even in challenging environments where GPS is denied or unreliable [23].

The core objective of VIO is to estimate the state vector $\mathbf{x}_t$ at time t . A typical state vector includes the position, velocity, orientation (quaternion), and IMU biases:

$$\mathbf{x}_t = \begin{bmatrix} {}^W \mathbf{p}_B \\ {}^W \mathbf{v}_B \\ {}^W \mathbf{q}_B \\ \mathbf{b}_g \\ \mathbf{b}_a \end{bmatrix} \quad (3.14)$$

where ${}^W \mathbf{p}_B \in \mathbb{R}^3$ is position, ${}^W \mathbf{v}_B \in \mathbb{R}^3$ is velocity, ${}^W \mathbf{q}_B \in \mathbb{S}^3$ is orientation quaternion, $\mathbf{b}_g, \mathbf{b}_a \in \mathbb{R}^3$ are gyroscope and accelerometer biases.

VIO algorithms are broadly categorized into two families based on their backend estimation machinery: filter-based methods (real-time state estimation) and optimization-based methods (batch state refinement)

3.2.1 Filter-Based VIO

Filter-based methods use recursive Bayesian estimation, primarily through Extended Kalman Filter (EKF) or its variants. In this framework, the system state is propagated forward in time using the high-frequency IMU measurements (Prediction Step) and corrected when visual features are observed (Update Step). A critical challenge in EKF-based VIO is the management of feature points including 3D feature positions in the state vector that leads to cubic computational complexity growth. To address this, modern filter-based approaches utilize tightly-coupled marginalization. Instead of maintaining features in the state, the algorithm forms a geometric constraint involving all camera poses from which a feature was tracked, and then projects the feature out of the state vector.

3.2.2 Optimization-Based VIO

Optimization-based methods formulate VIO as nonlinear least squares over a temporal window, achieving higher accuracy through batch processing. Rather than processing measurements sequentially and discarding them, this approach minimizes a joint cost function $J(\mathbf{x})$ that aggregates the residuals from both inertial and visual measurements over the entire window. To estimate the state of the quadrotor, the VINS-Fusion algo-

rithm employs a tightly-coupled sliding window optimization framework. The optimal state vector $\mathcal{X}$ is obtained by minimizing the sum of the prior, inertial, and visual residuals. The objective function is formulated as a non-linear least squares problem:

$$\min_{\mathcal{X}} \left\{ |\mathbf{r}_{\text{prior}}|^2 + \sum_{k \in \mathcal{B}} \left| \mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{k+1}^k, \mathcal{X}) \right|_{\mathbf{P}_{\mathcal{B}}}^2 + \sum_{(i,j) \in \mathcal{C}} \left| \mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_{i,j}, \mathcal{X}) \right|_{\mathbf{P}_{\mathcal{C}}}^2 \right\} \quad (3.15)$$

where:

- $\mathbf{r}_{\text{prior}}$ represents the marginalization residual, incorporating prior information from past states that have slid out of the optimization window.
- $\mathbf{r}_{\mathcal{B}}$ is the IMU residual, which captures the error between the pre-integrated inertial measurements and the relative motion of the estimated states.
- $\mathbf{r}_{\mathcal{C}}$ is the visual reprojection residual, defining the geometric error between the observed feature coordinates on the image plane and the projected 3D landmarks.

This technique integrates raw inertial measurements between two visual keyframes into a single relative motion constraint that is independent of the global state, avoiding the need to re-integrate thousands of IMU samples during every optimization step. As the state estimate improves during optimization, the Jacobians are re-computed allowing the system to correct early estimation errors.

3.3 Path Planning

Path planning involves computing collision-free trajectories from a start configuration q_{start} to a goal q_{goal} within a configuration space $C = C_{\text{free}} \cup C_{\text{obs}}$. The problem is decomposed into global planning (strategic pathfinding on a map) and local planning (reactive trajectory generation).

3.3.1 Global Path Planning

Global planners operate on a known static map to generate a rough sequence of way-points.

i. Dijkstra's Algorithm

Dijkstra's algorithm is a uniform-cost search method that guarantees the shortest path in a graph. It explores nodes by minimizing the accumulated cost $g(n)$ from the start node:

$$f(n) = g(n) \quad (3.16)$$

The cost update rule for a neighbor v of node u is given by:

$$g(v) = \min(g(v), g(u) + c(u, v)) \quad (3.17)$$

where $c(u, v)$ is the edge weight. While optimal, Dijkstra's expands in all directions equally, making it computationally expensive for large 3D spaces.

ii. A* Algorithm

The A* algorithm extends Dijkstra's by incorporating a heuristic function $h(n)$ to guide the search toward the goal, significantly reducing the search space. The total cost function is:

$$f(n) = g(n) + h(n) \quad (3.18)$$

For 3D quadcopter navigation, the Euclidean distance is commonly used as the admissible heuristic:

$$h(n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2 + (z_n - z_g)^2} \quad (3.19)$$

A* maintains optimality provided that $h(n)$ never overestimates the true cost-to-go.

3.3.2 Local Path Planning

Local planners generate immediate control commands or short-term trajectories to avoid dynamic obstacles and satisfy kinematic constraints.

i. Artificial Potential Fields (APF)

APF treats the robot as a particle moving in a gradient field. The goal exerts an attractive potential U_{att} , while obstacles exert a repulsive potential U_{rep} . The total potential function $U(q)$ is defined as:

$$U(q) = \frac{1}{2}\xi|q - q_{\text{goal}}|^2 + \sum_i \frac{1}{2}\eta \left(\frac{1}{\rho_i(q)} - \frac{1}{\rho_0} \right)^2 \quad (3.20)$$

where $\rho_i(q)$ is the distance to the i -th obstacle and ρ_0 is the influence radius. The control input is derived from the negative gradient of the field:

$$F(q) = -\nabla U(q) \quad (3.21)$$

While computationally inexpensive, APF suffers from local minima where the attractive and repulsive forces cancel out before reaching the goal.

- ii. **B-Spline Trajectory Optimization** To ensure smoothness and dynamic feasibility, modern planners represent the local trajectory as a uniform B-Spline. A B-Spline is parameterized by control points $Q_1, Q_2, \dots, Q_N$. Due to the convex hull property, if the control points are collision-free, the entire continuous trajectory is collision-free.

The planner optimizes the position of these control points to minimize a multi-objective cost function J_{total} :

$$J_{\text{total}} = \lambda_s J_{\text{smooth}} + \lambda_c J_{\text{collision}} + \lambda_d J_{\text{feasibility}} \quad (3.22)$$

where:

- J_{smooth} minimizes higher-order derivatives (velocity, acceleration, jerk) to ensure smooth flight.
- $J_{\text{collision}}$ utilizes gradient information from the sensor map to push control points away from obstacles.
- $J_{\text{feasibility}}$ penalizes the trajectory if the dynamic limits (v_{max} , a_{max}) are exceeded.

This optimization-based approach generates high-quality, continuous trajectories suitable for agile quadrotor flight in cluttered environments.

Chapter 4

METHODOLOGY

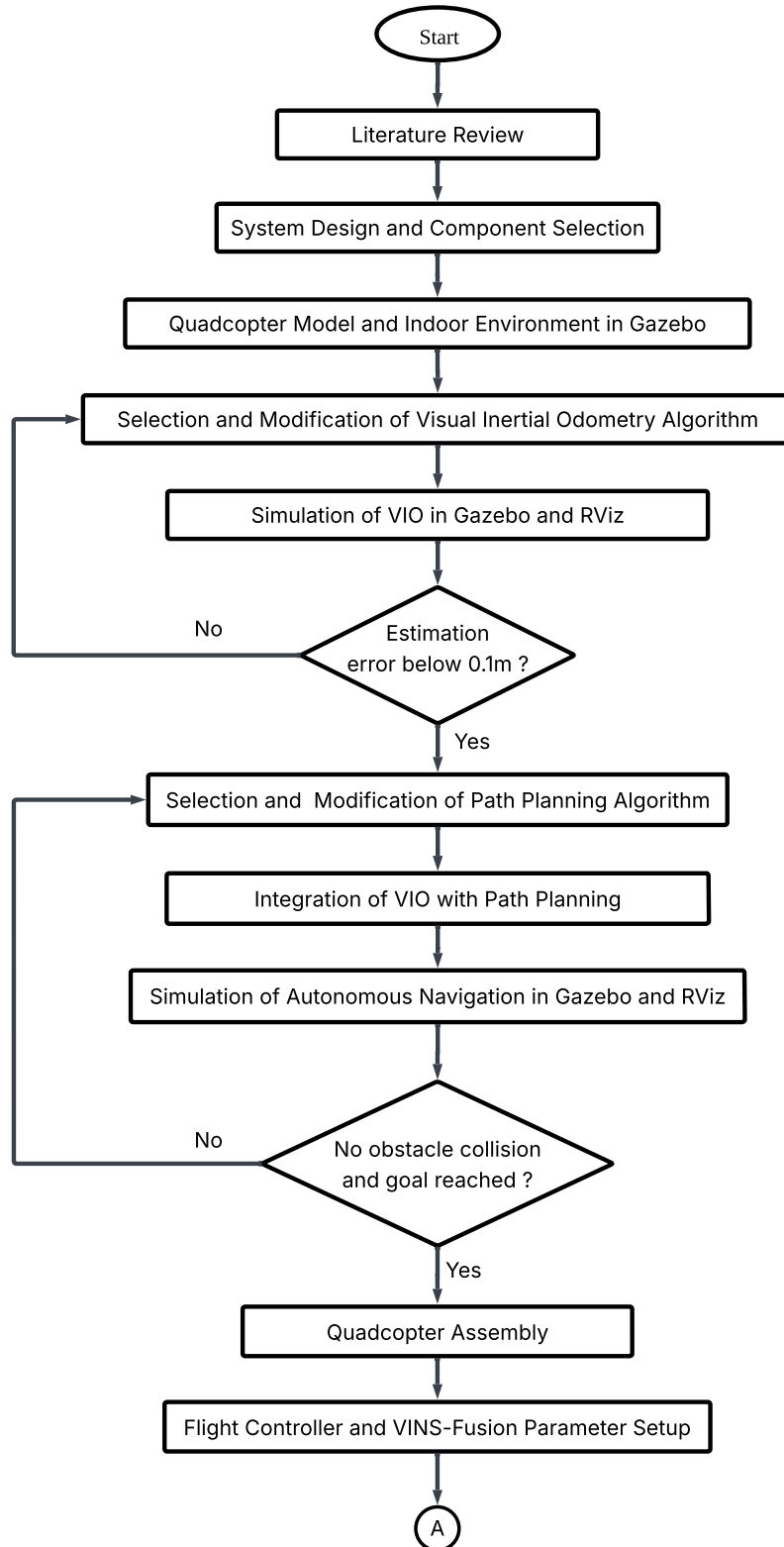


Figure 4.1: Detailed Methodology of Project- Part I

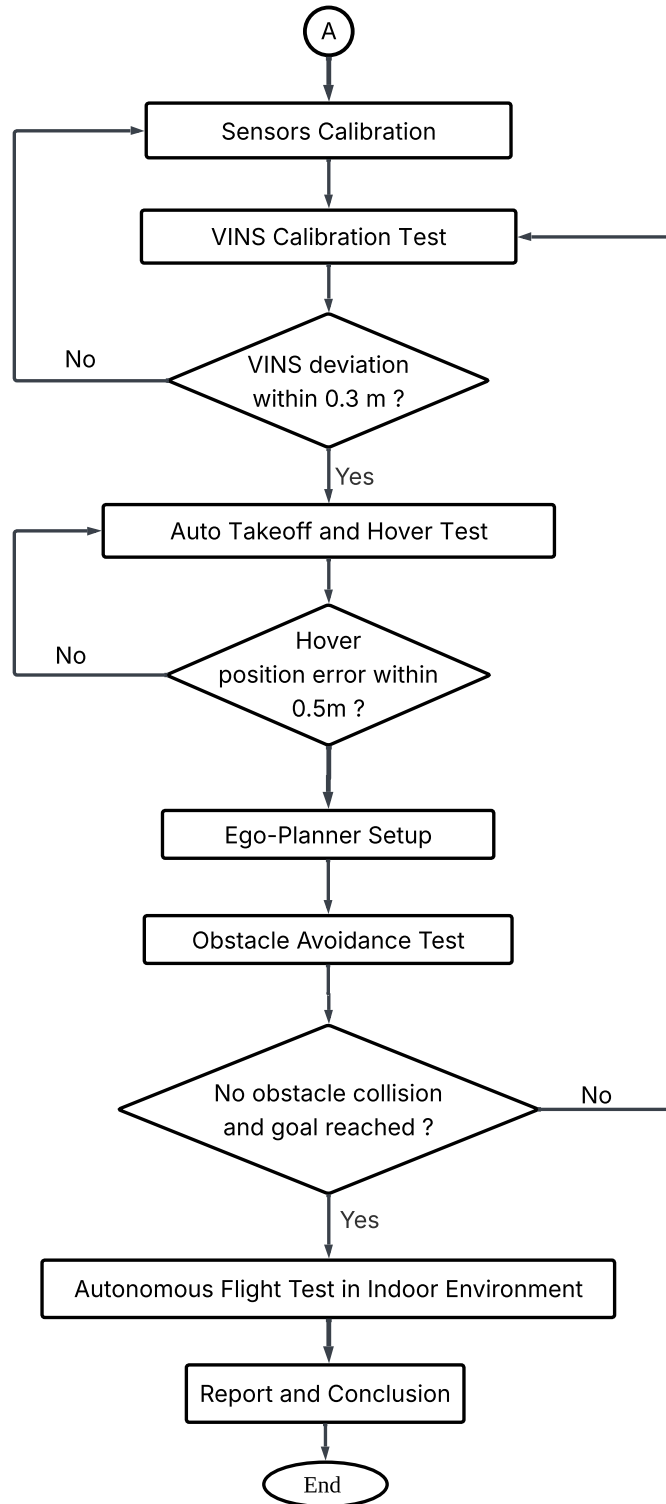


Figure 4.2: Detailed Methodology of Project- Part II

The methodology is the major aspect of the project in which the actual approach to completing the project is explained. The flowchart covers our approach to solving the problem statement of the project. The project methodology is divided into 2 main parts: Simulation and Hardware Integration, and each individual block is described as follows:

4.1 Simulation

This section explains the simulation framework established to validate autonomous navigation, developed to mitigate the safety risks associated with physical flight. A high-fidelity Software-in-the-Loop (SITL) environment was created using the PX4 flight stack which was selected for its versatile sensor and companion computer APIs. The Gazebo simulator was selected for its realistic physics-based environment that provides the necessary real-time sensor data. This integrated simulation enables rigorous testing of the Visual-Inertial Odometry pipeline and path planning algorithms by accurately simulating the quadcopter's dynamics and sensor feedback within a virtual world.

4.1.1 Quadcopter model and Indoor Environment in Gazebo

For simulation, the robot's physical description was defined using the Simulation Description Format (SDF). While URDF is common in general ROS applications, SDF is the preferred format for the Gazebo physics engine and the PX4 flight stack due to its ability to describe the robot and the environment within a single standardized XML framework. The SDF explicitly defines the robot as a hierarchy of links (rigid bodies) and joints (constraints), detailing visual elements for rendering, collision geometries for physics interactions, and inertial properties for dynamic simulation.

To create the simulation quadcopter model and environment, the following implementation steps were taken:

i. Modification of the standard Iris Quadcopter

Rather than modeling a vehicle from scratch, the standard Iris Quadcopter model provided by the PX4 firmware was utilized as the baseline. This model provides a validated aerodynamic structure. The SDF of the model was modified to replicate the specific mass distribution of our custom hardware setup.

ii. Integration of RealSense D435 Sensor

A critical modification was the integration of the Intel RealSense D435 depth camera into the Iris SDF. Unlike standard cameras, the D435 provides depth data, RGB data and Infrared data which are essential for the algorithm.

The camera was modeled as a child link attached to the quadcopter's base link via a fixed joint. The transform `<pose>` tag was calibrated to match the physical mounting position of the camera on the real quadcopter. To simulate active sensor streams, the `realsense_gazebo_plugin` was embedded into the SDF.



(a) Quadcopter with realsense camera

```
prakash@Warrior:~$ rostopic list
/iris/d435/color/camera_info
/iris/d435/color/image_raw
/iris/d435/depth/camera_info
/iris/d435/depth/image_raw
/iris/d435/infra1/camera_info
/iris/d435/infra1/image_raw
/iris/d435/infra2/camera_info
/iris/d435/infra2/image_raw
```

(b) Camera topics published in ROS

Figure 4.3: Quadcopter with camera and its published topics

iii. Visual and Collision Models

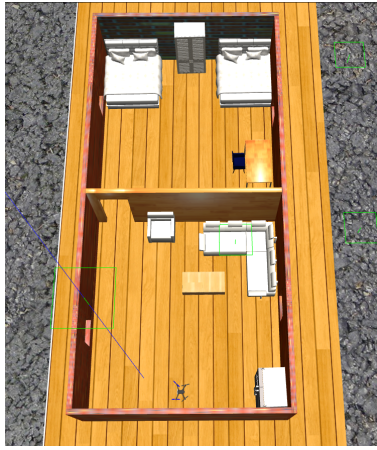
The visual properties utilize high-resolution mesh files (.dae) for realistic rendering in Gazebo, while collision properties use simplified geometric primitives (cylinders and boxes) to reduce the computational load on the physics engine during collision checks.

iv. Dynamic Inertial Properties

For the algorithm to function correctly and for the flight controller to maintain stability, the dynamic properties of the model were rigorously defined. The `<inertial>` block within the SDF was updated to reflect the total mass of the quadcopter plus the added payload. The mass (m) and the inertia matrix (I_{xx} , I_{yy} , I_{zz}) were adjusted. This ensures that the simulated response to thrust and torque commands mirrors the real-world dynamics, allowing the PID gains tuned in simulation to remain valid during real-world flight.

v. Virtual Indoor Environment

To create a robust testing ground for the quadcopter, a virtual simulation environment was constructed within the Gazebo physics engine. This environment is defined using the Simulation Description Format (SDF) in a (.world) file, which specifies essential physical parameters including terrain properties, gravity vectors, and lighting conditions. Custom models were integrated via their URIs to populate the scene. To facilitate a graduated validation process, two distinct virtual worlds were developed. The first is a simplified Validation Environment, consisting of a two-room layout with minimal obstacles, designed specifically for the initial tuning of VIO covariance parameters and PID controller gains. The second is a Complex Testing Environment, a feature-rich indoor world mimicking a workspace with multiple rooms and random static obstacles. This environment serves as the primary stress test for the integrated system, challenging the planner's ability to generate collision-free trajectories in confined spaces.



(a) Environment 1



(b) Environment 2

Figure 4.4: Virtual Environments in Gazebo

4.1.2 Selection and Modification of Visual Inertial Odometry Algorithm

State estimation is the cornerstone of autonomous flight in GPS-denied indoor environments. To select the most suitable algorithm for the Raspberry Pi companion computer, a comparative analysis was conducted focusing on the trade-off between localization accuracy, mapping capabilities, and computational efficiency.

Table 4.1: Comparison of Visual-Inertial Odometry Algorithms

Method	Strengths	Accuracy	Mapping Capabilities	Computational Power
VINS-Fusion	Tightly-coupled VIO; Allows separation of odometry and loop closure	High	Sparse visual map; Supports loop closure for drift correction	Moderate (Tunable)
ORB-SLAM3	Full support for stereo-inertial; Multi-map support	Very High	Dense keyframe-based map; High memory usage	High
MSCKF (OpenVINS)	Efficient marginalization; Does not require maintaining a map state	High	None (Pure Odometry); No loop closure capabilities	Moderate
SVO 2.0	Semi-direct method; Fast tracking of pixels; Good for high-speed flight	Moderate	Sparse map; Vulnerable to motion blur	Low

VINS-Fusion was selected as the primary estimator for this project. While filter-based methods like MSCKF offer lower computational costs, they often lack the robust loop-closure capabilities required for long-duration flights in complex environments. Conversely, ORB-SLAM3 offers superior accuracy but places a computational load that frequently exceeds the overhead limits of the Raspberry Pi 4B, potentially inducing latency in the control loop.

VINS-Fusion provides the optimal balance for our hardware configuration. The primary reason for its selection is its modular multi-threaded architecture. This design allows the loop-closure thread to be decoupled, ensuring that the core visual-inertial odometry thread remains real-time capable. Even if the global optimization lags due to CPU constraints, the high-frequency odometry required for flight stability is preserved. Furthermore, its ability to fuse stereo cameras with IMU data makes it robust against scale ambiguity, a critical factor for navigating the simulated indoor corridor environments.

4.1.3 VINS-Fusion

VINS-Fusion is an optimization-based multi-sensor state estimator which extends the popular VINS-Mono framework. It is designed to achieve highly accurate VIO by fusing high-frequency IMU data with visual measurements from one or more cameras (Monocular or Stereo). Unlike loose-coupling methods, VINS-Fusion employs a tightly-coupled sliding window optimization approach.

The system architecture consists of three main threads: Measurement Preprocessing, Local Bundle Adjustment, and Loop Closure.

i. Measurement Preprocessing

Visual features are tracked across consecutive frames using the KLT (Kanade-Lucas-Tomasi) sparse optical flow algorithm, with a minimum pixel separation enforced to maintain a spatially uniform feature distribution. Concurrently, IMU measurements are pre-integrated between visual frames, compactly encoding the relative motion between frames without requiring re-integration at each optimization step, significantly reducing the computational burden of the solver.

ii. Sliding Window Optimization

The core estimation is performed by a non-linear solver operating over a fixed-size sliding window of recent keyframes (typically 10–15 frames), optimizing the full state vector of position, velocity, orientation, and IMU biases. As new frames enter the window and old frames are evicted, marginalization is applied: the information from the discarded frame is condensed into a prior factor and

folded into the remaining window, preventing loss of accumulated constraints while bounding computational complexity.

iii. **Parameter Tuning of VINS-Fusion**

Extensive parameter tuning is required to ensure the stability of the state estimator on the limited computational resources of the Raspberry Pi 4B and to adapt to the noise characteristics of IMU and RealSense D435 camera.

In simulation, IMU data is often cleaner but can suffer from discrete time-stepping artifacts, while the visual features in a synthetic environment can be repetitive. The parameters were adjusted to reduce the CPU load of the non-linear solver and to tighten the outlier rejection thresholds.

4.1.4 Simulation of VIO in Gazebo and RViZ

The quadcopter model with integrated camera and an indoor world was launched in gazebo and VINS-Fusion algorithm was launched that subscribes to IMU and camera topics. The validation of the VINS-Fusion algorithm within the Gazebo environment was executed in two distinct phases: Open-Loop Estimation Analysis and Closed-Loop Autonomous Control. This phased approach ensured that the state estimator was accurate before being trusted to actively close the control loop of the quadrotor.

Phase 1: Open Feedback Loop Odometry Analysis

The primary objective of this phase was to verify the accuracy of the VIO pipeline where the quadcopter's stability did not depend on the vision algorithm. The quadcopter was commanded to execute a predefined waypoint mission using QGroundControl (QGC). During this flight, the PX4 flight controller utilized the simulated GPS module for position hold and navigation. Concurrently, the VINS-Fusion node was active, receiving sensor data from the simulated IMU and RealSense camera but acting solely as a passive observer.

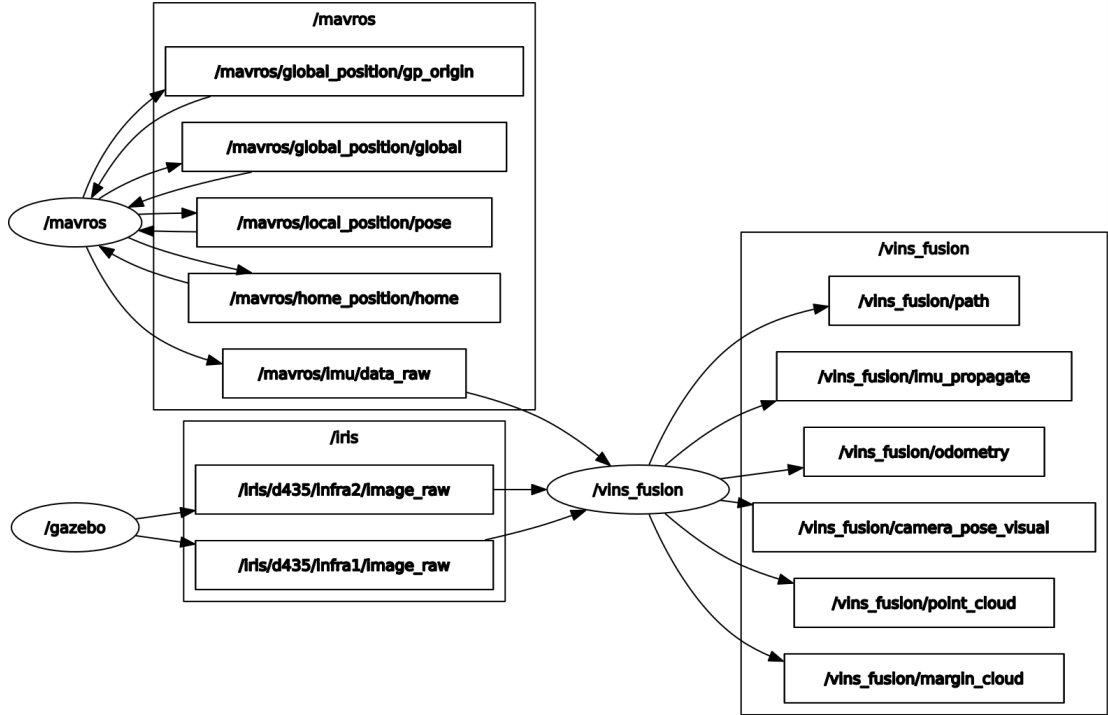


Figure 4.5: Open-Loop System Architecture

The estimator's performance was monitored in real-time using RViz. We specifically analyzed three key visualizations:

`/vins_estimator/odometry`: The low-frequency, high-precision pose estimate from the sliding window optimization.

`/vins_estimator/imu_propagate`: The high-frequency state estimate derived from IMU pre-integration, which fills the gaps between visual frames.

`/vins_estimator/path`: The global trajectory history.

Phase 2: Closed Feedback Loop Autonomous Control

In this phase, the simulated GPS was disabled, forcing the quadcopter to rely entirely on the onboard VIO for localization. To enable vision-based control, a custom vins to mavros bridge command was implemented. A ROS node acts as an interface between the VIO output and the flight controller. It subscribes to `/vins_estimator/odometry` topic, transforms the coordinate frame from the VIO local frame to the ENU frame required by MAVROS, and publishes the result to `/mavros/vision_pose/pose` topic. This topic feeds the visual position estimate directly into the PX4 EKF2, effectively replacing GPS as the local position source.

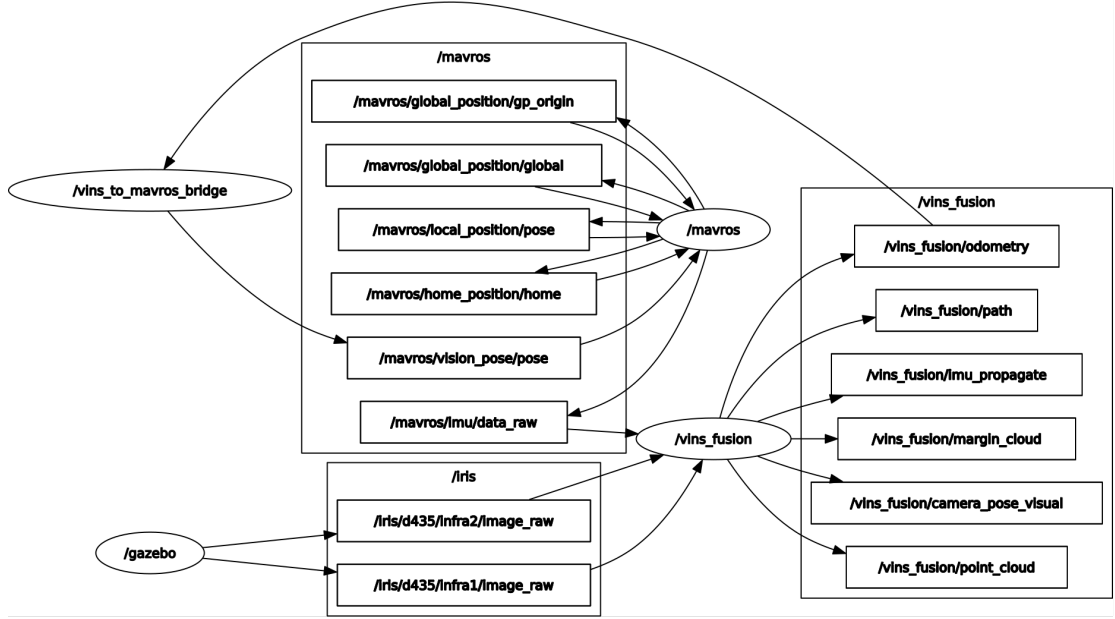


Figure 4.6: Closed-Loop System Architecture

The quadcopter was switched to OFFBOARD mode and commanded to execute a square trajectory maneuver. The performance was evaluated by observing the quadcopter’s ability to maintain a stable hover and precisely trace the corners of the square without drifting.

4.1.5 Selection and Modification of Path Planning Algorithm

To enable agile autonomous navigation within the computational constraints of the Raspberry Pi, the EGO-Planner framework was selected. Traditional gradient-based planners rely on constructing a Euclidean Signed Distance Field (ESDF) for the entire local map to evaluate obstacle gradients. Computing an ESDF is computationally expensive and redundant, as the trajectory optimization often occurs in a small subspace of the map.

EGO-Planner[17] was chosen because it eliminates the ESDF construction step, reducing computation time by orders of magnitude while maintaining trajectory smoothness and safety. The planning architecture consists of two hierarchical layers: a geometric front-end and an optimization based back-end.

i. Global Front-End: Kinodynamic Path Searching

The front-end utilizes a Kinodynamic A* algorithm to generate an initial, collision-free trajectory. Geometric A* treats the quadcopter as a point mass moving between grid cell but Kinodynamic A* samples the state space of the quadcopter. It considers the vehicle’s current state (position, velocity, acceleration) and samples

discrete control inputs (jerk) to propagate the state forward. This ensures that the global path is not only collision free but also respects the physical dynamics of the quadcopter, serving as a feasible initialization for the local optimizer.

ii. **Local Back-End: ESDF-Free B-Spline Optimization**

The core of the navigation stack is the local planner, which refines the global path into a smooth Uniform B-Spline trajectory. B-splines has convex hull property, which guarantees that if all control points of a spline segment are collision free, the entire trajectory segment is also collision free.

The planner deforms the B-spline control points $Q_1, Q_2, \dots, Q_N$ to minimize a total cost function J_{total} , formulated as:

$$J_{\text{total}} = \lambda_s J_{\text{smooth}} + \lambda_c J_{\text{collision}} + \lambda_d J_{\text{feasibility}} \quad (4.1)$$

where $\lambda_s, \lambda_c, \lambda_d$ are weighting coefficients tuned for the specific environment.

(a) **Smoothness Cost (J_{smooth}):**

To ensure agile flight without abrupt control inputs, smoothness minimizes the jerk and higher order derivatives of the trajectory. For a B-spline, this is efficiently computed using the finite difference of adjacent control points.

(b) **Collision Cost ($J_{\text{collision}}$):**

Instead of querying a pre-computed distance map, the algorithm checks for collisions only along the current trajectory.

- **Collision Check:** The planner iterates through the control points. If a point (or the segment between points) falls inside an obstacle, it computes a discrete anchor point (p) on the obstacle's surface and a direction vector ($\mathbf{v}$) pointing away from the obstacle.
- **Penalty:** The cost function models the obstacle as a repulsive field relative to this anchor point. The gradient is calculated explicitly using the distance between the control point and the obstacle surface pair $\{p, \mathbf{v}\}$.

This method restricts gradient computation strictly to the areas necessary for collision resolution, drastically reducing CPU usage on the Raspberry Pi.

(c) **Feasibility Cost ($J_{\text{feasibility}}$):**

This term applies soft constraints to ensure the generated trajectory does not exceed the quadcopter’s physical limits.

- **Velocity Limit:** Penalizes segments where $|\mathbf{v}_i| > v_{\text{max}}$.
- **Acceleration Limit:** Penalizes segments where $|\mathbf{a}_i| > a_{\text{max}}$.

iii. **Egoplanner Parameter Tuning**

To adapt the generic EGO-Planner to the simulated Iris Quadcopter and the RealSense D435 sensor, specific parameters were modified in the `planning_params.yaml` file.

4.1.6 Integration of VIO and Path Planning Algorithm

The successful deployment of the autonomous stack requires a precise interface between the state estimator (VINS-Fusion) [9], the trajectory generator (EGO-Planner) [17], and the flight controller (PX4). Rather than relying on the standard PX4 cascaded P-PID position loops, we implemented a robust Geometric Control architecture to ensure high-speed tracking accuracy.

i. **State Estimation**

The `vins_node` publishes two state outputs: a globally consistent pose at 20 Hz via `/vins_estimator/odometry`, derived from sliding-window optimization, and a high-rate IMU-propagated state at 200 Hz via `/vins_estimator/imu_propagate`, carrying the full state vector $(\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega})$. The latter is the primary data source for both the planner and the controller, as its frequency is sufficient for real-time operation.

ii. **Planner Integration**

The EGO-Planner subscribes to `/vins_estimator/imu_propagate` to initialize each planning cycle at the quadcopter’s instantaneous state $(\mathbf{p}_{\text{start}}, \mathbf{v}_{\text{start}}, \mathbf{a}_{\text{start}})$. This high-rate pose is synchronized with the depth stream from the topic named `/iris/d435/depth/image_raw`, ensuring that obstacles are registered into the local voxel map with minimal motion blur and that the occupancy grid remains consistent with the quadcopter’s rapid movements.

iii. **Geometric Controller Implementation**

The `px4_ctrl` node implements a Non-linear Geometric Controller that oper-

ii. **VIO Initialization and Stabilization:**

Once the simulation is active, the VINS-Fusion node is launched and the system is kept stationary for approximately 3-5 seconds for the estimator to initialize its gravity vector alignment and for the IMU bias estimation to converge.

iii. **PX4 Controller:**

With a stable state estimate, the `px4_ctrl` node is initiated which actively subscribes to the high-frequency VIO data and computes control setpoints but remains disarmed (sending neutral commands) until explicitly triggered. This ensures the controller is synchronized with the state estimator before flight.

iv. **Trajectory Planner Initialization:**

The EGO-Planner framework is then launched to handle path generation. It initializes the local gradient-based planner. It begins subscribing to the depth camera stream and the VIO pose to build the local ring-buffer voxel map, preparing the system for collision avoidance.

v. **Visualization Interface (RViz):**

A pre-configured RViz session is launched to monitor the system internals which allows to visualize the Voxel Map, the Global Path, and the B-Spline Trajectory. It also serves as the user interface for sending 2D Nav Goals to the planner.

vi. **Autonomous Takeoff and Engagement:**

Finally, the mission is triggered by sending a command to the control node. The command signals the `px4_ctrl` node to switch the PX4 flight mode to OFFBOARD, arm the motors, and execute a smooth takeoff maneuver to a predefined hover altitude. Once hovering, the system is fully autonomous and awaiting trajectory goals from the planner.

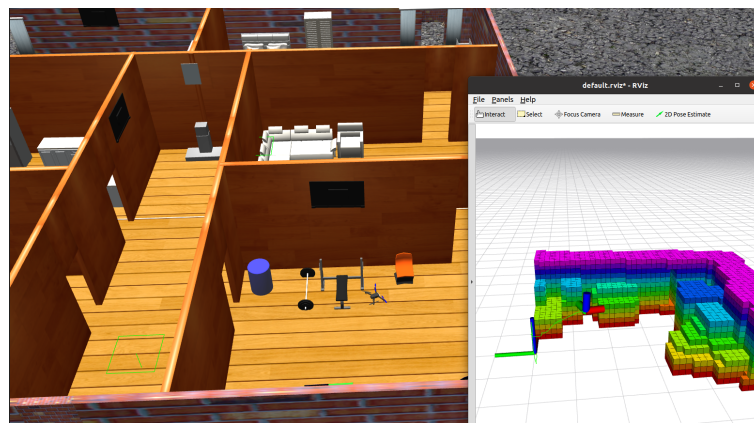


Figure 4.8: Voxel Map of surrounding

After successful autonomous takeoff and stabilization at the target hover altitude, the system transitions into the active navigation phase. The quadcopter is capable of executing missions through two distinct command interfaces: Predefined Waypoint Navigation and Interactive 2D Pose Navigation.

For Waypoint Navigation, specific target coordinates can be hardcoded directly into the `ego_planner` launch file. Upon initialization, the planner sequentially executes these waypoints, allowing for consistent performance benchmarking of the trajectory generation in identical scenarios. Alternatively, for dynamic exploration, we can utilize the RViz interface to provide real-time targets. By selecting the 2D Nav Goal tool and clicking on a reachable location within the voxel map, a target state ($\mathbf{p}_{\text{goal}}, \mathbf{v}_{\text{goal}} = 0$) is published to the planner.

Once a goal is received, the planning stack visualizes the generated solution using distinct color-coded trajectories:

- **Global Reference Path:** It is the blue curve generated by the Kinodynamic A* front-end. This path represents the initial, collision-free geometric guess that connects the start position to the goal while respecting the quadcopter's kinematic limits. It is continuously updated or locally repaired as the quadcopter progresses and discovers new obstacles, serving as a collision-free topological guide for the optimizer.
- **Optimized Local Trajectory:** It is red curve generated by the B-Spline optimization back-end, which refines the blue reference path by minimizing gradients for smoothness and dynamic feasibility. The quadcopter tracks this red trajectory, adjusting it in real-time to avoid newly detected obstacles while maintaining high-speed flight.

The simulation of Autonomous Navigation showed the quadcopter successfully integrated VINS-Fusion and EGO-Planner for localization, real-time obstacle avoidance and replanning. The quadcopter performed collision free navigation in indoor environment reaching the goal without relying on external positioning systems like GPS.

4.2 Hardware and System Integration

4.2.1 Quadcopter Assembly and Sensor Integration

The quadcopter was constructed using a lightweight carbon fiber frame, driven by brushless DC (BLDC) motors and electronic speed controllers (ESCs). The avionics payload consisted of a Pixhawk 4 flight controller and a Raspberry Pi 4B companion computer. A buck converter was utilized to step down and regulate the lithium-polymer (LiPo) battery voltage, ensuring stable power delivery to the companion computer. To mitigate the impact of high-frequency mechanical vibrations on the inertial sensors, stress-absorbing dampeners were integrated into the motor mounts. An external 256 GB SSD was interfaced with the Raspberry Pi to facilitate high-speed, real-time logging of flight data.

Prior to flight, ESC calibration and motor direction verification were executed to ensure symmetric thrust generation. All external exteroceptive sensors were rigidly mounted and geometrically aligned with the vehicle's center of mass (body frame) to prevent transformation errors during state estimation.

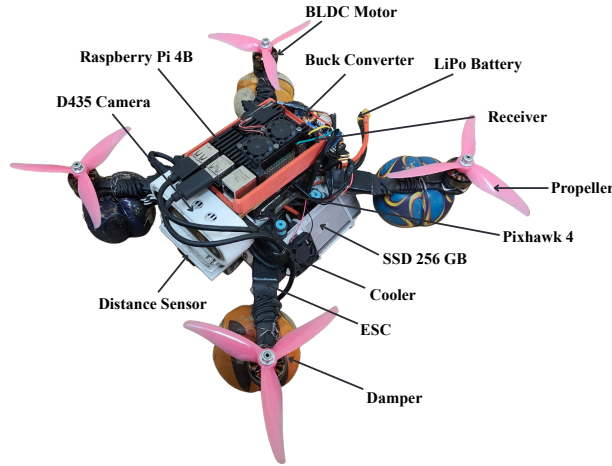


Figure 4.9: Assembled quadcopter platform highlighting component placement

i. Integration of Distance Sensor:

To enhance vertical state estimation, a VL53L1X Time-of-Flight (ToF) distance sensor was mounted in a downward-facing configuration. The sensor was interfaced with the Pixhawk 4 via an I2C telemetry port. Within the PX4 firmware, the hardware drivers were enabled and configured (`SENS_EN_VL53L1X` and `EKF2_RNG_CTRL`). These precise altitude measurements were fused with internal IMU and barometric data via the Extended Kalman Filter (EKF2) to suppress measurement noise and provide a reliable ground-truth reference against the visual-inertial odometry (VIO) altitude estimates.

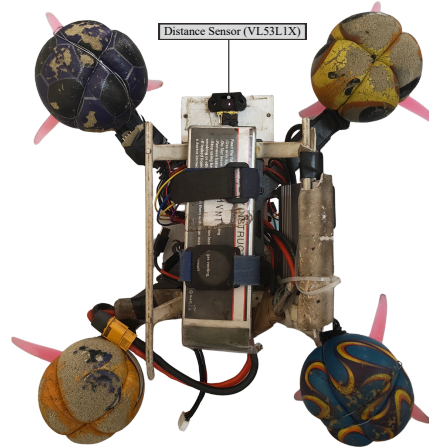


Figure 4.10: Downward-facing attachment of the VL53L1X distance sensor

ii. Integration of the Depth Camera:

An Intel RealSense D435 depth camera was mounted forward-facing and connected to the Raspberry Pi 4B via a high-bandwidth USB 3.0 interface. The Raspberry Pi hosted the VINS-Fusion ROS node, which processed the synchronized stereo infrared frames and IMU data to compute the vehicle's 6-DOF pose. This pose was continuously transmitted to the flight controller via MAVROS as a `VISION_POSE_ESTIMATE` message. In the PX4 firmware, the `EKF2_EV_CTRL` and `EKF2_HGT_MODE` parameters were activated, instructing the EKF to utilize this external vision data as the primary source for horizontal and vertical localization.

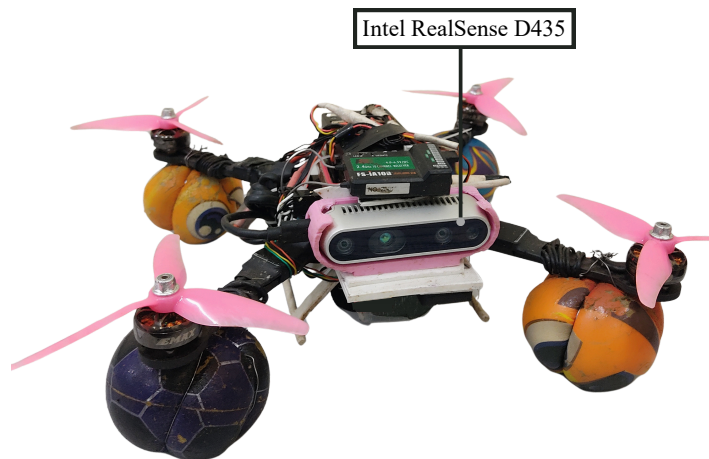


Figure 4.11: Forward-facing integration of the Intel RealSense D435 depth camera

4.2.2 Firmware Configuration and Parameter Tuning

i. Flight Controller Setup:

The Pixhawk 4 was flashed with PX4 firmware version 1.16.0, utilizing a generic quadcopter airframe configuration. To accommodate the dense data flow between the flight controller and the companion computer, the serial telemetry baud rate (SER_TEL1_BAUD) was maximized to 921,600 bps. Furthermore, the IMU publication frequency was forced to 200 Hz via the `extras.txt` configuration file to satisfy the high-rate preintegration requirements of the VINS-Fusion algorithm. Key PX4 parameters are summarized in Table 4.4.

Table 4.2: VINS Parameter Setup

S.N.	Parameters	Value	Remarks
1	max_cnt	100	Maximum number of features tracked
2	min_dist	30 pixels	Minimum pixel distance between features
3	freq	8 Hz	The frequency (Hz) at which the VIO outputs the odometry pose
4	F_threshold	10	RANSAC threshold
5	estimate_td	1	Time offset between Camera and IMU
6	solver_time	0.05 s	Maximum solver time (seconds)
7	num_iterations	6	Maximum iterations for the solver

ii. EGO-Planner Configuration:

The EGO-Planner path-planning framework was heavily parameterized to suit the constrained computing resources of the Raspberry Pi and the physical dynamics of the indoor testing environments. Velocity limits, acceleration constraints, and obstacle inflation radii were defined within the ROS launch and XML configuration files. These parameters, detailed in Table 4.3, were optimized to guarantee real-time trajectory generation without inducing computational latency.

Table 4.3: Parameters Tuning for Ego-Planner

S.N.	Parameters	Value	Remarks
1	Grid Map Resolution	0.1 m	Set to capture the details of the cluttered simulation environment
2	Obstacle Inflation	0.299 m	quadcopter radius plus safety margin to account for the physical footprint of the Iris frame
3	Planning Horizon	6 m	Defines how far ahead the local planner optimizes the trajectory
4	Maximum Velocity	0.5 m/s	Reduced for the safe indoor flight in the simulated environment
5	Maximum Acceleration	1 m/s ²	Aligned with the thrust-to-weight ratio of the loaded quadcopter

4.2.3 Sensor Calibration

Accurate spatial and temporal calibration between the RealSense camera and the Pixhawk IMU is critical for preventing VIO divergence. The calibration was executed in three sequential phases:

- i. **Pixhawk IMU Calibration:** The quadcopter was secured on a perfectly level, stationary surface while the PX4 firmware calculated static accelerometer and gyroscope bias offsets. Raw IMU measurements (linear acceleration and angular velocity) were logged into ROS bag files to verify that the 200 Hz publication rate was stable and free of severe high-frequency noise prior to visual fusion.
- ii. **Camera Intrinsic Calibration:** Because the Intel RealSense D435 is factory-calibrated for depth generation via its stereo infrared sensors, manual depth recalibration was unnecessary. The required intrinsic parameters (focal lengths f_x, f_y and principal points c_x, c_y) were extracted directly from the ROS camera information topics and mapped into the VINS-Fusion configuration files. To conserve USB bandwidth and prevent the camera's structural dot-pattern projector from interfering with the VIO's Kanade-Lucas-Tomasi (KLT) optical flow tracker, both the RGB color stream and the IR emitter were disabled.
- iii. **IMU-Camera Extrinsic Calibration:** Extrinsic calibration determines the rigid 3D transformation matrices (body_T_cam0 and body_T_cam1) that map the camera's optical frame to the IMU's body frame. This was achieved dynamically by initiating the VINS-Fusion estimator and manually moving the quadcopter through a highly textured indoor environment. The estimator continuously optimized the spatial translation and rotation parameters until the odometry drift converged to within an acceptable tolerance (< 0.3 m), after which the finalized extrinsic matrix was hard-coded into the system parameters.

Table 4.4: Pixhawk and Camera Parameter Setup

S.N.	Parameters	Value	Remarks
1	EKF2_HGT_REF	Vision	The reference source of height data
2	EKF2_EV_DELAY	200 ms	Vision Position Estimator delay relative to IMU measurements
3	EKF2_GPS_CHECK	0	Enable/Disable GPS
4	MAV_1_CONFIG	TELEM 2	Serial Configuration for MAVLink (Rpi to PX4)
5	SER_TEL2_BAUD	921600	Baud rate for the TELEM 2 Serial Port
6	Color	Disabled	Color disabled to save USB bandwidth
7	IR Emitter	Disabled	IR projected dot pattern interferes with VINS-Fusion's visual feature tracking
8	Infra and Depth	Enabled	Feed for VINS and EGO-Planner
9	Infra and Depth Freq	15 Hz	Frequency at which Raspberry Pi receives data
10	Resolution	420 * 240	Lower resolution to reduce image processing time and CPU usage

4.2.4 Pre-Flight Testing and Validation Protocol

To ensure the safety and reliability of the integrated system, testing progressed through three controlled stages before full autonomous deployment.

- i. **Closed-Loop Hover Validation:** An automated takeoff and hover test was executed utilizing the px4ctrl ROS position controller. The quadcopter was commanded to ascend and hold a 1.0 m altitude setpoint. Hover performance was monitored via the RViz visualization interface. If positional oscillations exceeded 0.5 m, the PID gain parameters (K_p , K_v) and hover thrust estimates were iteratively tuned until rigid station-keeping was achieved.
- ii. **Static Obstacle Avoidance Verification:** Prior to waypoint navigation, a controlled avoidance test was conducted. Static physical barriers were placed directly between the quadcopter and a single assigned goal point. Both the VINS-Fusion and EGO-Planner nodes were initialized via a centralized launch script. This test verified that the depth map was successfully converting into a 3D occupancy grid and that the planner could autonomously generate and execute a collision-free geometric detour around the barriers.
- iii. **Continuous Autonomous Waypoint Navigation:** Following successful static avoidance, full autonomous capability was evaluated. Multiple spatial waypoints were sequentially published to the navigation stack. The quadcopter was required to autonomously transition between waypoints, continuously replanning its trajectory in real-time to bypass unstructured environmental clutter.

4.3 Post-Flight Data Processing and Error Computation

All flight data was recorded onboard during each test and processed offline after the flights. This section describes the complete pipeline used to extract, align, and evaluate the trajectory tracking performance from the recorded ROS bag files, from raw binary data extraction through to the final RMSE and mean error metrics reported in Table 5.2.

4.3.1 Data Recording and Extraction

During each hardware flight test, all relevant ROS topics were recorded into bag files on a 256 GB SSD connected to the Raspberry Pi 4B companion computer. Two topics formed the basis of the tracking error analysis: `/vins_fusion/odometry`, publishing the VINS-Fusion position estimate at approximately 8 Hz, and `/position_cmd`, publishing the EGO-Planner commanded position at approximately 100 Hz. Where available, `/cpu_usage` and `/cpu_temp` topics were also recorded to characterize onboard computational load during flight.

Bag files were transferred offline to Google Colab and processed using the `rosbags` Python library (version 0.11.0). The `/vins_fusion/odometry` topic was deserialized normally using the standard ROS1 Noetic typestore. The `/position_cmd` topic uses the custom message type `quadrotor_msgs/PositionCommand`, which is not included in the standard typestore and could not be deserialized through the normal API. Instead, the raw binary payload was parsed manually using Python’s `struct` library, reading the header fields sequentially before extracting the three double-precision floating-point position values for x , y , and z .

4.3.2 Temporal Alignment and Frame Correction

The odometry recording begins at system initialization, well before the navigation mission starts. The `/position_cmd` topic, by contrast, only publishes once EGO-Planner receives an active goal and begins generating trajectories. The command signal’s time range was therefore used directly as the navigation evaluation window - trimming the odometry to the interval $[t_{\text{cmd,start}}, t_{\text{cmd,end}}]$. This approach cleanly isolates the active navigation phase without relying on heuristic displacement thresholds, and excludes takeoff, hover-wait, and landing transients from the error computation.

VINS-Fusion and EGO-Planner each initialize their own local coordinate frame independently at startup. A small spatial offset between the two frames is therefore expected. This offset was estimated by averaging the positional difference between the

first 20 matched samples of the trimmed odometry and command signals:

$$\delta_k = \frac{1}{20} \sum_{i=1}^{20} \left(p_k^{(i)} - c_k^{(i)} \right), \quad k \in \{x, y, z\} \quad (4.2)$$

where $p_k^{(i)}$ is the odometry position and $c_k^{(i)}$ is the commanded position at sample i . Using the first 20 samples provides a stable average before any significant drift accumulates, while remaining well within the early navigation phase. Across all flight tests, the estimated offsets were small, typically in the range of 0.01–0.10 m, confirming good initial frame alignment between the two subsystems.

Since the command signal was published at a significantly higher rate than the odometry, the command was resampled to the odometry timestamps using linear interpolation via `scipy.interpolate.interp1d`. The odometry timestamps define the evaluation grid, as they represent the lower-rate but more accurate state estimates from the sliding-window optimizer.

4.3.3 Error Metrics

After temporal alignment and frame offset correction, the per-axis tracking error at each timestep i was computed as:

$$e_k^{(i)} = p_k^{(i)} - \left(c_k^{(i)} + \delta_k \right), \quad k \in \{x, y, z\} \quad (4.3)$$

The three-dimensional Euclidean tracking error was then computed as:

$$e_{3D}^{(i)} = \sqrt{\left(e_x^{(i)} \right)^2 + \left(e_y^{(i)} \right)^2 + \left(e_z^{(i)} \right)^2} \quad (4.4)$$

Two summary metrics were computed over the full navigation window of N samples. The mean 3D error:

$$\bar{e}_{3D} = \frac{1}{N} \sum_{i=1}^N e_{3D}^{(i)} \quad (4.5)$$

treats all timesteps equally and gives a straightforward measure of average spatial deviation. The Root Mean Square Error:

$$\text{RMSE}_{3D} = \sqrt{\frac{1}{N} \sum_{i=1}^N \left(e_{3D}^{(i)} \right)^2} \quad (4.6)$$

penalizes large instantaneous deviations more heavily, making it more sensitive to brief excursions caused by replanning events or turbulence. Both metrics are reported in Table 5.2 alongside per-axis mean absolute errors.

CPU utilization and temperature data were extracted from the `/cpu_usage` and `/cpu_temp` topics, each published as `std_msgs/Float32` at approximately 1 Hz, and aggregated to compute mean and maximum values over the full flight duration. These are reported separately in Table 5.1 to characterize the onboard computational load during autonomous navigation.

Chapter 5

RESULTS AND DISCUSSION

5.1 Simulations

This section presents the quantitative results of the simulation validation procedure conducted in three progressive stages. The first stage evaluates the accuracy of the VINS-Fusion state estimator in an open-loop configuration, reporting the translational drift accumulated over a complete waypoint mission relative to the GPS ground truth. The second stage evaluates closed-loop performance, assessing the quadcopter's ability to accurately trace a predefined square trajectory under full VIO-based localization with GPS disabled. The third stage presents the full autonomous navigation test, where the complete integrated stack (VINS-Fusion, EGO-Planner, and the geometric controller) was validated in a simulated indoor environment, demonstrating real-time obstacle avoidance and goal-directed flight without any external or GNSS-aided positioning system.

5.1.1 Open Feedback Loop Odometry Analysis

The open-loop VIO estimation accumulated drift of $\Delta X = 0.390$ m, $\Delta Y = -0.176$ m, and $\Delta Z = 1.400$ m over the $10\text{ m} \times 10\text{ m}$ square mission. The Z-axis drift (1.400 m, 14% of flight altitude) significantly exceeded horizontal drift, attributable to repetitive floor textures in the simulated environment reducing vertical feature trackability.

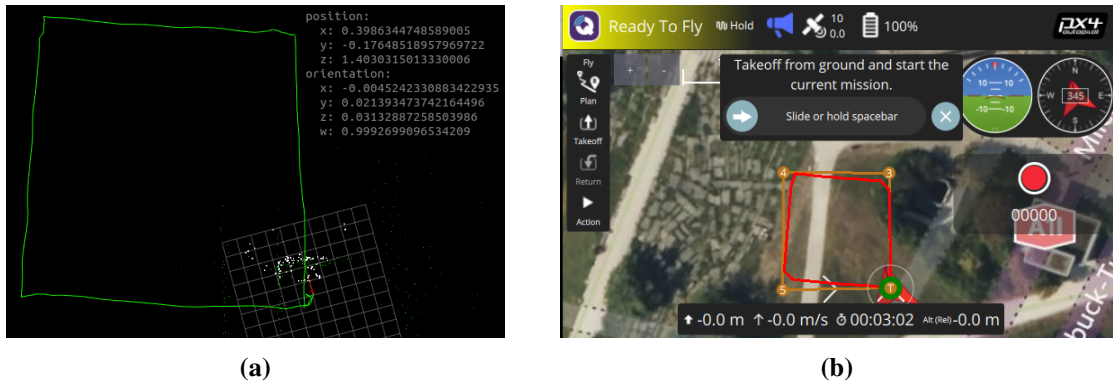


Figure 5.1: Open-loop VIO estimation: (a) VINS-Fusion estimated trajectory and (b) QGround-Control mission interface.

5.1.2 Closed Feedback Loop Odometry Analysis

The quadcopter was commanded to execute a $5\text{ m} \times 5\text{ m}$ square trajectory utilizing the estimator's odometry output for local position feedback. Upon mission completion, the estimator exhibited terminal drifts of $\Delta X = -0.026\text{ m}$, $\Delta Y = 0.100\text{ m}$, and $\Delta Z = 0.190\text{ m}$. The successful completion of this maneuver, maintaining stability at the trajectory vertices, verifies the low latency and accuracy of the integrated VIO-control loop.

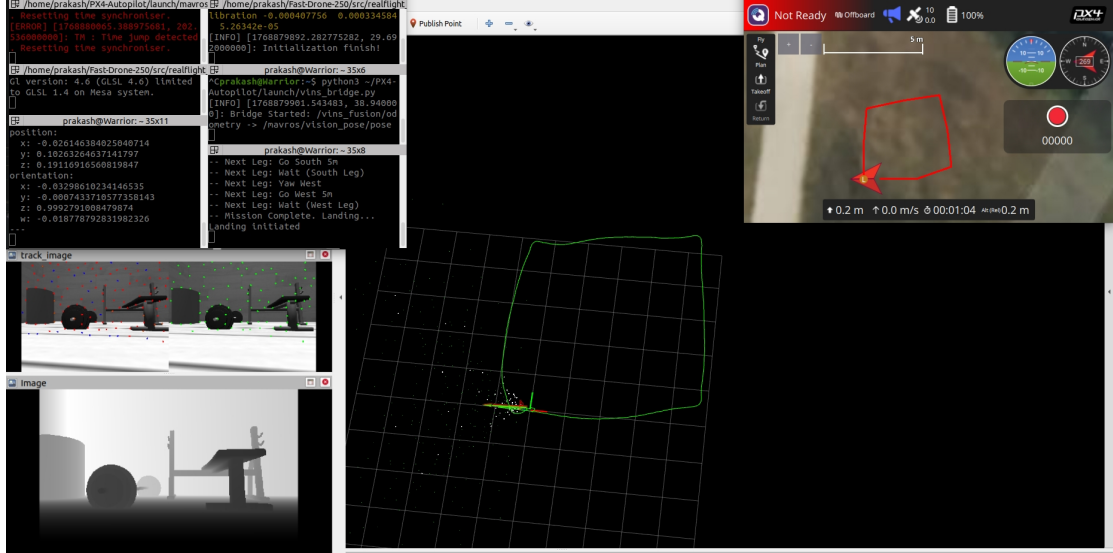


Figure 5.2: RViz visualization of closed-loop VINS-Fusion odometry estimation.

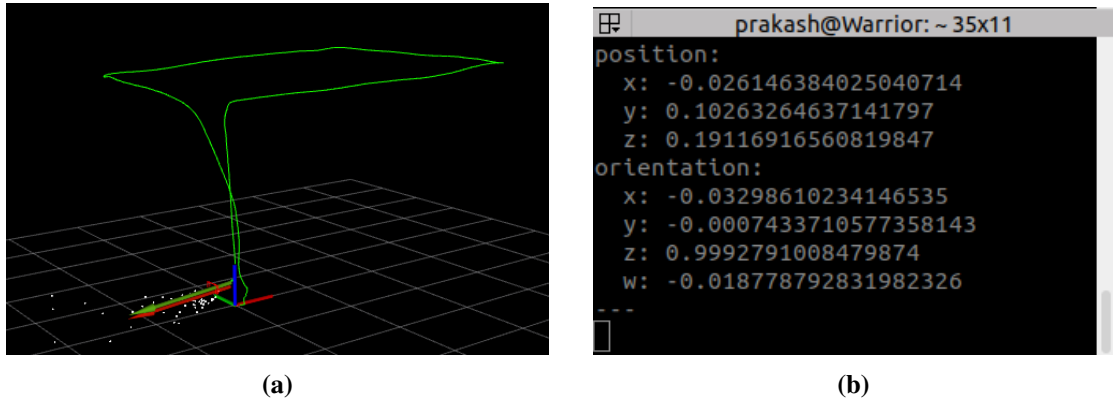


Figure 5.3: Closed-loop mission results in SITL: (a) estimated trajectory in RViz and (b) VINS-Fusion odometry output

5.1.3 Simulation of Autonomous Navigation

The simulation environment (Figure 5.4) consisted of a Gazebo-replicated spatial layout of the Manufacturing Lab. The environment featured an unstructured obstacle field comprising static, furniture-like objects distributed across the floor area bounded by

perimeter walls, providing a controlled virtual testbed for validating the full navigation stack.

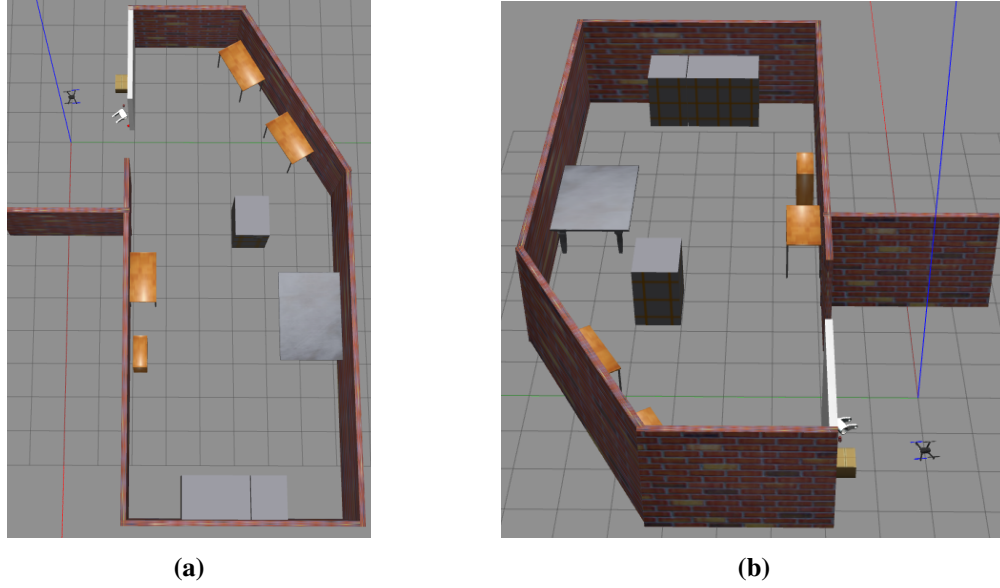


Figure 5.4: Simulation test environment setup of Manufacturing Lab in Gazebo

A two-waypoint mission profile was defined to navigate the quadcopter through the populated obstacle field, testing both horizontal avoidance routing and vertical altitude variations.

Figure 5.24b presents the executed 2D and 3D trajectories overlaid on the EGO-Planner occupancy grid. The quadcopter successfully navigated the mission with zero collisions. The 2D view illustrates how the planner routed the vehicle through free corridors between detected obstacles, while the 3D spatial trajectory confirms that the planner successfully exercised three-dimensional route generation.

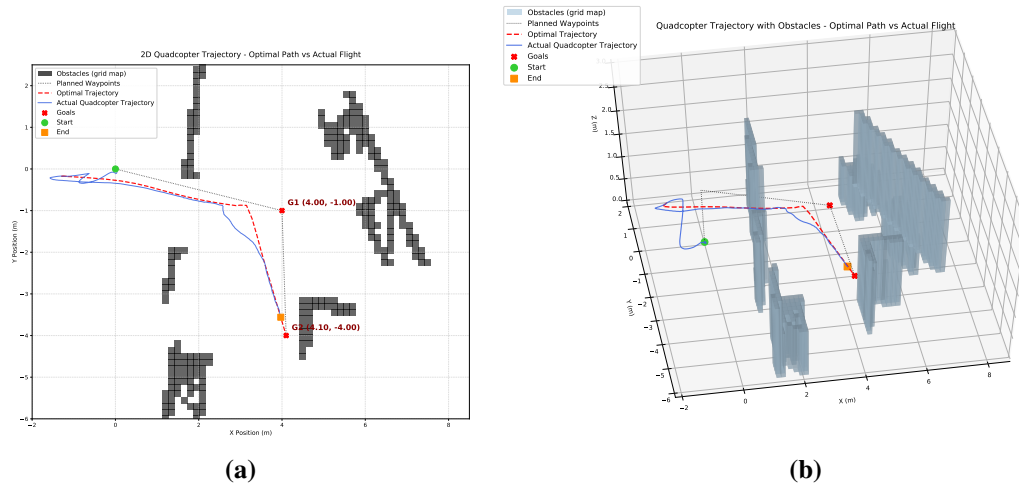


Figure 5.5: Quadcopter trajectory during mission(a) 2D top-down view with actual flight path and (b) 3D spatial trajectory commanded by the planner

The three-stage EGO-Planner occupancy grid evolution (Figure 5.6) demonstrates the obstacle detection capability in the simulated environment. Following a 0.3 m safety inflation around each obstacle cell, the planner successfully resolved a collision-free route. The final navigation overlay confirms collision-free path execution through the available free space between inflated obstacle regions.

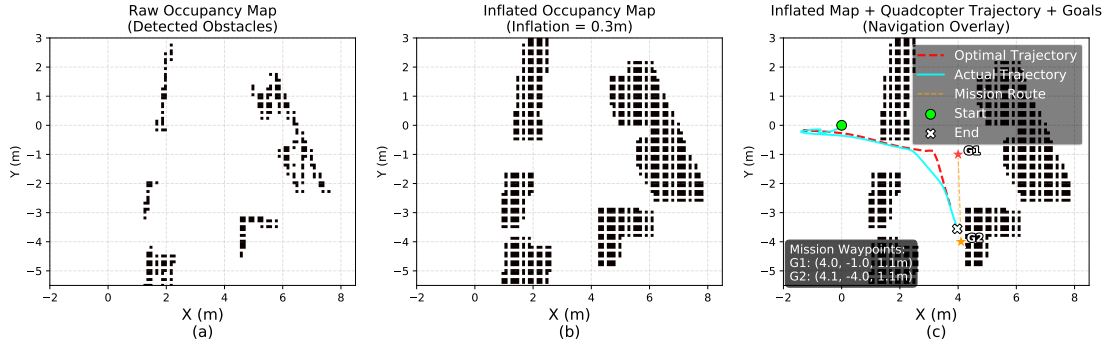


Figure 5.6: 2D EGO-Planner occupancy grid: (a) raw occupancy map of obstacles detected (b) inflated occupancy map and (c) navigation overlay .

Per-axis tracking analysis (Figure 5.7) recorded the commanded versus actual position over the full flight duration. The X and Y channels demonstrated mean navigation errors of 0.364 m and 0.162 m respectively during the active navigation phase.

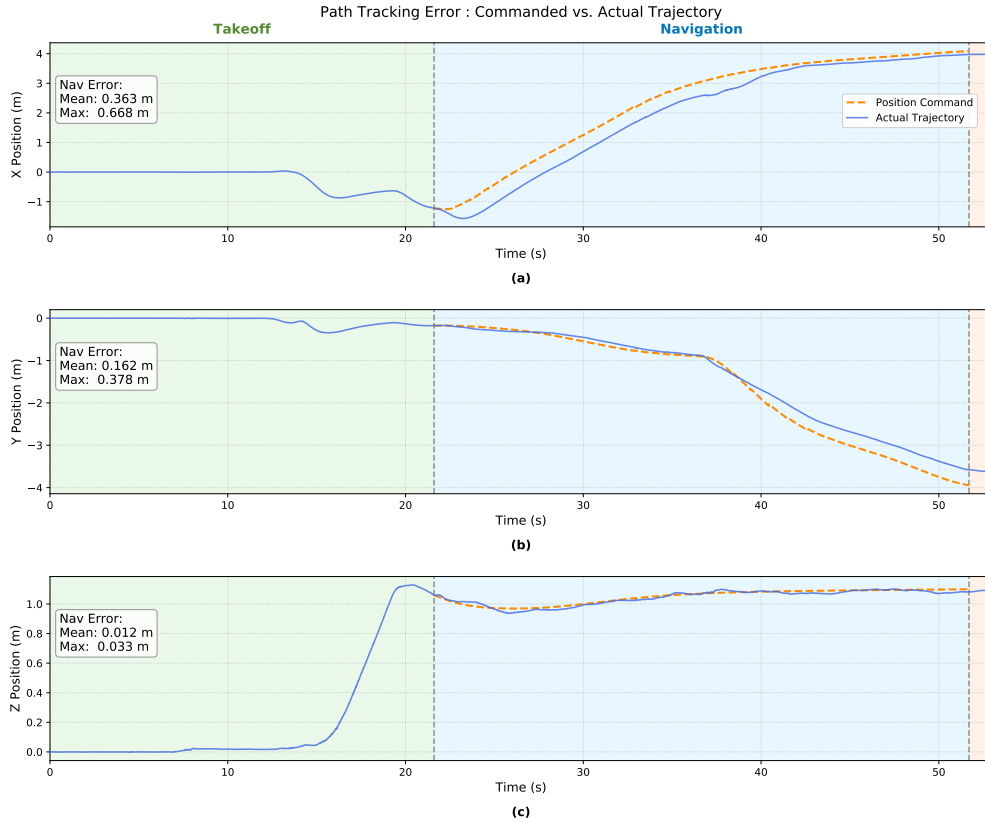


Figure 5.7: Per-axis path tracking error during Takeoff and Navigation phase

The commanded speed profile was maintained with a mean tracking error of 0.037 m/s and a standard deviation of 0.052 m/s (Figure 5.8). Transient deviations were most prominent at the start of the navigation phase, consistent with initial acceleration demands, before the geometric controller converged closely with the commanded profile.

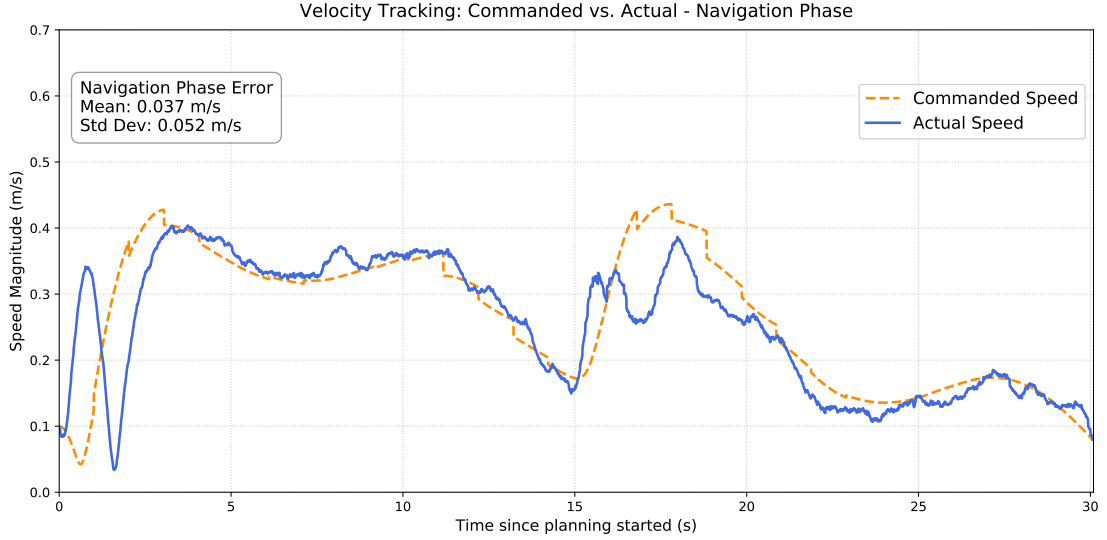


Figure 5.8: Velocity tracking performance during the Navigation phase

5.2 System Integration, Calibration and Initial Tests

Following the successful validation of the navigation stack in simulation, the complete system was deployed on the physical quadcopter to evaluate real-world performance. Flight testing was conducted in three stages of increasing complexity. First, a baseline flight using only the Pixhawk 4's onboard IMU was performed to characterize the drift characteristics of inertial-only navigation and to establish a reference for comparison with the full VIO stack. This was followed by VINS-Fusion calibration flights and a closed-loop hover test to verify the stability of the VIO-control loop on real hardware. Finally, full autonomous navigation with obstacle avoidance was evaluated in distinct environments, including an outdoor test at the Aerospace Garden, testing the integrated stack under real-world conditions.

5.2.1 Initial Flight Testing Using IMU

The initial flight tests were conducted relying exclusively on the internal inertial sensors and barometer of the flight controller. The objective of this phase was to establish a quantitative baseline for flight stability and attitude control prior to the integration of visual odometry.

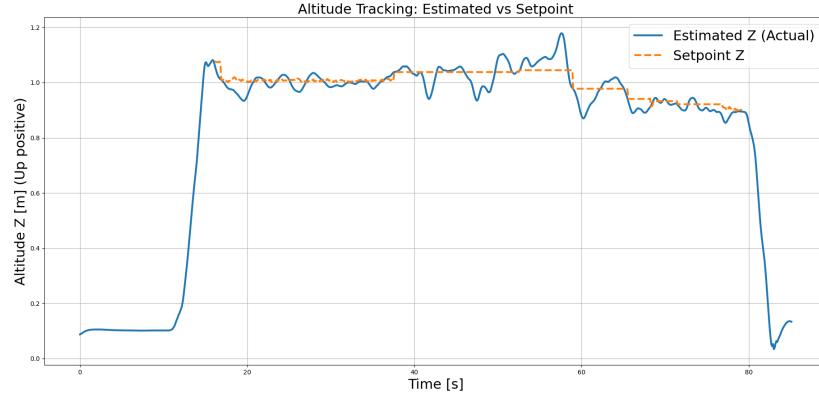


Figure 5.9: Vertical state estimation during hover using internal IMU and barometer

The vertical state estimation performance during a hover (Figure 5.9) exhibited characteristic baro-inertial instability. While the vehicle broadly tracked the 1.0 m setpoint, the estimated Z position demonstrated continuous low-frequency oscillations with tracking errors ranging from ± 0.05 m to a maximum transient deviation of 0.15 m. These deviations quantify the inherent noise and drift limitations of utilizing dead-reckoning and pressure data without external position corrections.

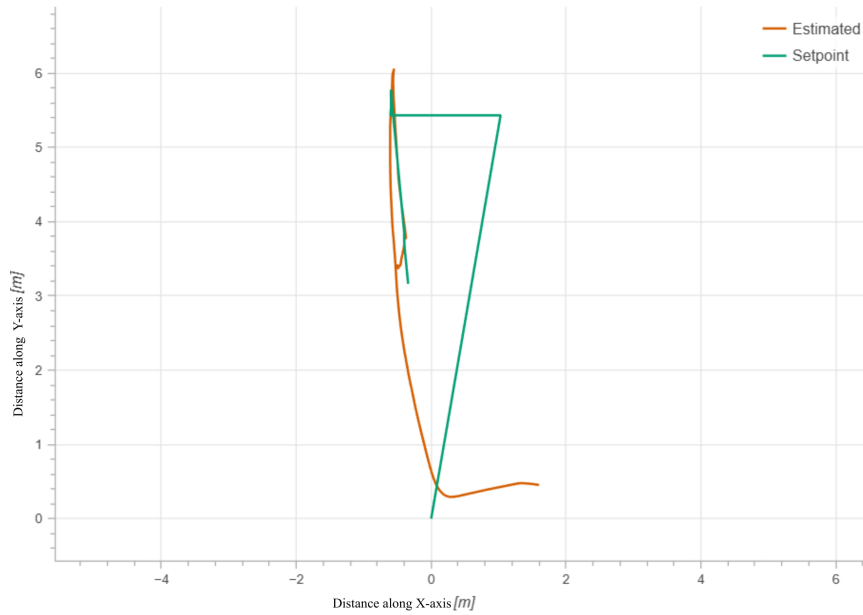


Figure 5.10: Horizontal (XY) divergence during OFFBOARD-commanded flight using IMU-only localization

During an OFFBOARD-commanded flight relying solely on internal inertial navigation, the lateral (XY) trajectory exhibited critical dead-reckoning divergence (Figure 5.10). The accumulated double-integration errors caused the vehicle to fail in tracing the geometric path, confirming the necessity of external visual corrections for precise horizontal positioning.

5.2.2 VINS Calibration Test

To evaluate the impact of IMU-camera extrinsic calibration on estimator accuracy, a manual flight was performed before and after the calibration procedure following an identical flight path. Prior to calibration, the estimator exhibited significant accumulated drift of $\Delta X = -0.367$ m, $\Delta Y = -0.993$ m, and $\Delta Z = +0.244$ m over the complete flight, with the Y-axis error indicating poor lateral consistency. Following calibration, the drift was substantially reduced to $\Delta X = -0.059$ m, $\Delta Y = -0.081$ m, and $\Delta Z = +0.177$ m. This represents an 81.3% reduction in horizontal drift. The end point visibly converged toward the start point after calibration (Figure 5.11), confirming that the corrected extrinsic parameters significantly improved the consistency of the VIO estimate.

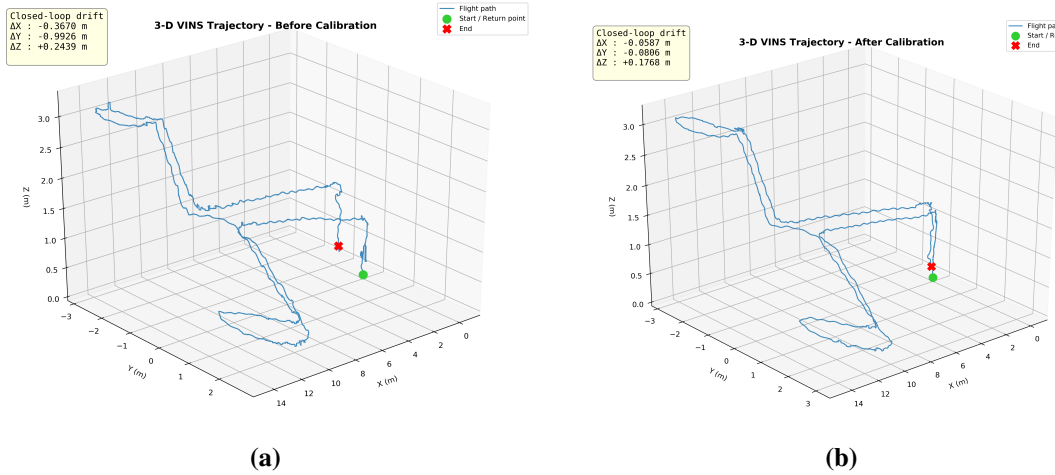


Figure 5.11: Effect of IMU-camera extrinsic calibration on VINS-Fusion: (a) Before calibration and (b) After calibration

5.2.3 Hover Test

The quadcopter was commanded to take off and maintain a hover altitude of 1.0 m. Altitude was measured simultaneously by two independent sources: the VINS Z estimate from the visual-inertial odometry pipeline, and the VL53L1X time-of-flight distance sensor mounted on the Pixhawk 4. As shown in Figure 5.12, the VINS Z channel maintained a mean error of 0.036 m ($\sigma = 0.022$ m) relative to the 1.0 m command, demonstrating tight altitude regulation throughout the hover phase. The distance sensor recorded a mean offset of 0.115 m ($\sigma = 0.030$ m), attributable to the sensor's sensitivity to floor reflectance and surface texture.

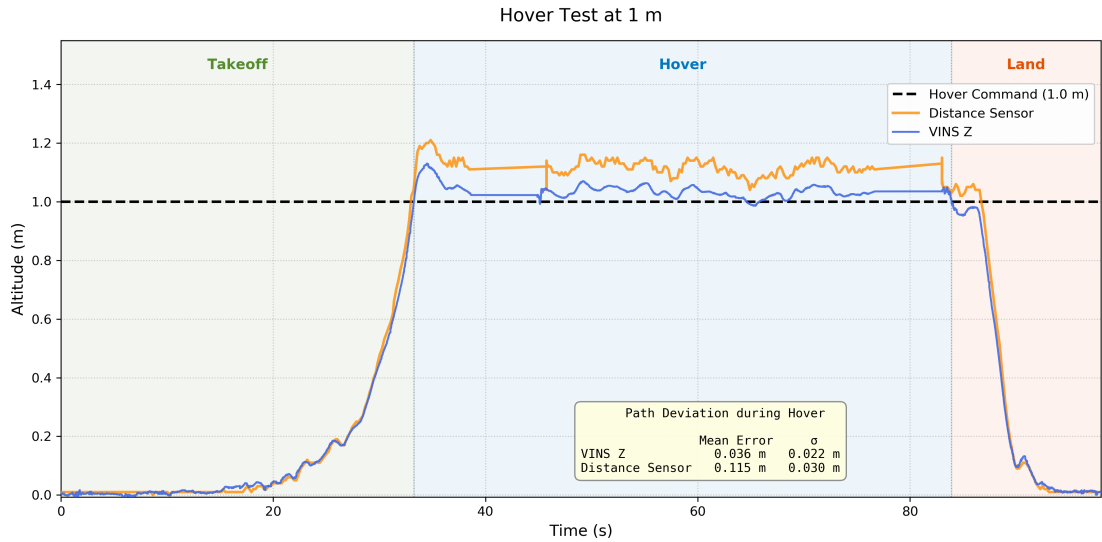


Figure 5.12: Altitude time series during autonomous hover at a commanded height of 1.0 m

5.2.4 Obstacle Avoidance Test

The Aerospace Garden at Pulchowk Campus provided an open outdoor test environment consisting of a grass field (Figure 5.13). The test was conducted in two configurations: first without any obstacles to establish a baseline trajectory, and then with three large flat-panel obstacles placed in the path to trigger the EGO-Planner's real-time replanning response.



Figure 5.13: Test setup at Aerospace Garden: (a) baseline configuration with no obstacles and (b) With obstacle configuration.

A three waypoint triangular mission covering approximately $5 \text{ m} \times 4 \text{ m}$ was commanded. Identical waypoints (G1: (5,0), G2: (5,4), G3: (0,0)) were used at a flight altitude of 1.0 m for two distinct runs: a baseline run without obstacles and an active run with obstacles to trigger the EGO-Planner's real-time replanning response.

Figure 5.14 compares the 2D top-down trajectories for both runs. In the baseline run, the executed trajectory closely followed the optimal straight-line paths across all three waypoints. During the obstacle run, the EGO-Planner successfully deviated the trajectory around both flat panels, with the quadcopter visibly curving away from the obstacle boundaries before converging back toward the planned route to reach all goals.

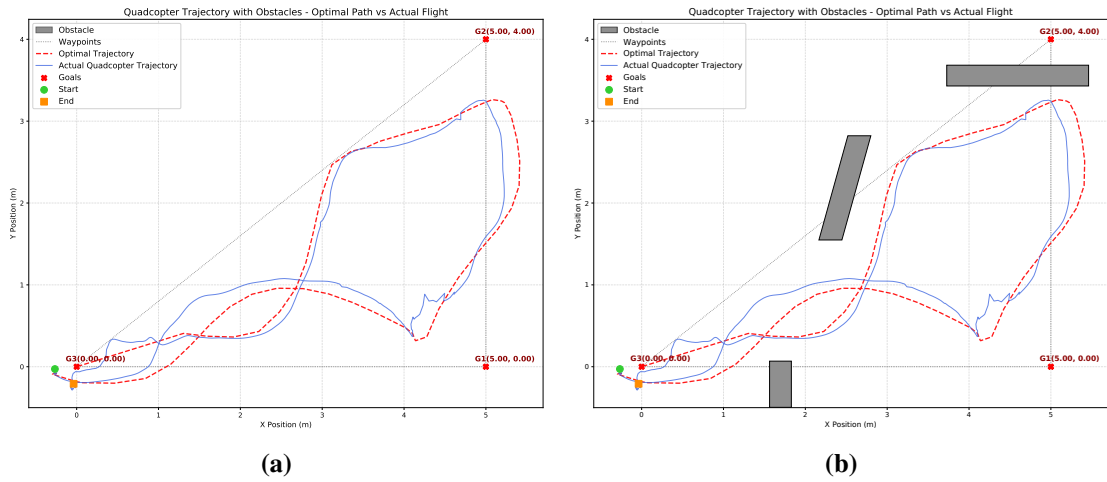


Figure 5.14: Comparison of 2D top-down quadcopter trajectories: (a) baseline run without obstacles and (b) run with obstacles

The per-axis tracking comparison (Figure 5.15) quantifies the effect of avoidance maneuvers on path adherence. In the baseline run, the mean tracking errors were 0.165 m (X) and 0.079 m (Y). When obstacles were introduced, the mean errors increased moderately to 0.160 m (X) and 0.102 m (Y), directly reflecting the lateral deviations required for collision avoidance.

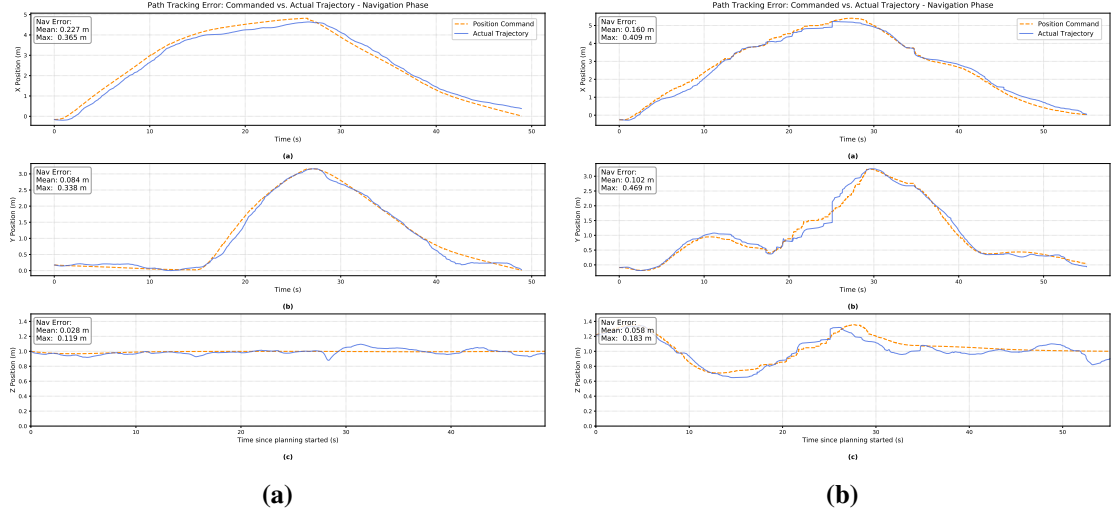


Figure 5.15: Per-axis path tracking error during Navigation phase: (a) without obstacles and (b) with obstacles

The Z-axis remained the best-tracked axis in both configurations. The baseline run yielded a mean Z error of 0.024 m. During the obstacle run, the mean Z error rose slightly to 0.058 m, reflecting the vertical altitude adjustments commanded by the planner to safely clear the taller panel.

5.3 Indoor Autonomous Navigation Tests

After calibration and hover stability were confirmed, the complete navigation stack was tested in two indoor environments on Pulchowk Campus. The quadcopter navigated without GNSS reliance, depending entirely on the integrated VINS-Fusion and EGO-Planner stack. Results are quantified in terms of trajectory tracking, spatial deviation, velocity regulation, and real-time occupancy mapping.

5.3.1 Test Environment 1: IIEC, Pulchowk Campus

The first indoor test was conducted at the IIEC building, Pulchowk Campus (Figure 5.16). The testing area comprised a large open hall with high ceilings, scattered industrial equipment, wood panels, and debris distributed across the floor. This configuration provided a naturally cluttered and unstructured environment, ideal for evaluating the real-time obstacle avoidance capabilities of the navigation stack.



Figure 5.16: Test environment at IIEC, Pulchowk Campus

The quadcopter was commanded to execute a four-waypoint rectangular mission covering an area of approximately $9 \text{ m} \times 6.5 \text{ m}$. The flight path intentionally incorporated distributed obstacles to trigger and evaluate the EGO-Planner's real-time replanning response under continuous forward flight.

Figure 5.17 presents the executed 2D and 3D trajectories relative to the optimal path. The 2D top-down view illustrates the quadcopter's lateral deviations around obstacle clusters, while the 3D spatial trajectory confirms the planner's capacity to command vertical altitude variations, opting to climb over specific obstacles rather than bypass them laterally. Across the mission, the executed trajectory successfully adhered to the general route while demonstrating continuous, collision-free replanning.

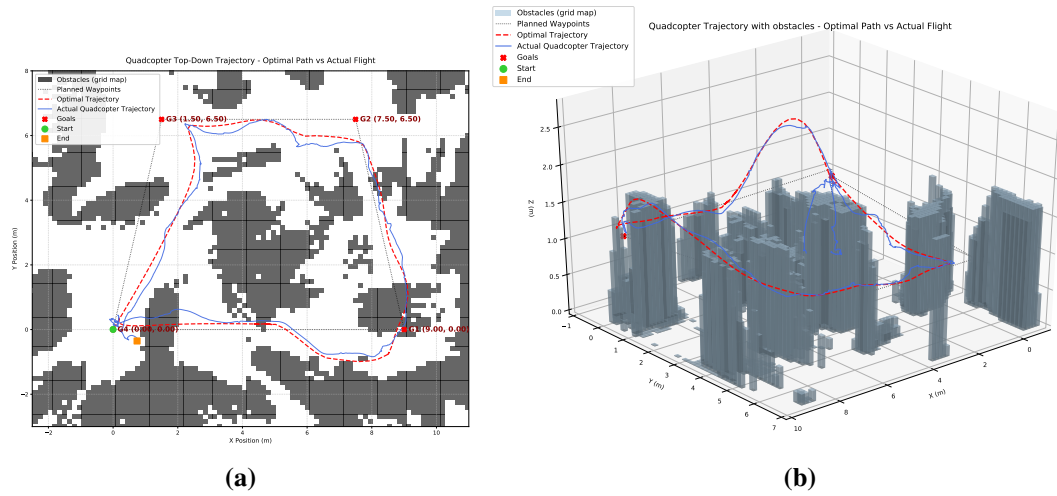


Figure 5.17: Quadcopter trajectory during mission at IIEC: (a) 2D top-down view with actual flight path (b) 3D spatial trajectory with altitude variations commanded by the planner

The occupancy grid (Figure 5.18) demonstrates the depth sensor's obstacle detection capability in unstructured indoor conditions. The mapping pipeline successfully transitioned from the raw point cloud to an inflated occupancy map utilizing a 0.3 m safety margin. The final navigation overlay confirms collision-free path execution, with the planned trajectory routing smoothly through navigable corridors between inflated obstacle boundaries.

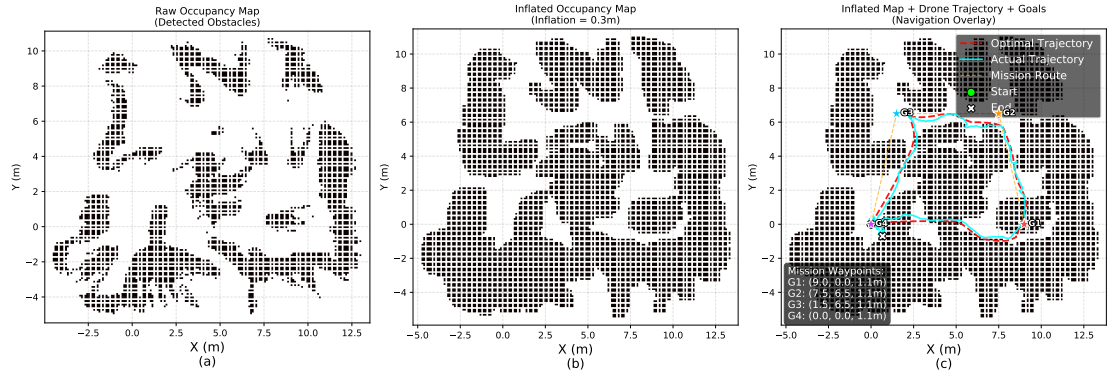


Figure 5.18: 2D EGO-Planner occupancy grid of IIEC: (a) raw occupancy map of obstacles detected (b) inflated occupancy map and (c) navigation overlay

Per-axis tracking analysis (Figure 5.19) details the commanded versus actual position over the complete flight duration. The X and Y axes exhibited mean navigation errors of 0.086 m and 0.125 m respectively. The slightly elevated Y-axis deviation correlated with the middle segment of the mission, coinciding with the region of highest obstacle density where lateral avoidance maneuvers were concentrated.

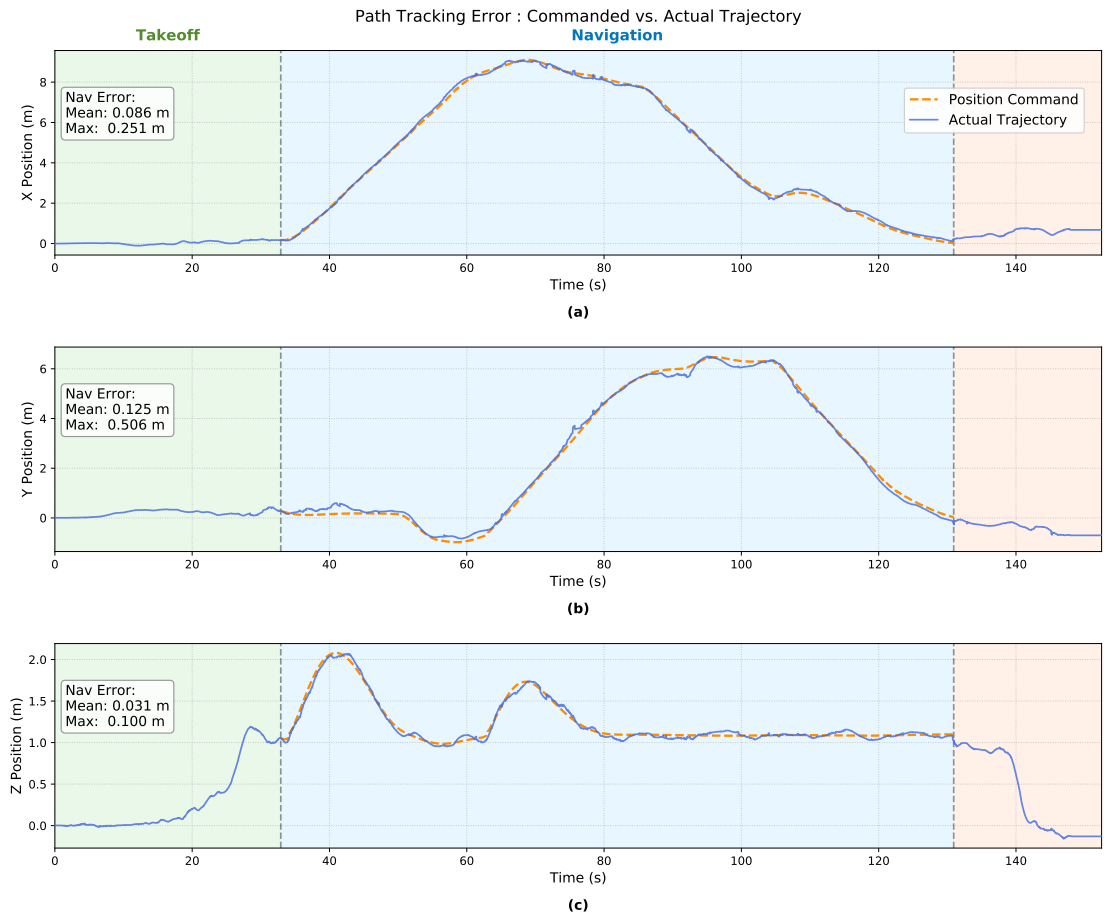


Figure 5.19: Per-axis path tracking error over the complete takeoff, navigation, and landing sequence at IIEC

The Z-axis maintained the tightest tracking throughout the flight (Figures 5.19 and 5.20). The VINS Z-estimate tracked the 1.0 m setpoint with an RMSE of 0.037 m and a maximum deviation of 0.100 m. Independent corroboration from the distance sensor confirmed these measurements, validating highly reliable altitude regulation during complex lateral and vertical avoidance maneuvers.

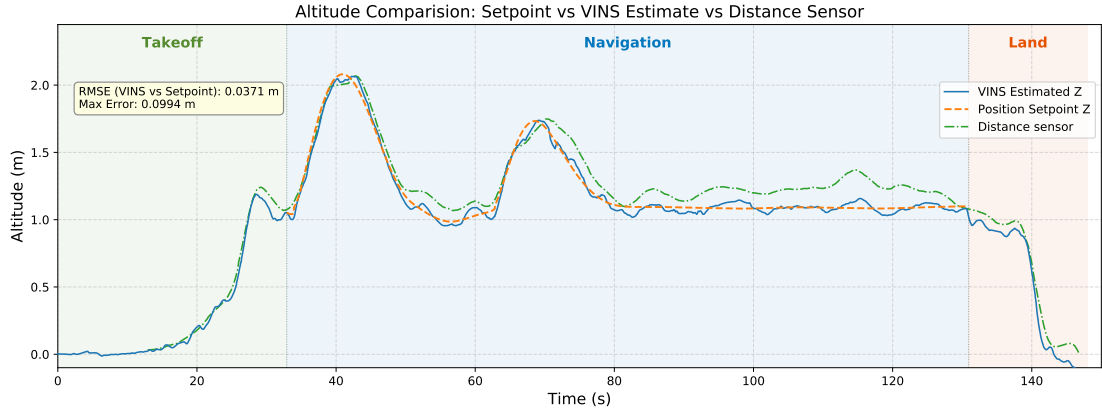


Figure 5.20: Altitude time series during the complete autonomous flight at IIEC

The 3D Euclidean distance error (Figure 5.21) quantifies the absolute spatial deviation from the commanded path. The system achieved a mean 3D tracking error of 0.308 m with a standard deviation of 0.137 m during the active navigation phase. Error peaks predominantly aligned with waypoint transitions and dense obstacle encounters, reflecting necessary planner-induced deviations from the optimal straight-line path.

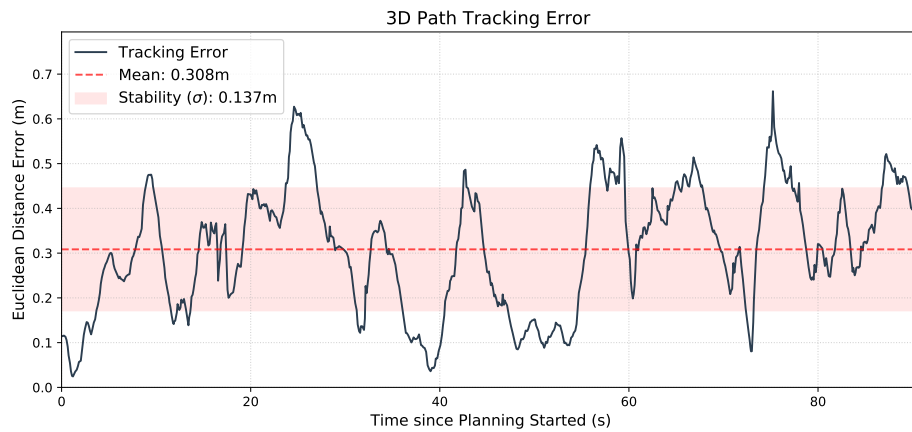


Figure 5.21: 3D Euclidean path tracking error over the navigation phase at IIEC

The commanded speed profile was accurately maintained (Figure 5.22), yielding a mean velocity tracking error of 0.041 m/s and a standard deviation of 0.036 m/s. This confirms smooth velocity regulation by the geometric controller. Brief transient spikes in velocity error corresponded directly to dynamic replanning events triggered by newly detected obstacles entering the depth sensor's field of view.

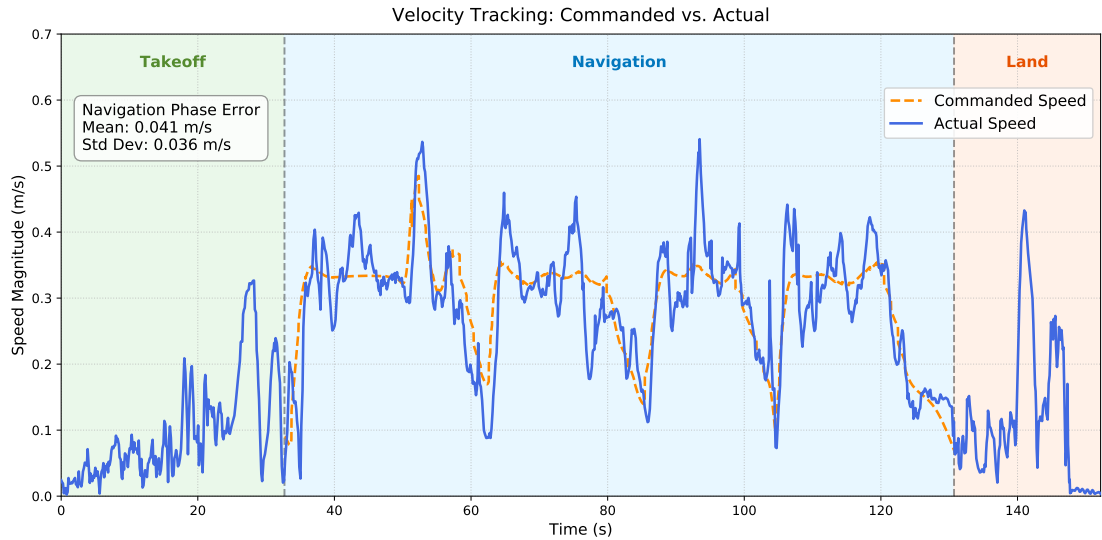


Figure 5.22: Velocity tracking performance over the complete flight at IIEC

5.3.2 Test Environment 2: Manufacturing Lab, Aerospace D2 Block

The second indoor test was conducted in the Manufacturing Lab of the Mechanical and Aerospace Department, D2 Block (Figure 5.23). The space consisted of an indoor workspace densely populated with specialized industrial equipment and structures, presenting a highly unstructured obstacle field for the navigation stack to negotiate.



Figure 5.23: Test environment at the Manufacturing Lab, Aerospace D2 Block

The quadcopter was tasked with a sequential two waypoint mission at a commanded altitude of 1.0 m. The mission required navigating from the takeoff origin through the obstacle field to reach goal point G1 (4.00, -1.00), followed immediately by a transition to goal point G2 (4.10, -4.00).

Figure 5.24 presents the 2D and 3D trajectory plots for the mission. The quadcopter successfully navigated the cluttered obstacle field, reaching both G1 and G2 with zero collisions. The actual flight path adhered closely to the EGO-Planner's optimal tra-

jectory, effectively routing through available free space while adapting to the physical obstacles.

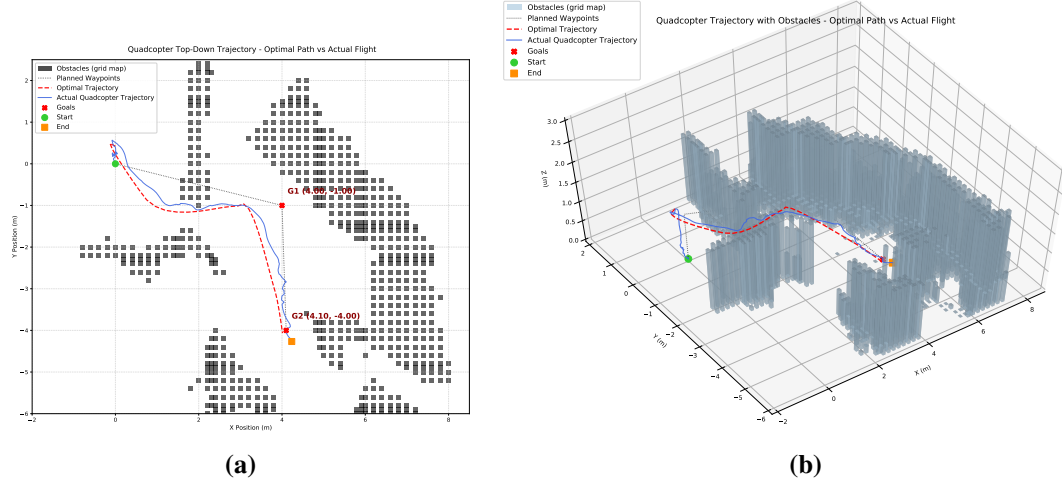


Figure 5.24: Quadcopter trajectory during mission at Manufacturing Lab: (a) 2D top-down trajectory (b) 3D spatial trajectory of the mission.

The three-stage EGO-Planner occupancy grid (Figure 5.25) illustrates the real-time mapping and obstacle inflation process. The depth camera successfully captured the complex obstacle distribution (raw map), which was subsequently expanded using a 0.3 m safety inflation radius. The final navigation overlay confirms that the planned and executed trajectories remained safely within the computed navigable free space.

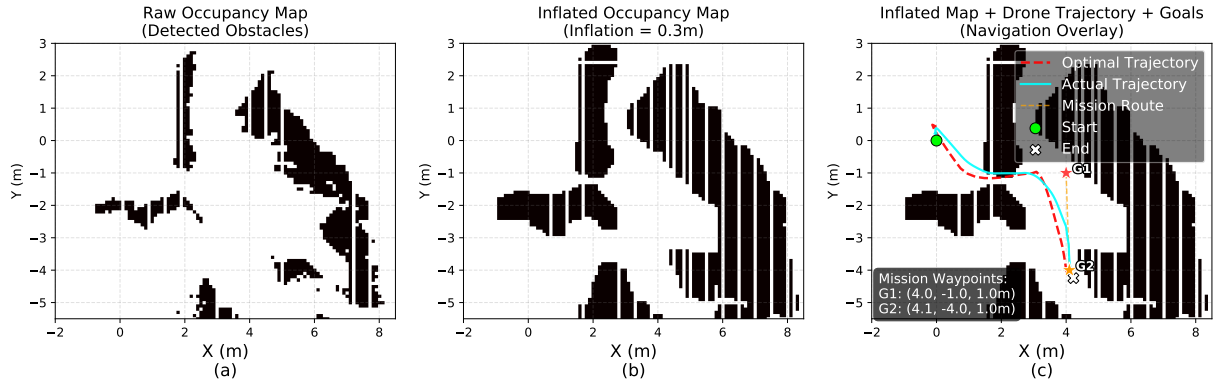


Figure 5.25: 2D EGO-Planner occupancy grid of Manufacturing Lab: (a) raw occupancy map (b) inflated occupancy map and (c) navigation overlay

Per-axis tracking analysis (Figure 5.26) quantifies the positional deviation between the command and the actual trajectory during the autonomous navigation phase (blue shaded region). The horizontal axes exhibited mean navigation errors of 0.132 m (X) and 0.130 m (Y), with maximum deviations of 0.655 m and 0.364 m, respectively. These deviations reflect the dynamic lateral maneuvering required to bypass dense obstacle clusters.

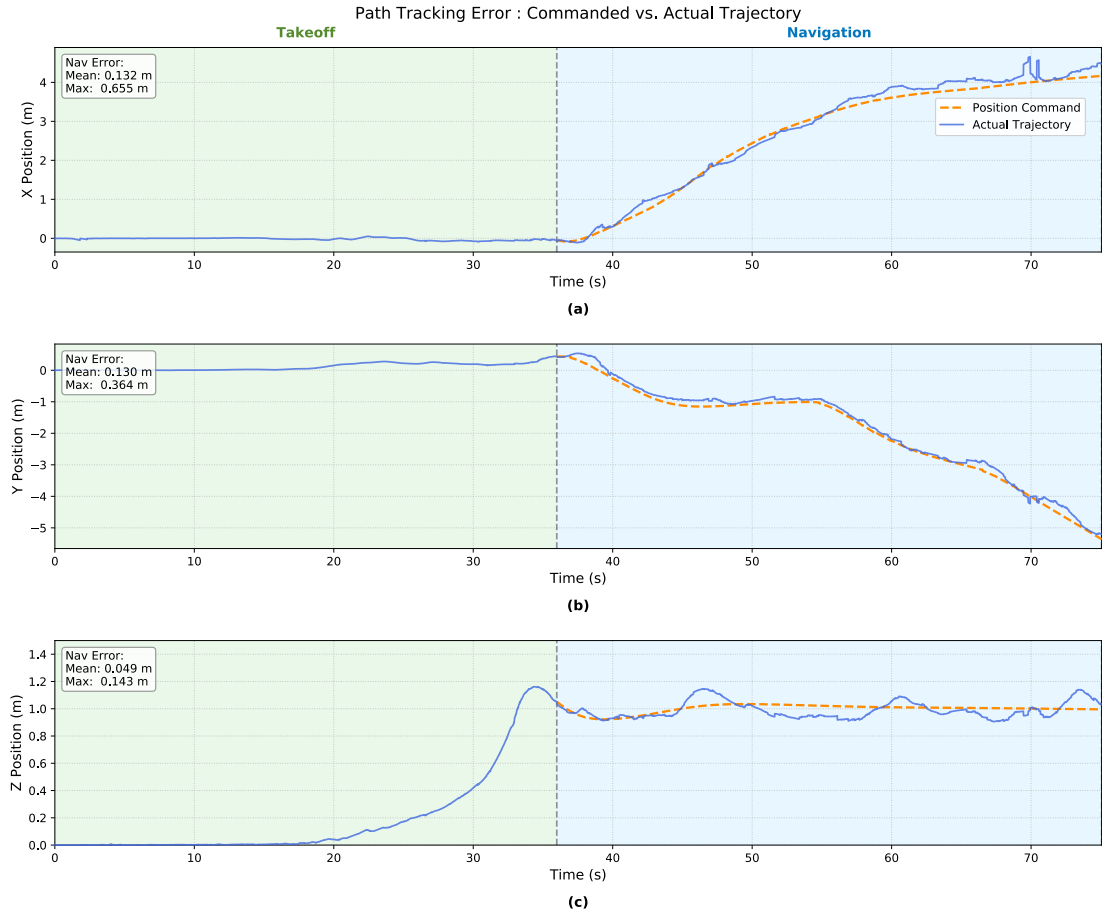


Figure 5.26: Per-axis path tracking error over the takeoff and navigation phases at the Manufacturing Lab

The Z-axis demonstrated the highest tracking precision throughout the navigation phase. As detailed in Figure 5.27, the quadcopter maintained a mean Z-axis error of 0.0487 m and a maximum deviation of 0.0941 m relative to the commanded height of 1.0 m, confirming highly stable altitude regulation during active avoidance maneuvers.

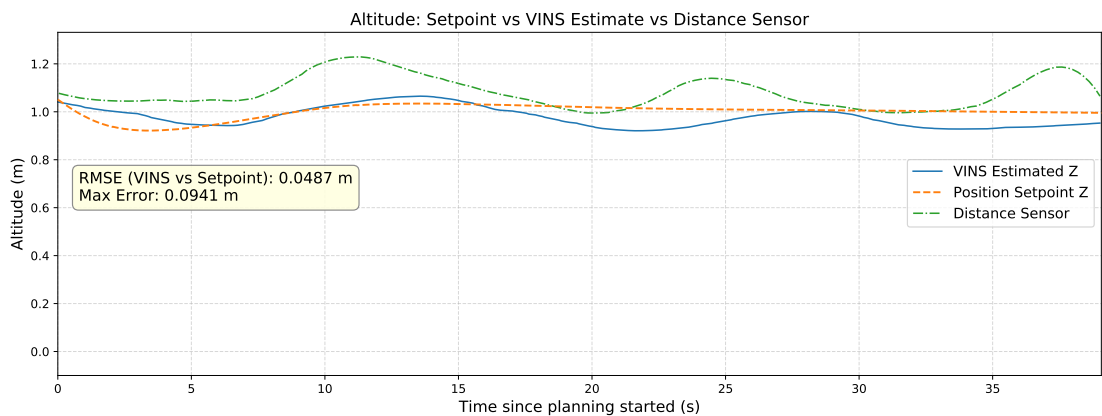


Figure 5.27: Altitude time series during the Navigation Phase at Manufacturing Lab

The 3D Euclidean distance error (Figure 5.28) aggregates the spatial tracking perfor-

mance. The quadcopter achieved a mean 3D tracking error of 0.277 m with a standard deviation of 0.127 m. The error exhibited transient peaks up to 0.43 m mid-flight and reached a maximum of 0.70 m near the final goal, a combined result of accumulated VIO drift and sharp planner deviations necessitated by complex obstacles near the G2 coordinate.

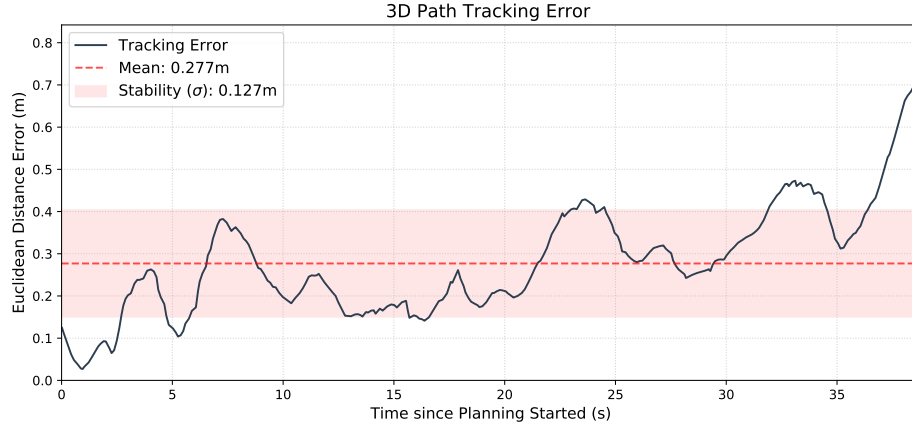


Figure 5.28: 3D Euclidean path tracking error over the navigation phase at the Manufacturing Lab

Velocity tracking performance (Figure 5.29) yielded a mean error of 0.059 m/s with a standard deviation of 0.044 m/s. An initial acceleration overshoot reached 0.50 m/s against a 0.33 m/s command before converging. Increased velocity oscillations (up to 0.41 m/s) observed between $t = 30$ s and $t = 38$ s directly correspond to the complex avoidance maneuvers executed while approaching the densely cluttered final waypoint.

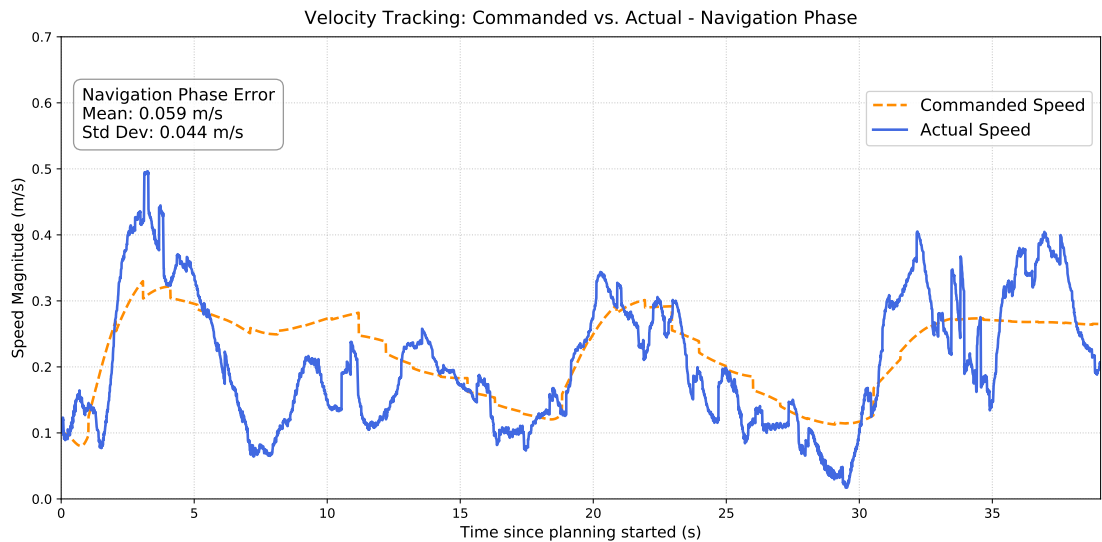


Figure 5.29: Velocity tracking performance during the autonomous navigation phase at the Manufacturing Lab

5.4 Outdoor Autonomous Navigation Tests: Aerospace Garden

Following successful indoor validation, the complete navigation stack was evaluated in an outdoor environment at the Aerospace Garden, Pulchowk Campus. This test was designed to assess the robustness of the integrated system under ambient wind disturbances and varying outdoor lighting conditions, relying exclusively on the onboard navigation stack without GNSS assistance.

The outdoor test environment (Figure 5.30) comprised an open garden space featuring natural terrain and physical boundaries. The outdoor conditions introduced unmodeled aerodynamic disturbances (wind) and dynamic lighting variations, providing a rigorous test for the visual-inertial odometry and control systems.



Figure 5.30: Test setup at Aerospace Garden, Pulchowk Campus

The quadcopter was commanded to execute a continuous six-waypoint rectangular loop covering an area of $8 \text{ m} \times 2 \text{ m}$. This profile forced the navigation stack to continuously map its surroundings and execute real-time path replanning to avoid garden features while compensating for environmental drift.

The 2D and 3D trajectory plots (Figure 5.31) illustrate the execution of the $8 \text{ m} \times 2 \text{ m}$ rectangular mission. The 2D top-down projection demonstrates the actual flight path closely adhering to the EGO-Planner's optimal trajectory while safely navigating the detected obstacle grid. The 3D spatial perspective confirms the quadcopter successfully reached all six designated waypoints, maintaining a stable altitude profile.

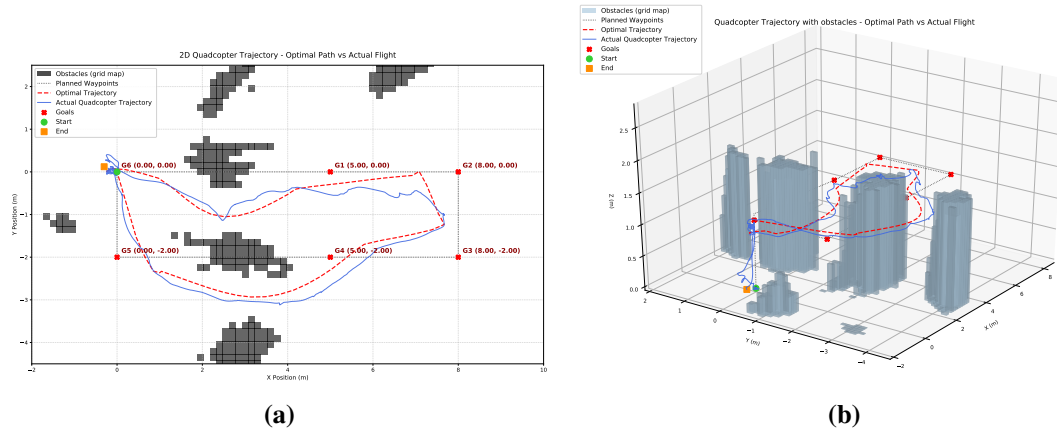


Figure 5.31: Quadcopter trajectory during mission at Aerospace Garden (a) 2D top-down view with actual flight path and (b) 3D spatial trajectory commanded by the planner

The EGO-Planner successfully adapted to the outdoor conditions (Figure 5.32). The system generated a raw occupancy map of detected environmental features, subsequently applying a 0.3 m safety inflation boundary to ensure adequate clearance. The final navigation overlay confirms that the quadcopter's executed trajectory successfully routed around these buffered hazard zones to complete the mission.

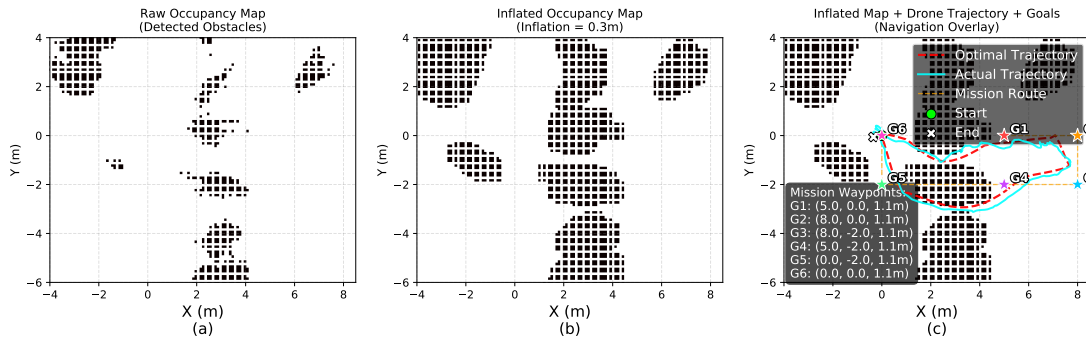


Figure 5.32: 2D EGO-Planner occupancy grid of Aerospace Garden: (a) raw occupancy map (b) inflated occupancy map and (c) navigation overlay

Per-axis path tracking analysis (Figure 5.33) quantifies the positional deviation across the takeoff, navigation, and landing phases. The quadcopter maintained reliable adherence to the commanded route despite outdoor wind disturbances. The most significant deviation occurred along the X-axis, yielding a mean error of 0.264 m and a transient maximum drift of 0.854 m, directly attributable to the system compensating for aerodynamic perturbations.

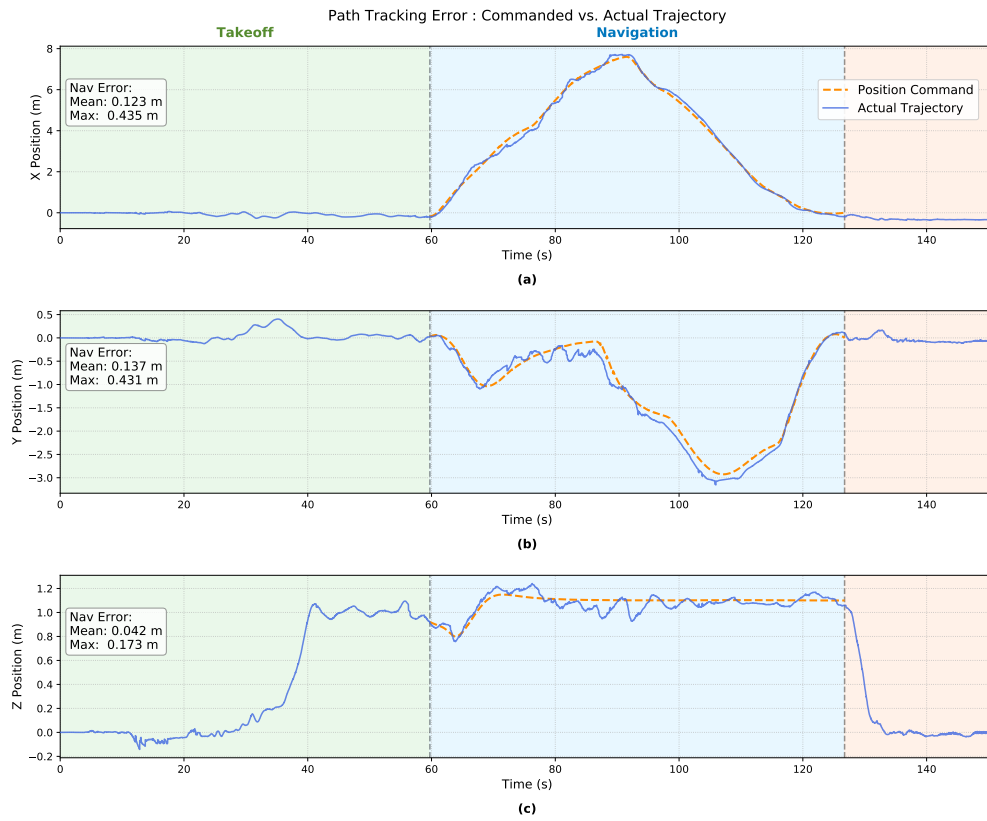


Figure 5.33: Per-axis path tracking error over the takeoff, navigation and landing phases at the Aerospace Garden

The Z-axis demonstrated precise tracking (Figure 5.34). Over the 60-second flight, the VINS Z-estimate tightly matched the position setpoint, recording a mean altitude error of just 0.051 m and a maximum transient error of 0.114 m. Independent distance sensor telemetry corroborated the flight altitude, proving the system reliably maintained vertical stability in an outdoor setting.

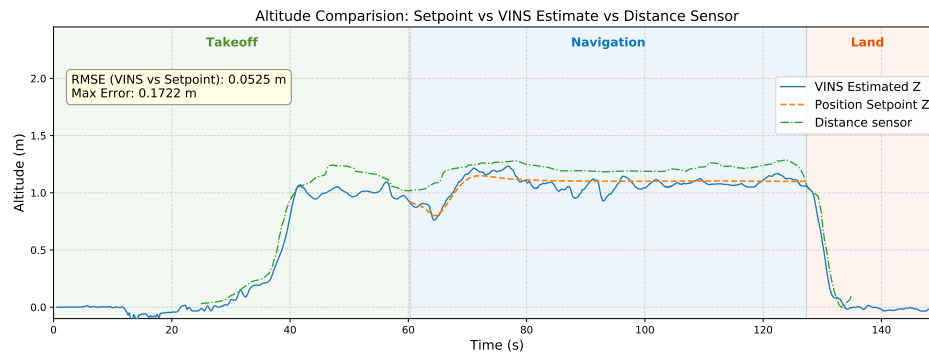


Figure 5.34: Altitude time series during the outdoor flight at Aerospace Garden

The 3D Euclidean distance error (Figure 5.35) evaluates the absolute spatial deviation over the 50-second active navigation phase. The system recorded a mean 3D tracking error of 0.316 m with a standard deviation of 0.140 m. While normal fluctuations

occurred as the quadcopter counteracted wind currents, the relatively low standard deviation confirms a consistently stable and accurate flight trajectory.

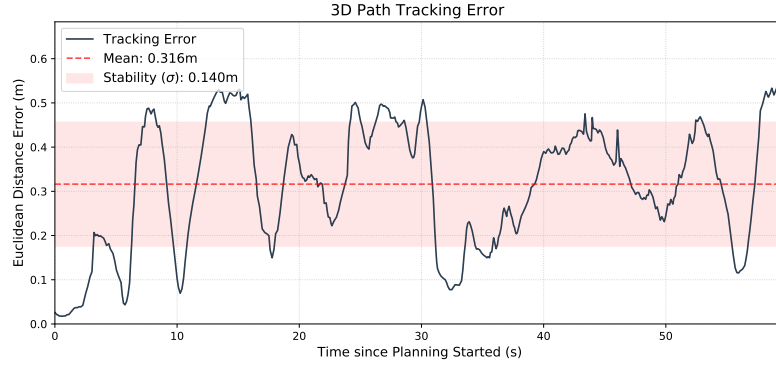


Figure 5.35: 3D Euclidean path tracking error over the navigation phase at Aerospace Garden

Velocity tracking performance (Figure 5.36) compares the commanded speed magnitude against the actual speed generated from the VINS-Fusion odometry. During the navigation phase, the system achieved a mean velocity error of 0.053 m/s with a standard deviation of 0.050 m/s. This confirms that the geometric controller successfully regulated the quadcopter’s speed despite external aerodynamic variables.

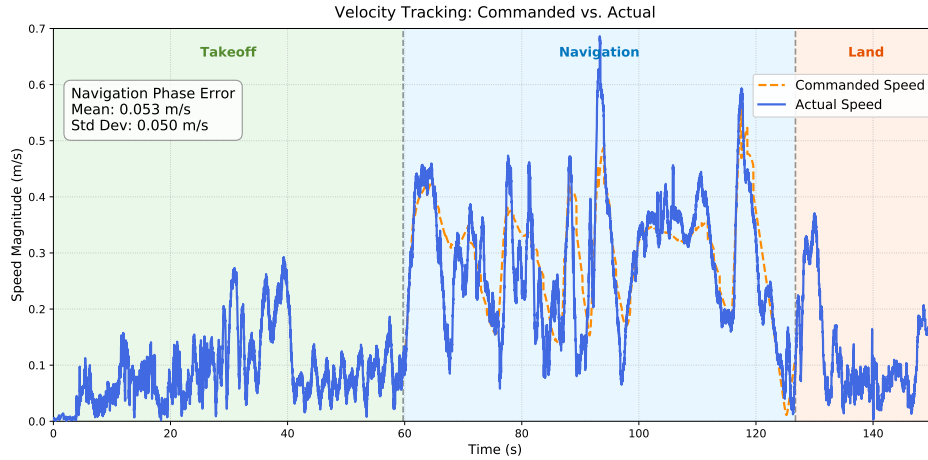


Figure 5.36: Velocity tracking performance over the complete flight at Aerospace Garden

5.5 Onboard System Performance

To evaluate the computational viability of the integrated navigation stack (VINS-Fusion and EGO-Planner) on the onboard Raspberry Pi 4B companion computer, system resource utilization was continuously logged across all real-world flight tests. The evaluation focused on CPU load and core thermal performance to ensure the system maintained real-time processing capabilities without experiencing thermal throttling or control loop latency.

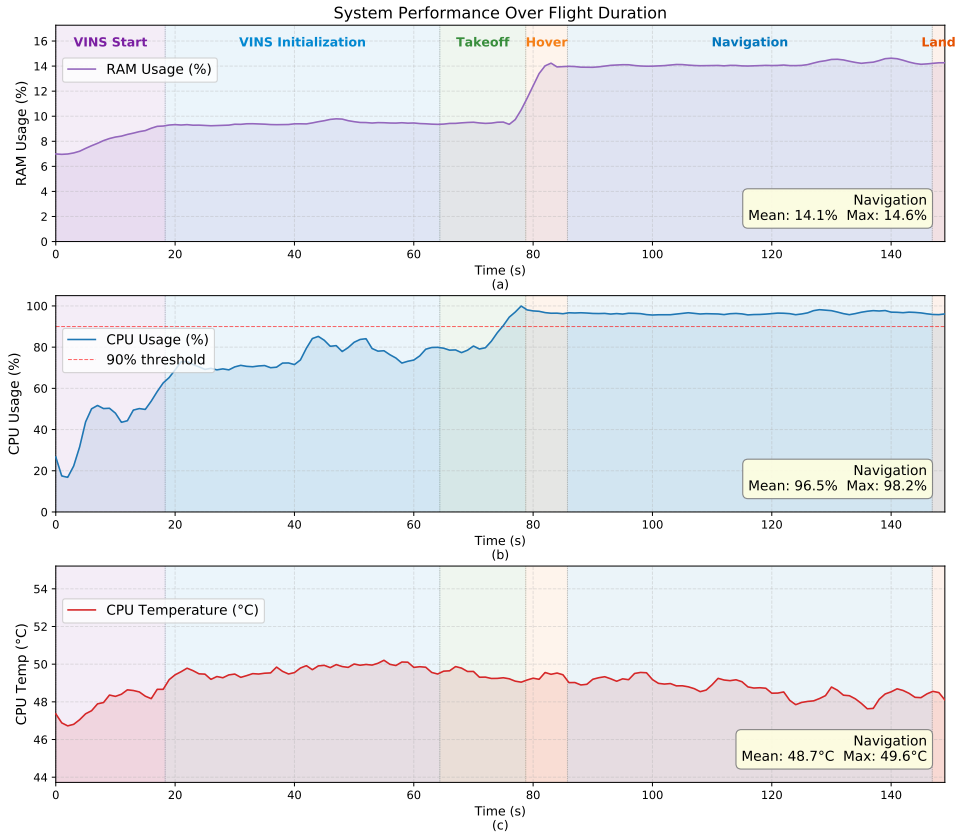


Figure 5.37: System health over the whole flight duration for Triangle with obstacle

Figure 5.37 presents the time-series profile of the system's health metrics during the active avoidance triangular mission at the Aerospace Garden. The CPU load exhibits distinct transient spikes that directly correspond to dynamic replanning events specifically, the moments when unmapped obstacles enter the depth sensor's field of view, forcing the EGO-Planner to rapidly compute new collision-free trajectories.

Table 5.1: Onboard Computational Load During Navigation Tests

Test Environment	CPU Mean (%)	CPU Max (%)	Temp Mean (°C)	Temp Max (°C)
Test with no obstacles	89.7	95.3	46.0	47.2
Test with obstacles	90.2	99.2	42.7	46.3
Incubation Center (IIEC)	91.9	99.7	47.0	49.2
Manufacturing Lab	92.7	99.5	47.0	48.2
Outdoor Flight - Aerospace Garden	88.9	99.0	40.7	42.4
Average	90.8	99.4	45.2	46.8

Table 5.1 aggregates the computational and thermal load across all tested environments. The overall mean CPU utilization stood at 90.8%, with transient maximums frequently reaching 99.4%. The highest sustained computational demands were recorded during the indoor tests at the Manufacturing Lab (92.7% mean) and the IIEC (91.9% mean, 99.7% max). This elevated resource consumption directly correlates with the dense, unstructured nature of these indoor environments, which necessitated continuous point-cloud processing, obstacle map inflation, and frequent trajectory optimization.

5.6 Summary of Navigation Accuracy

The tracking errors for all hardware flight tests were computed using the post-flight data processing procedure described in Section 4.3. Results are summarized in Table 5.2.

Table 5.2: Summary of Navigation Tracking Performance Across All Tests

Test	Mean X (m)	Mean Y (m)	Mean Z (m)	Mean 3D (m)	σ 3D (m)	RMSE 3D (m)
Closed-loop Simulation	0.026	0.100	0.190	—	—	—
Complete Autonomous Simulation	0.364	0.162	0.012	0.3874	0.0910	0.3979
Test with no obstacles	0.165	0.079	0.024	0.234	0.108	0.290
Test with obstacles	0.160	0.102	0.058	0.285	0.123	0.289
Incubation Center (IIEC)	0.086	0.125	0.031	0.308	0.137	0.268
Manufacturing Lab	0.132	0.130	0.049	0.277	0.127	0.583
Outdoor Flight - Aerospace Garden	0.123	0.137	0.042	0.275	0.140	0.352
Mean (Real flights)	0.136	0.109	0.040	0.276	—	0.358

5.7 DISCUSSION

5.7.1 VIO Performance Analysis

The VINS-Fusion state estimator demonstrated a clear sensitivity to IMU-camera extrinsic calibration quality, which was the most critical tuning step in the hardware deployment process. Prior to calibration, the closed-loop drift over a full manual flight reached $\Delta Y = -0.993$ m, rendering the estimator unsuitable for autonomous control. Following careful extrinsic calibration, the overall drift was reduced by 81.3%, with

post-calibration values of $\Delta X = -0.059$ m, $\Delta Y = -0.081$ m, and $\Delta Z = +0.177$ m. This result highlights that for tightly-coupled VIO systems, the accuracy of the physical sensor mounting and its software representation has a greater impact on estimator performance than algorithmic parameter tuning .

The subsequent hover test confirmed that the calibrated estimator was suitable for closed-loop control. The VINS Z channel tracked the 1.0 m hover command with an RMSE of 0.037 m and a maximum deviation of 0.099 m throughout the hover phase, while the independent VL53L1X distance sensor corroborated the altitude estimate. The higher offset recorded by the distance sensor (mean 0.115 m) is attributed to its sensitivity to floor surface reflectance and texture variation rather than any deficiency in the VIO pipeline.

In simulation, the closed-loop square mission produced drift values of $\Delta X = -0.026$ m, $\Delta Y = 0.100$ m, and $\Delta Z = 0.190$ m. The larger Z drift in simulation compared to hardware is consistent with the known limitation of synthetic environments, where repetitive floor textures and limited vertical visual features reduce the number of trackable features available to the estimator in the vertical direction. On real hardware, the richer and more varied visual texture of physical environments provided more reliable feature tracking, resulting in better Z-axis consistency across all real flight tests.

The outdoor test at the Aerospace Garden provided an additional validation of VIO robustness under conditions not present in any indoor test. Wind gusts introduced unmodeled aerodynamic disturbances that caused the quadcopter to deviate from its planned trajectory, forcing the estimator to track rapid attitude changes while maintaining position consistency. Varying sunlight and shadows also affected feature visibility across different segments of the 6-waypoint mission. Despite these factors, the VINS-Fusion estimator maintained a mean Z error of just 0.042 m outdoors, comparable to the 0.031–0.049 m range observed indoors. This confirms that the stereo infrared front-end of the D435 is not critically dependent on controlled indoor lighting for stable feature tracking, at least under the moderate outdoor conditions encountered during testing.

5.7.2 Trajectory Tracking and Obstacle Avoidance Performance

Table 5.2 summarizes the navigation tracking performance across all simulation and hardware flight tests. Across all real hardware tests, the integrated navigation stack demonstrated consistent trajectory tracking performance, with a mean 3D tracking error of 0.276 m and an average RMSE 3D of 0.358 m, both within the project’s target threshold of 0.3 m for mean error.

The per-axis analysis reveals a consistent trend across all environments: the Z-axis

consistently achieved the lowest tracking error, with a mean value of 0.040 m across real flights. This is attributable to the complementary fusion of the VINS Z estimate with the VL53L1X distance sensor within the PX4 EKF2, which provides an additional altitude anchor unavailable for the horizontal axes. The Y-axis consistently exhibited slightly higher errors than X across most tests, which can be attributed to lateral motion being more susceptible to VIO drift as the quadcopter moves primarily along the X-axis during straight segments.

The Aerospace Garden obstacle avoidance tests demonstrated that the EGO-Planner successfully replanned around manually introduced obstacles in real time without any collisions. Comparing the baseline and obstacle runs, the introduction of obstacles caused only a marginal increase in tracking error mean 3D increased from 0.234 m to 0.285 m and RMSE 3D remained virtually unchanged at 0.290 m and 0.289 m respectively. This indicates that the replanning deviations were smooth and did not significantly destabilize the controller, confirming the effectiveness of the ESDF-free B-spline optimization in generating dynamically feasible avoidance trajectories in real time.

The indoor tests at IIEC and the Manufacturing Laboratory presented a more challenging environment for the navigation stack due to higher obstacle density, narrower navigable corridors, and more varied lighting conditions. The IIEC test produced a mean 3D error of 0.308 m and RMSE 3D of 0.268 m, while the Manufacturing Laboratory produced a mean 3D error of 0.277 m. The Manufacturing Laboratory RMSE 3D of 0.583 m is inflated by a known Y-axis coordinate frame offset between the EGO-Planner and VINS-Fusion local frames during that specific flight, and the X and Z axis errors remain consistent with other test environments.

Comparing simulation and real hardware performance, the complete autonomous simulation in the Gazebo environment produced per-axis mean errors of $\Delta X = 0.364$ m, $\Delta Y = 0.162$ m, and $\Delta Z = 0.012$ m. The real hardware tests showed lower X and Y errors on average, which is somewhat counterintuitive but explainable. The simulated cluttered environment was geometrically more constrained than the real indoor spaces tested, forcing more aggressive replanning maneuvers and larger deviations from the optimal path. The Z-axis error in simulation (0.012 m) was lower than hardware (0.040 m mean) due to the idealized flat floor in the simulated world providing consistent altitude reference features.

The outdoor flight at the Aerospace Garden recorded a mean 3D tracking error of 0.275 m and an RMSE 3D of 0.352 m over a 6-waypoint rectangular mission covering $8\text{ m} \times 2\text{ m}$. These numbers are consistent with the indoor results and fell within the 0.3 m mean error target. The per-axis breakdown (X: 0.123 m, Y: 0.137 m, Z: 0.042 m) shows a slightly more balanced distribution between X and Y compared to the indoor

tests, where one horizontal axis typically dominated. This is likely because the rectangular mission profile required roughly equal travel in both horizontal directions, distributing drift more evenly. Wind disturbances were the primary source of transient error spikes, but the geometric controller recovered quickly and the overall standard deviation (0.140 m) remained comparable to indoor flights. The successful completion of this outdoor mission without GPS confirms that the navigation stack is not limited to indoor settings and can handle moderate environmental disturbances while maintaining acceptable tracking accuracy.

5.7.3 Computational Performance

Table 5.1 summarizes the onboard computational load recorded during each flight test on the Raspberry Pi 4 Model B companion computer. Across all tests, CPU utilization consistently reached 88-93% mean load during the navigation phase, with peak values touching 99% in all indoor tests. Despite operating at near-maximum CPU capacity, the system maintained stable autonomous flight throughout all tests, validating the computational efficiency of the chosen algorithm stack particularly the ESDF-free approach of EGO-Planner and the modular threading architecture of VINS-Fusion which keeps the core odometry thread real-time capable even under high load.

CPU temperature stabilized between 40-49°C across all tests, well within the thermal limits of the Raspberry Pi 4 hardware. The lower temperatures recorded during the Aerospace Garden outdoor test (mean 40.7°C) compared to indoor tests (mean 47°C) are consistent with the better ambient ventilation available in the open outdoor environment. RAM utilization remained low throughout all tests, confirming that memory was not a bottleneck for the navigation stack on this hardware configuration.

However, operating at near-maximum CPU load introduces a meaningful risk. Any transient computational spike caused by a sudden increase in obstacle density, a large loop closure event in VINS-Fusion, or system interrupts could cause the VIO thread to miss frames or the planner to delay trajectory updates, potentially destabilizing the controller. The absence of GPU acceleration on the Raspberry Pi 4 means all image processing, feature tracking, and optimization runs entirely on the CPU, leaving no headroom for additional processing tasks such as dense mapping or dynamic obstacle detection.

5.7.4 Limitations

Several limitations of the current system were identified during hardware testing and are important to acknowledge for an honest characterization of the platform's capabilities.

The forward-facing RealSense D435 camera provides an approximately 87° horizontal field of view, creating blind spots to the sides and rear of the quadcopter. Obstacles entering from outside this field of view are not detected until the quadcopter has already partially passed them, which limits the system's ability to react to lateral threats. Additionally, the absence of an upward-facing sensor creates a ceiling blind spot, making the quadcopter vulnerable to overhead obstacles and low ceilings in confined indoor spaces.

The VIO performance degrades in low-texture and low-light environments as it requires sufficient visual features to maintain stable tracking. This was observed in sections of the IIEC hall where large featureless concrete walls reduced the available feature count, causing brief increases in the tracking error.

EGO-Planner's occupancy grid assumes a fully static environment. Moving obstacles such as people walking through the flight space can cause incorrect and persistent map updates, potentially blocking navigable corridors. The ring-buffer voxel map also has a fixed spatial extent, meaning obstacles outside the configured map boundary are entirely ignored by the planner until the quadcopter moves close enough to bring them within the mapping range.

A central goal of this project was to implement VINS and Ego-Planner on a low-power onboard computer. Achieving this resulted in a critical hardware bottleneck. The Raspberry Pi 4 Model B operates at an 88-99% CPU utilization rate maintaining baseline navigation. Because there is virtually no remaining processing capacity, we are restricted from adding extra features or extending the algorithms for greater robustness.

Finally, the mean current draw of approximately 50 A under full flight load limits the practical flight duration to approximately 5 minutes on the 3S-4000 mAh LiPo battery. This constrains the complexity and range of missions executed in a single flight.

5.8 Problems Faced

The development and integration of the hybrid visual-inertial navigation system presented several significant engineering challenges. The team had to mitigate software stability issues within the physics simulator, as well as overcome severe hardware malfunctions during real-world deployment, including catastrophic electrical failures and structural damage. These challenges are categorized below.

5.8.1 Challenges Encountered During Simulation

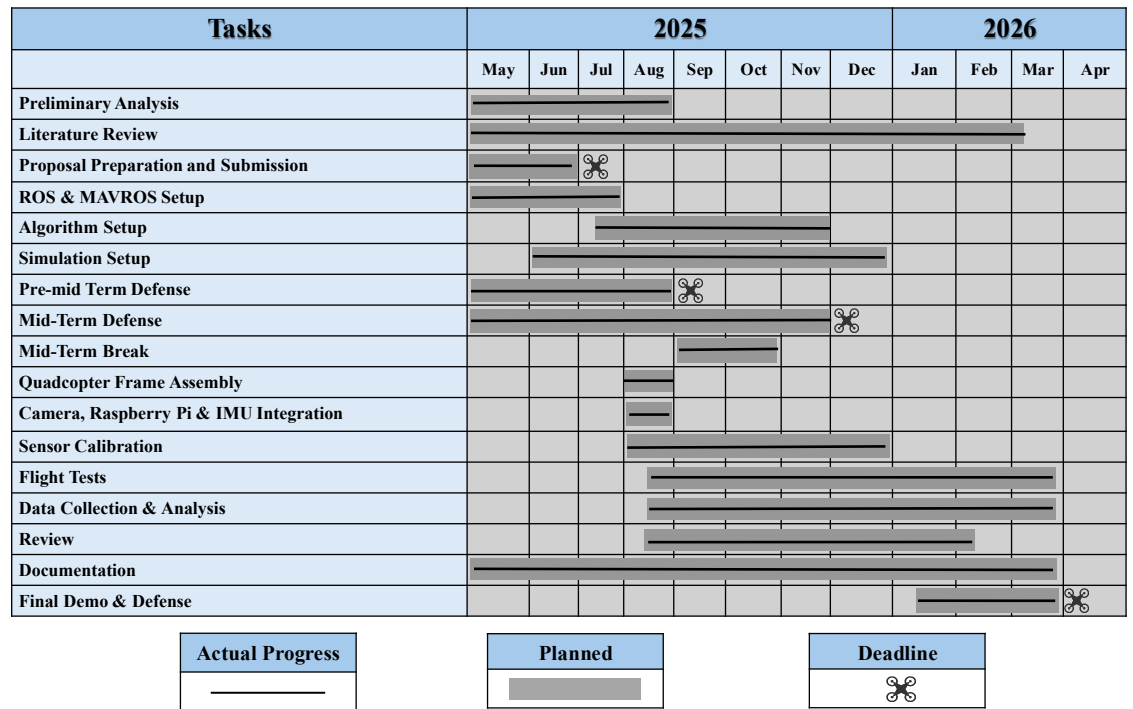
- i. **Extrinsic Calibration and Coordinate Frame Mismatch:** Initial testing revealed immediate VIO divergence due to asynchronous reference frames between the depth camera's optical frame and the IMU's body frame.
- ii. **Feature Starvation in Simulated Environments:** Simulated environments in Gazebo often lack the complex, high-contrast visual textures present in the real world. Hovering near plain, uniform walls caused the visual feature tracker to drop near zero, leading to severe state estimation drift and crashes.
- iii. **Message Dropping and Time Synchronization:** Unsynchronized time between ROS and Gazebo simulator caused dropping messages, forcing the VIO system to discard synchronized camera frames to prevent estimation failure.
- iv. **VIO Preintegration Instability:** Instantaneous trajectory generation by the EGO-Planner occasionally commanded physically unrealistic accelerations. This induced extreme simulated IMU spikes, causing mathematical instability (NaN errors) in the VINS-Fusion preintegration step and forcing the solver to abort.

5.8.2 Challenges Encountered in Hardware Implementation

- i. **Mechanical Resonance and IMU Noise:** High-frequency vibrations from the brushless motors introduced significant noise into the Pixhawk 4's accelerometer data, leading to inconsistent readings and Extended Kalman Filter (EKF) divergence.
- ii. **Catastrophic Sensor and Compute Failure:** A critical voltage surge caused permanent damage to the quadcopter's avionics system. This resulted in the simultaneous burnout of the Raspberry Pi 4 companion computer and Pixhawk Flight Controller necessitating a complete hardware replacement.
- iii. **Airframe Structural Failure:** A collision during preliminary physical testing resulted in the fracture of one quadcopter arm. This compromised the structural integrity of the airframe, requiring replacement parts and recalibration of the vehicle's center of mass and moment of inertia.
- iv. **Serial Communication Latency:** Inconsistencies in the baud rate and message buffers between the Pixhawk 4 and the Raspberry Pi led to delayed state feedback. This latency caused oscillatory behavior during high-speed autonomous maneuvers, which was resolved by optimizing the MAVROS message rates.

5.9 Work Timeline

Table 5.3: Project Timeline



5.10 Budget Estimation

Table 5.4 shows proposed budget that includes expenses for both new and existing resources.

Table 5.4: Budget Estimation

S.N.	Description	Quantity	Estimated Cost (NPR)
1	X360 Drone Frame	1	6,000
2	EMax ECO II Motor	4	10,000
3	3s LiPo Battery	1	6,000
4	Raspberry Pi 4B	1	30,000
5	20A ESC	4	4,000
6	PixHawk 4	1	40,000
7	Intel RealSense D435 Camera	1	45,000
8	SSD 256 GB	1	5,000
9	5A Buck Converter	2	500
10	VL53L1x - TOF	1	1,000
11	Miscellaneous	-	20,000
Total			1,67,500

Chapter 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1 Conclusion

This report presented the design, implementation, and experimental validation of a hybrid visual-inertial navigation system for autonomous, collision-free flight in GPS-denied indoor environments using a custom quadcopter. The complete navigation stack, comprising VINS-Fusion for state estimation, EGO-Planner for real-time trajectory generation and obstacle avoidance, and a non-linear geometric controller, was successfully deployed on a Raspberry Pi 4 Model B companion computer achieving a mean 3D tracking error of 0.276 m and an average RMSE 3D of 0.358 m across all real flight environments, with the system operating at 88%-99% CPU utilization. These results validate the proposed architecture as a viable solution for autonomous GPS-denied indoor navigation on commodity embedded hardware, demonstrating reliable obstacle avoidance and goal directed flight across multiple real-world indoor and outdoor environments while remaining within the project's target accuracy threshold of 0.3 m mean error.

6.2 Future Work

Several directions are identified for extending the capabilities of the current system:

i. **Mapping Capabilities.**

Dense 3D Reconstruction of the surrounding environment from raw depth images and EGO-Planner grid map.

ii. **Active Multi-Sensor Fusion.**

Integrating a rotating 2D LiDAR alongside a tactical-grade IMU would provide full 360° obstacle awareness, eliminating the current forward-facing field of view blind spot and significantly reducing localization drift over longer missions.

iii. **GPU-Enabled Companion Computer.**

Replacing the Raspberry Pi 4 with an NVIDIA Jetson Orin Nano or Xavier NX would provide GPU acceleration, reducing CPU load from approximately 95% to 40–50% and enabling higher flight speeds without risking control loop latency.

iv. **Dynamic Obstacle Avoidance.**

Extending EGO-Planner beyond its current static environment assumption by incorporating lightweight real-time object detection would enable the system to handle moving obstacles for practical deployment in real indoor environments.

REFERENCES

- [1] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, “LVI-SAM: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.
- [2] D. Dardari, P. Closas, and P. M. Djurić, “Indoor tracking: Theory, methods, and technologies,” *IEEE Transactions on Vehicular Technology*, vol. 64, no. 4, 2015.
- [3] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [4] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [5] A. I. Mourikis and S. I. Roumeliotis, “A multi-state constraint kalman filter for vision-aided inertial navigation,” in *2007 IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy, 2007.
- [6] J. Roberts, F. Gonzalez, F. Vanegas, and K. Gaston, “A framework for uav navigation and exploration in gps-denied environments,” Queensland University of Technology, Brisbane, QLD, Australia, Tech. Rep., 2018.
- [7] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, and Y. Mulgaonkar, “Robust stereo visual inertial odometry for fast autonomous flight,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3025–3032.
- [8] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, “Robust visual inertial odometry using a direct ekf-based approach,” in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 298–304.
- [9] T. Qin, P. Li, and S. Shen, “Vins-mono: A robust and versatile monocular visual-inertial state estimator,” *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1094–1107, 2018.
- [10] T. Qin, J. Pan, S. Cao, and S. Shen, “A general optimization-based framework for local odometry estimation with multiple sensors,” *arXiv preprint arXiv:1901.03638*, 2019.
- [11] X. Kong, W. Wu, L. Zhang, and Y. Wang, “Tightly-coupled stereo visual-inertial navigation using point and line features,” *Sensors*, vol. 15, 2015, Publisher: MDPI.

- [12] M. Jaimez and J. Gonzalez-Jimenez, “Fast visual odometry for 3-d range sensors,” *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 883–892, Aug. 2015.
- [13] C. Forster, M. Pizzoli, and D. Scaramuzza, “Svo: Fast semi-direct monocular visual odometry,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 15–22.
- [14] A. James, A. Seth, E. Kuantama, S. Mukhopadhyay, and R. Han, “Sensor fusion for autonomous indoor uav navigation in confined spaces,” in *2023 IEEE Sensors Applications Symposium (SAS)*, 2023, pp. 1–6.
- [15] S. Nur, “Precision without gps: Multi-sensor fusion for autonomous drone navigation in complex environments,” *International Journal of Innovative Research in Computer Science & Technology*, vol. 12, no. 6, pp. 34–43, 2024.
- [16] O. Seiskari, P. Rantalankila, J. Kannala, J. Ylilammi, E. Rahtu, and A. Solin, “Hybvio: Pushing the limits of real-time visual-inertial odometry,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3421–3428, 2022.
- [17] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, “Ego-planner: An esdf-free gradient-based local planner for quadrotors,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2021.
- [18] S. Liu, M. Watterson, S. Tang, and V. Kumar, “High speed navigation for quadrotors with limited onboard sensing,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2482–2489.
- [19] K. Mohta et al., “Fast, autonomous flight in GPS-denied and cluttered environments,” *Journal of Field Robotics*, vol. 35, no. 1, pp. 121–163, 2018.
- [20] T. Lee, M. Leok, and N. H. McClamroch, “Geometric tracking control of a quadrotor UAV on SE(3),” in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.
- [21] S. A. Kumar, Y. Bibek, J. L. Babu, and K. Samim, “Simulation and real time implementation of visual slam in an indoor environment using a quadcopter,” in *Proceedings of the Department of Mechanical and Aerospace Engineering Project Conference*, Tribhuvan University Institute of Engineering, Pulchowk Campus, Lalitpur, Nepal, Mar. 2025.
- [22] A. Bachrach, S. Prentice, R. He, and N. Roy, “Range - robust autonomous navigation in gps-denied environments,” *Journal of Field Robotics*, vol. 28, no. 5, pp. 863–891, Oct. 2011.
- [23] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, “Review of visual odometry: Types, approaches, challenges, and applications,” *SpringerPlus*, vol. 5, no. 1, p. 1897, 2016.

Appendix A

Python Code

PX4 Offboard Attitude Control Script

```
import asyncio
from mavsdk import System
from mavsdk.offboard import (OffboardError, Attitude)

HOVER_THRUST = 0.573
FLYING_THRUST = 0.573

TILT_ANGLE_DEG = 2.5
FLIGHT_TIME_SEC = 10.0
BRAKE_TIME_SEC = 1.0
BRAKE_ANGLE_DEG = 2.0

async def run():
    connection_string = "serial:///dev/ttyUSB0:57600"
    drone = System()
    await drone.connect(system_address=connection_string)

    async for state in drone.core.connection_state():
        if state.is_connected:
            break

    await asyncio.sleep(5)

    await drone.offboard.set_attitude(Attitude(0.0, 0.0, 0.0,
        HOVER_THRUST))

    try:
        await drone.offboard.start()
    except OffboardError:
        return

    await drone.offboard.set_attitude(Attitude(0.0, -TILT_ANGLE_DEG
        , 0.0, FLYING_THRUST))
    await asyncio.sleep(FLIGHT_TIME_SEC)

    await drone.offboard.set_attitude(Attitude(0.0, BRAKE_ANGLE_DEG
        , 0.0, HOVER_THRUST))
    await asyncio.sleep(BRAKE_TIME_SEC)

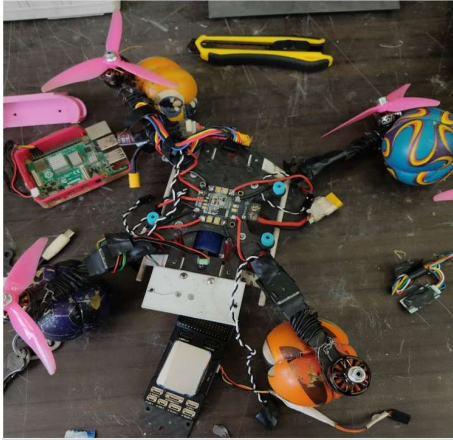
    await drone.offboard.set_attitude(Attitude(0.0, 0.0, 0.0,
        HOVER_THRUST))
    await asyncio.sleep(2)

    await drone.offboard.stop()

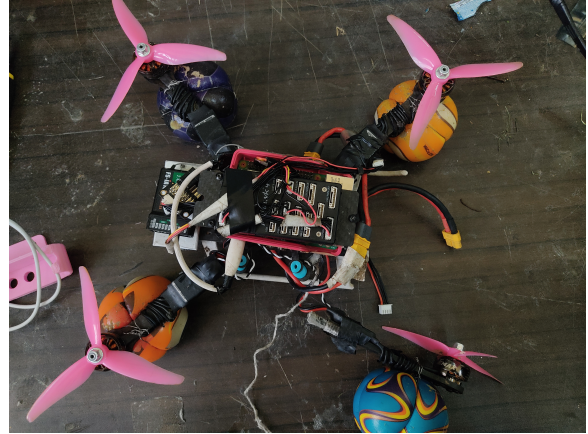
if __name__ == "__main__":
    asyncio.run(run())
```

Appendix B

Test Failures and Results



(a) First Crash



(b) Second Crash



Figure B.2: Qr code for google drive results