Autonomous aerial navigation in indoor and obscurant-filled environment

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science

by

Arvind L. Pandit
(Roll No. 211022001)

Under the guidance of
Dr. Ameer K. Mulla

॥ सा विद्या या विमुक्तये ॥
भारतीय प्रौद्योगिकी संस्थान धारवाड
Indian Institute of Technology Dharwad

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY DHARWAD
June, 2023

Acknowledgements

I would like to thank my supervisor Dr. Ameer K. Mulla for his constant support, invaluable assistance and insights which were crucial in developing this work. I would also like to thank Assistant Professor Dr. Sudheer Siddapureddy, Dr. Bharat B.N, and Dr. Apurva Joshi for their guidance and advice.

I would be amiss if I did not mention my lab mates, Rajat Joshi, Ajul Dinesh, Jeslin Jacob M, Lokesh Kumar and Chandan N, who were always ready to help me during the difficult times. I also like to express my gratitude to my hostel (Asavari) for their moral support throughout my masters journey. Last but not least, I want to express my sincere gratitude to my family for their unwavering support and the optimism they have given me. This thesis would not have been conceivable without such hope. I'm grateful to you all for giving me courage.

Arvind L. Pandit

Thesis Approval

The thesis entitled

Autonomous aerial navigation in indoor and obscurant-filled environment

by

Arvind L. Pandit
(Roll No. 211022001)

is approved for the degree of

Master of Science

_____	_____
Examiner	Examiner
_____	_____
Guide	Chairman

Date: _____

Place: _____

INDIAN INSTITUTE OF TECHNOLOGY DHARWAD, INDIA

CERTIFICATE OF COURSE WORK

This is to certify that **Arvind L. Pandit** (Roll No. 211022001) was admitted to the candidacy of Master of Science degree on 1st August 2021, after successfully completing all the courses required for the Master of Science programme. The details of the course work done are given below.

S.No	Course Code	Course Name	Credits
1	EE303	Linear Algebra and Its Applications	6
2	ME435	Design of Mechatronic Systems	6
3	–	Seminar	4
4	–	Communication Skills (PP/NP Course)	0
5	EE804	Stochastic Control and Learning for Networked Systems	6
6	EE438	Probability Models and Applications	6
7	CS420	Reinforcement Learning	6
8	CS201	Data Structures and Algorithms (Audit Couse)	0
		Total Credits	**34**

IIT Dharwad

Date: Asst. Registrar (Academic)

Declaration

I declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Arvind L. Pandit
Roll No. 211022001

Date: June, 2023

Abstract

Quadcopters have emerged as versatile platforms for various applications, showcasing their maneuverability and agility. While extensive research has been conducted on autonomous navigation in outdoor environments, the focus on indoor GPS-denied scenarios remains limited. This thesis addresses the challenges of autonomous quadcopter navigation in indoor GPS-denied environments, with particular emphasis on two key objectives: the development of a visual pipeline for complete autonomous navigation and the creation of a thermal navigation system for obscurant-filled environments.

The primary objective of this study is to devise a comprehensive visual pipeline tailored to indoor GPS-denied environments using a minimal sensor suite. The pipeline encompasses critical modules, including camera calibration, pose estimation, 3D mapping, and planning, utilizing only a monocular camera and an IMU(Inertial Measurement Unit). By leveraging the visual inertial navigation approach with this minimal sensor suite, the system can robustly navigate and traverse the indoor environment, even in the absence of GPS signals. This approach not only simplifies the system architecture but also addresses the constraint of limited payload capacity for quadcopters, as additional sensors such as LiDAR, stereo and depth camera can be heavy and expensive.

Furthermore, recognizing the significance of operating in obscurant-filled environments, such as structure fires or smoke-laden areas, a thermal navigation system is developed. This system capitalizes on the advantages of thermal cameras, specifically their immunity to smoke, soot, and low visibility conditions. By integrating thermal camera inputs and the visual inertial odometry technique, accurate pose estimation is achieved, facilitating reliable navigation in these challenging environments. The utilization of a thermal camera within the minimal sensor suite reinforces the system's ability to operate effectively in obscurant-filled environments while considering the constraints of weight and cost.

This research work successfully develops a visual pipeline for autonomous navigation in

indoor GPS-denied environments utilizing a single monocular camera and an IMU. The integration of visual inertial odometry, 3D mapping, and planning techniques demonstrates the system's effectiveness and robustness, while the inclusion of a thermal camera within the minimal sensor suite enhances the system's performance in obscurant-filled environments. The proposed approach strikes a balance between functionality and resource constraints, enabling cost-effective and efficient autonomous navigation in indoor and challenging scenarios.

Contents

List of Figures

Chapter 1

Introduction

Autonomous aerial vehicles, such as quadcopters, offer cost-effective and highly mobile benefits, making them ideal robotics platforms for surveillance and exploration purposes. Due to their fast dynamics and the fact that they are often used out of sight during exploration tasks, autonomous navigation is crucial for quadcopters. To achieve autonomous navigation, reliable pose estimation, dense 3D reconstruction, path planning, and obstacle avoidance are essential modules. While the GPS system is a well-established positioning system, it has limitations, particularly the lack of obstacle information and its inability to function in indoor environments like buildings, mines, and cluttered spaces. Therefore, in confined environments, it is necessary for quadcopters to be small in size, which introduces Size, Weight, and Power (SWaP) constraints, restricting the ability to carry heavy sensors like LiDAR or stereo setups. Consequently, extensive work has been conducted on autonomous aerial navigation in confined and GPS-denied environments using a Visual-Inertial Navigation System (VINS). A monocular VINS consists of a monocular camera fused with an IMU for the quadcopter's pose estimation.

This thesis aims to address the problem of indoor navigation for quadcopters in GPS-denied environments while considering SWaP constraints. The primary objective is to develop a system that utilizes a minimal sensor suite for the complete navigation pipeline. The system only employs a monocular camera and an device called IMU for tasks such as localization, mapping, and path planning, all with onboard computation. The proposed system pipeline is divided into five modules: camera modeling and calibration, Visual-Inertial Odometry (VIO), 3D dense mapping, path planning, and obstacle avoidance, which will be further discussed in subsequent sections.

The second part of this work focuses on visually degraded environments, including struc-

ture fires, mines, and fog. Specifically, the use case of structure fires is considered, with the main objective being the development of a quadcopter capable of autonomously navigating within a structure fire, detecting human victims using a minimal sensor suite comprising a monocular thermal camera and an IMU. The detected victim locations are then communicated to the base station, facilitating the rescue process by providing firefighters and rescue teams with the victims' known locations, ultimately saving lives. To solve the localization problem, an approach similar to Visual-Inertial Localization is employed, but instead of using a visual camera, an LWIR (Long Wave Infrared) Camera is used for feature detection and tracking, as it is immune to issues such as illumination and scattering by suspended particles.

Further the thesis is organized as follows. Chapter 1 provides a basic introduction to the system, explaining the system architecture and discussing the work done. Chapter 2 covers the necessary preliminaries required for better understanding, including basic camera modeling, camera calibration, and measurement preprocessing. Chapter 3 presents the formulation of VINS-mono [2], a state-of-the-art Visual-Inertial Odometry Architecture, and also includes experimental results for the same. Chapter 4 explains the process of 3D dense reconstruction using a monocular camera. The 3D motion planning for the quadcopter is discussed in Chapter 5, which concludes the visual pipeline of the proposed system. Chapter 6 showcases the performance of Visual Odometry using thermal images. Chapter 7 focuses on the development of the quadcopter for fire scenarios, followed by the conclusion and future work in Chapter 8. Additionally, Appendix A presents our work on MultiAgent Systems, which can be incorporated to enhance the overall efficiency of the system.

1.1 Literature Review

Quadcopters, widely used in various applications such as infrastructure inspection [7–9], precision agriculture [10,11], and search and rescue [12,13], have proven to be versatile and effective tools. Recent advancements in unmanned aerial vehicles (UAVs) have showcased their intelligence and capability to navigate in GPS-denied environments [14].

Stereo vision methods, commonly employed for indoor navigation, have been extensively explored. In [15], real-time stereo matching was implemented using three onboard computers, coupled with a probabilistic voxel map and an Octomap-based path planner to generate obstacle-free paths to designated waypoints [16]. Another study [17] proposed a vision-based

mapping framework in which the mapping module was operated offline, and loop closure was implemented separately.

While stereo or depth cameras are prevalent in these works, our focus lies in utilizing a monocular camera and a low-cost IMU for navigation. Several monocular camera-based navigation approaches have been proposed. For example, in [18], a flight platform combining SVO (Semi-direct Visual Odometry) [19] and REMODE (REgularized MOnocular Depth Estimation) [20] was designed for autonomous flight and dense 3D reconstruction. However, the system introduced high delays due to its offline mapping module. Another study [21] achieved autonomous flight but lacked mapping and obstacle avoidance capabilities.

In visually degraded and obscurant-filled environments, traditional vision and depth sensors may provide distorted information. Evaluations of navigation sensors in robotics [6] revealed that LWIR (Long-Wave Infrared) cameras and radars are reliable in such conditions. LWIR cameras, particularly useful in obscurant-filled environments, have been employed for aerial navigation. For instance, in [22]pose estimation was implemented on a quadcopter equipped with an LWIR camera (FLIR Tua 2) to track trajectories in fog-filled environments. Additionally, an extended VINS-mono (Visual-Inertial Odometry) [23] was implemented for an Advanced Driver Assistance System (ADAS).

Our proposed system focuses on autonomous aerial navigation of a quadcopter in an indoor GPS denied environment which uses a minimal sensor suite of monocular camera and an IMU. Work is also done for the Thermal navigation for an obscurant-filled environment, utilizing LWIR camera. This approach aims to address the challenges posed by visual degradation and obscurants while providing accurate and reliable navigation in such conditions.

1.2 Problem Definition

In the field of indoor navigation for drones, the existing technology heavily relies on stereo cameras and depth sensors. However, these sensors pose significant constraints in consideration of size, weight, and power consumption, which limit their practicality for drones.

To address this problem, this research proposes a approach that utilizes a monocular camera and IMU for a complete navigation pipeline in indoor environments. By leveraging a monocular camera and IMU, the system aims to overcome the limitations of traditional stereo and depth sensors while maintaining the necessary accuracy and efficiency for autonomous

navigation.

Furthermore, this research also focuses on developing a system specifically designed for navigation in obscurant-filled environments. By integrating a monocular thermal camera with the IMU, the system aims to provide reliable navigation even in conditions with visual degradation and low visibility caused by smoke, fog, or other obscurants.

The relevance of this problem lies in the increasing demand for autonomous drone navigation in indoor and challenging environments. By addressing the constraints of size, weight, and power, and enhancing navigation capabilities in obscurant-filled environments, the proposed research will contribute to the advancement of drone technology for various applications, including infrastructure inspection, exploration, precision agriculture, and search and rescue operations.

The objectives of this research are to:

- Develop a comprehensive navigation pipeline using a monocular camera and IMU for accurate and efficient autonomous navigation in indoor environments.
- Investigate the integration of a monocular thermal camera and IMU to enable reliable navigation in obscurant-filled environments.
- Evaluate the performance and effectiveness of the proposed navigation systems through experimental validation and comparative analysis.

By achieving these objectives, this research aims to provide valuable insights and advancements in the field of indoor drone navigation, addressing the limitations of existing technology and paving the way for more practical and versatile drone applications.

1.3 Complete Pipeline

This section explains the full system pipeline and the working of each module.

1.3.1 System Architecture

Figure 1.1 shows the full system architecture of the autonomous aerial navigation system for an autonomous quadcopter in visually degraded environments. It also shows the framework which is to be used for the specific task. The full pipeline is divided into four major modules, a detailed explanation of the working of each module is provided in the latter part of this section.

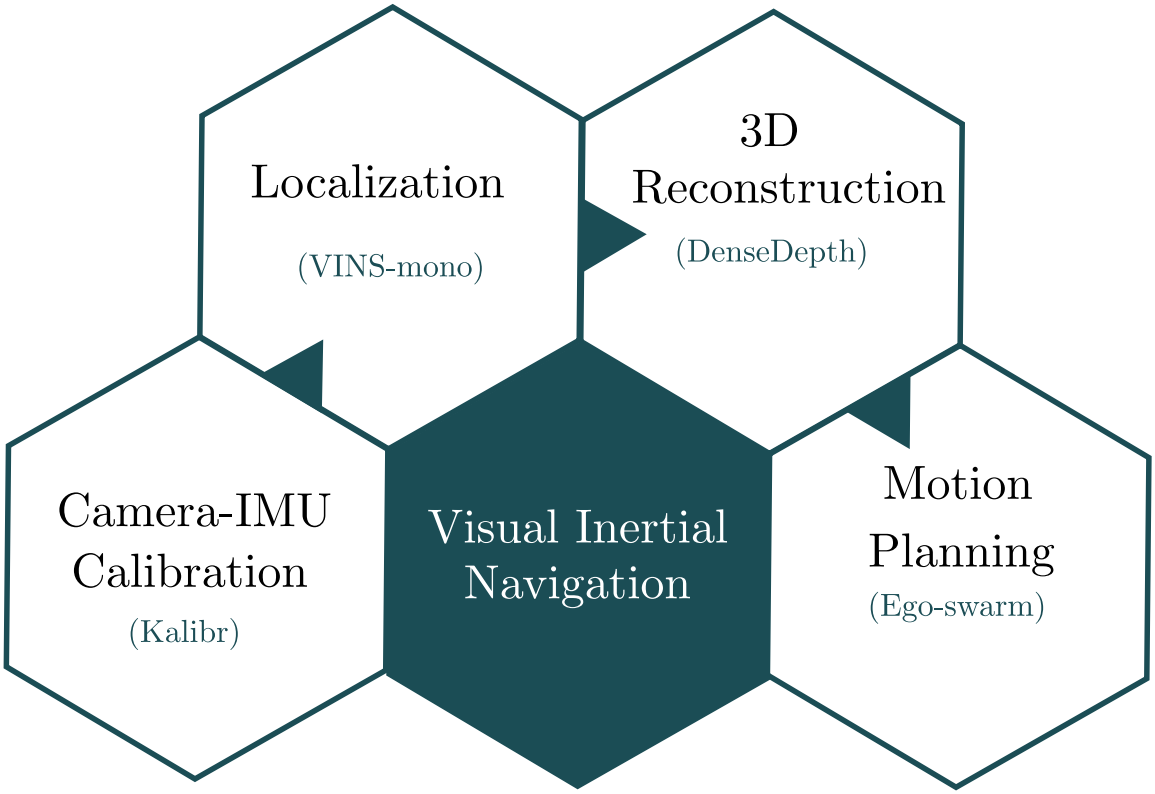


Figure 1.1: System Pipeline

1. Camera Calibration

The camera calibration process is an important step for any computer-vision algorithm. Due to the manufacturing defects and imperfect lenses, the image formed by the camera is distorted. The process of finding this distortion and intrinsic parameters of a camera is called camera calibration. As our system also uses the IMU for the pose estimation the extrinsic parameters such as transformation between camera frame and IMU frame are also calculated in this step. Kalibr framework is used for this process [24, 25]. For calibrating the thermal camera a custom calibration board is developed.

2. Visual-Inertial Localization

Localization is a process of estimating our own position in a 3D environment using onboard sensors. In this step, the pose of the Quadcopter is estimated using a camera and an IMU. The VIO(Visual Inertial Odometry) system uses a visual camera for the feature detection and fusing it with an IMU estimates the 6DOF pose of the quadcopter moving in 3D space.

3. Dense Mapping and 3D Reconstruction

After the pose estimation, in this step, the depth of each pixel in an image is estimated, and a 3D point cloud is generated. Using the depth estimation from these images and poses obtained through VIO, the depth image is fused into a global 3D map. This fusion process results in the generation of a dense map of the environment, which can be utilized for path planning and obstacle avoidance.

4. Autonomous Exploration and Obstacle Avoidance

This module deals with path planning in a 3D map and avoiding obstacles.Here we have used EGO-Planner [5] which is an ESDF free based local planner for the path planning avoiding obstacles. Finally, the control of the drone will be handled by a low-level flight controller.

1.4 MASCOT

During the exploration of navigation techniques, an extensive range of control algorithms designed for both single and multi-agent systems was encountered. It became evident that researchers in this field required a dedicated platform to deploy and evaluate their algorithms effectively. To address this need, the development of the MultiAgent Simulator (MASCOT) was undertaken. MASCOT serves as a comprehensive simulation environment that enables researchers to test and refine their control algorithms, taking into account the real-world dynamics of multi-agent systems.

It is important to be noted that while the development of MASCOT is an integral part of the research journey, it is not directly related to the main focus of the study. Therefore, the details and technical aspects of MASCOT are presented in Appendix A. This appendix provides a comprehensive overview of the simulator's architecture, functionality, and features. By providing researchers with a reliable and flexible platform, MASCOT enhances the experimental capabilities and supports the exploration of various control algorithms in a simulated environment.

Chapter 2

Preliminaries

This chapter provides essential background knowledge for understanding the subsequent chapters of this thesis. It covers key concepts such as camera modeling, calibration, structure from motion, quaternion representation of orientation, camera and IMU residuals, and IMU preintegration. These topics form the foundational elements of the visual and inertial navigation systems employed in the autonomous quadcopter navigation framework developed in this study. Understanding these concepts is crucial for accurately estimating camera parameters, generating 3D maps, and fusing sensor data for precise pose estimation. This chapter establishes the necessary groundwork for the subsequent chapters' exploration of autonomous quadcopter navigation in indoor GPS-denied environments and obscurant-filled scenarios.

2.1 Modeling and Calibration

2.1.1 Camera Model

A camera projects a 3D point onto a 2D image plane, and this process can be described by a geometric model. Several models exist to explain this phenomenon, but in this case, we will utilize the simplest one known as the pinhole model. Figure 2.1 illustrates the pinhole camera model. In this model, light from a real-world object passes through a small pinhole, resulting in an inverted image on the image plane situated at a distance f from the pinhole. To simplify the mathematical representation and eliminate negative signs, a virtual image plane is considered in front of the pinhole, where the image is formed in an upright position.

In coordinate system of camera, the origin of the camera is located at the pinhole, which

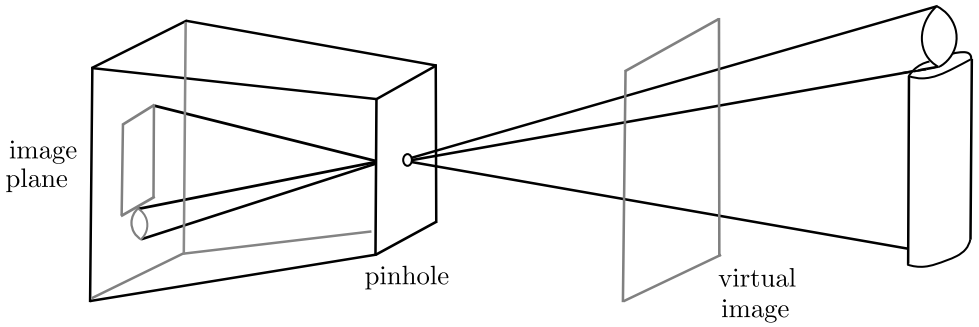


Figure 2.1: Representation of Pinhole Camera model

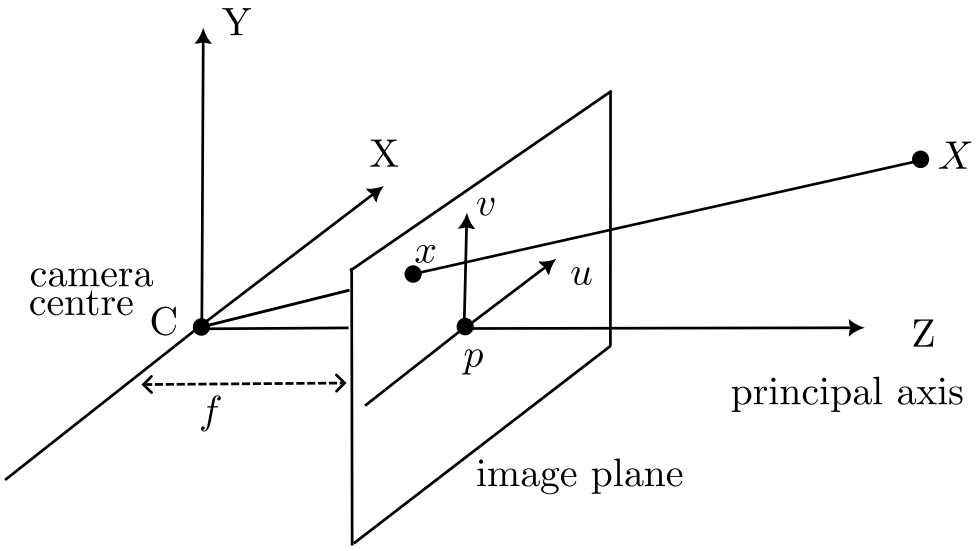


Figure 2.2: Camera Coordinate system and its similarity triangles

is known as camera center C. The three dimensional point $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ represents an arbitrary point in a camera coordinate system. Referring to Figure 2.2, the coordinates of the projection of point $\mathbf{X}$ onto the image plane are given by:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f\frac{X}{Z} \\ f\frac{Y}{Z} \end{bmatrix}$$

The mathematical model represented above is for the ideal camera, but in reality, if this kind of camera is used, the image formed will be too dim. If the size of the pinhole is increased, it will create a blurry image. The solution is to add a lens in place of the pinhole with a focal length f that may differ in the x and y directions ($\begin{bmatrix} f_x & f_y \end{bmatrix}$). Due to manufacturing defects, the focal length of the lens may vary.

In digital images, the origin $(0, 0)$ is usually located at the top-left corner of the image.

Therefore, we need to translate the image using a vector $\begin{bmatrix} c_x & c_y \end{bmatrix}$.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \end{bmatrix}$$

After converting to a homogenous system the camera model is given by

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} or \ \mathbf{p} = \mathbf{KX} \tag{2.1}$$

where $\mathbf{K}$ is called the camera intrinsic matrix, and f_x, f_y, c_x, and c_y are referred to as intrinsic parameters. The non-homogeneous coordinates in the image plane are given by $x = \frac{x'}{w'}$ and $y = \frac{y'}{w'}$.

The lenses used in the camera exhibit two types of distortion. Radial distortion occurs due to imperfections in the lens, causing the outer edges of the image to curve outward or inward. Tangential distortion occurs when the CMOS chip is not in perfectly parallel to the mounted lens.

The radial distortion model is given by:

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \begin{bmatrix} x \\ y \end{bmatrix}$$

The tangential distortion model is given by

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 2p_1 xy + p_2(r^2 + 2x^2) \\ 2p_2 xy + p_1(r^2) \end{bmatrix}$$

where $k_1, k_2, k_3, p_1 \ and \ p_2$ are distortion parameters.

2.1.2 Camera Calibration

The captured image does not represent the true perspective of the perfect camera; therefore, it requires transformation into the ideal camera image. To achieve this, we must utilize the intrinsic matrix $\mathbf{K}$ and the distortion parameters, which are essential for converting the raw image to the desired image. The primary objective of the camera calibration process is to determine these intrinsic and distortion parameters.

Here, we use Zhang's technique [26] for camera calibration using a chessboard. In this method, a chessboard with known side lengths for each square is used as the calibration object. The corner points of each square serve as features for calibration.

We define three coordinate frames:

1. The object coordinate frame.
2. The camera coordinate frame.
3. The image coordinate frame (2D image plane).

In the object coordinate frame, all points lie on the chessboard plane, implying that the z-coordinate is always 0. The corner points possess known coordinates, such as $(a, a, 0)$, $(a, 2a, 0)$, $(a, 3a, 0)$, etc., where a is the side length of the square. Figure 2.3 shows the chessboard with the detected corner points.

Now, to find the homography H, we need to capture the image of chessboard from a different angles using the camera we want to calibrate. Then, we identify the box corner points an image and establish a correspondence between each box corner point and the real-world object.

Equation 2.1 demonstrates that we can use the matrix K to transform coordinates from the camera coordinate system to the image coordinate system. The transformation of a point from the object coordinate system to the camera coordinate system can be expressed by the following equation:

$$P_{cam} = \Big[\mathbf{R}|\mathbf{t}\Big]\, \mathbf{P_{obj}} \tag{2.2}$$

where the rotation matrix is given by $\mathbf{R}$, $\mathbf{t}$ is a translation vector $\mathbf{P_{cam}}$ and $\mathbf{P_{obj}}$ is a homogeneous coordinate of a point in the camera and object frame respectively. From equation (2.1) and (2.2) the coordinate in the image frame is given by

$$\mathbf{p_{img}} = \mathbf{K}\Big[\mathbf{R}|\mathbf{t}\Big]\, \mathbf{P_{obj}} \tag{2.3}$$

The homographic projection is given by

$$\mathbf{p_{img}} = \lambda \mathbf{H} \mathbf{P_{obj}} \tag{2.4}$$

From (2.3) and (2.4)

$$\lambda \mathbf{H} = \mathbf{K}\Big[\mathbf{R}|\mathbf{t}\Big] \tag{2.5}$$

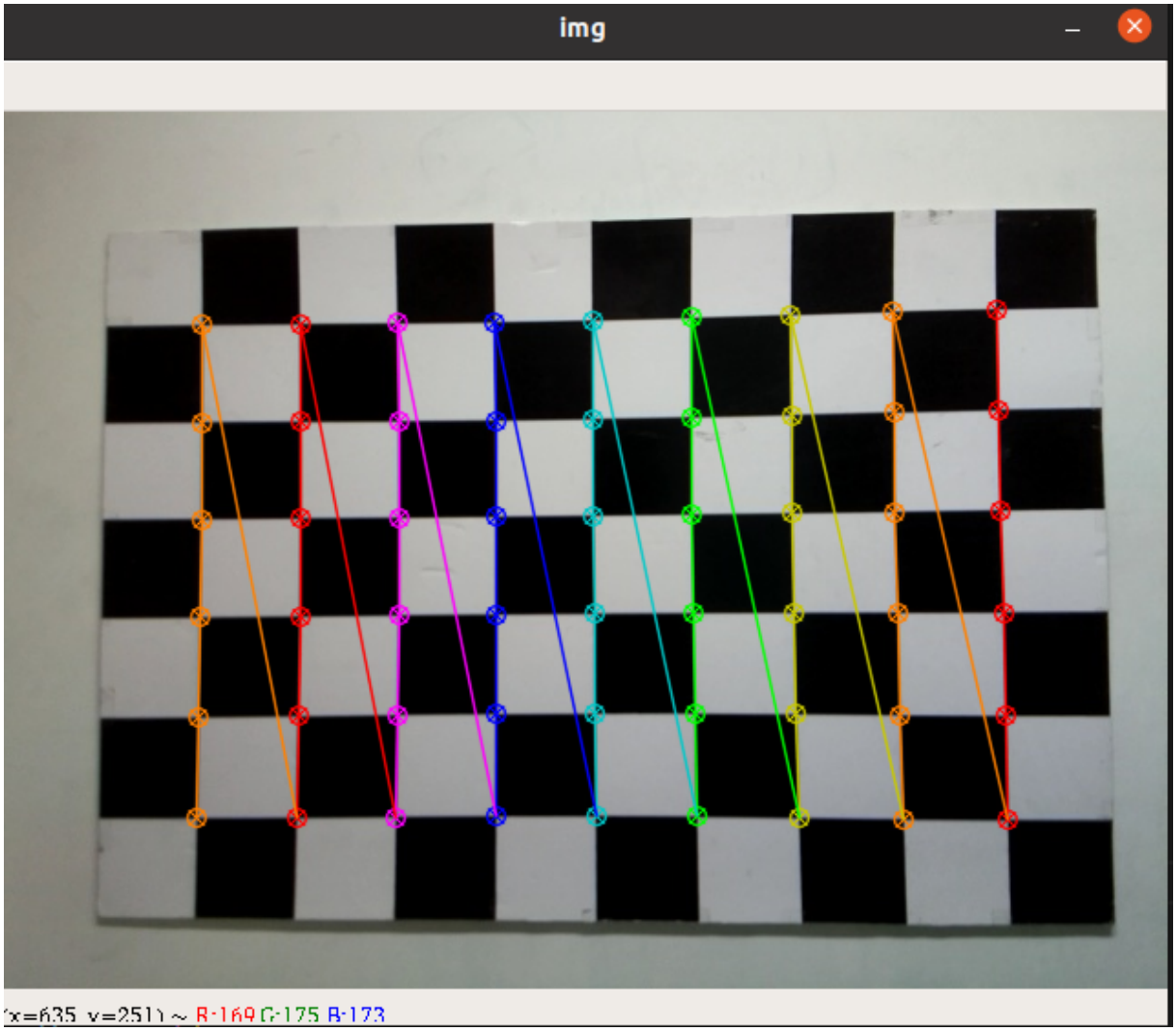


Figure 2.3: Chessboard with the detected corner points

$\mathbf{P_{obj}}$ is a homogenous vector expressed as $\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}$, we know that the Z is always 0 for the object coordinate frame. Thus by little simplification, we can get

$$\mathbf{p_{img}} = \mathbf{K} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.6}$$

Thus equation (2.5) will be modified to

$$\lambda \mathbf{H} = \mathbf{K} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{2.7}$$

For each view of the chessboard, we obtain a new homography matrix that introduces eight constraints. Furthermore, each view contributes six unknowns for the rotation and translation, referred to as the extrinsic parameters. The intrinsic parameters, which consist of the same

four unknowns for every view, remain constant. Thus, we need at least two or more views to over-constrain the system of equations and find the solution for the intrinsic parameters.

$$\mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \tag{2.8}$$

The above equation shows the H matrix as a set of column vectors thus equation (2.5) can be written as

$$\lambda \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \mathbf{K} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{2.9}$$

To obtain the Rotation and translation from the above equations

$$\begin{bmatrix} r_1 \\ r_2 \\ t \end{bmatrix} = \begin{bmatrix} \lambda \mathbf{K}^{-1} h_1 \\ \lambda \mathbf{K}^{-1} h_2 \\ \lambda \mathbf{K}^{-1} h_3 \end{bmatrix} \tag{2.10}$$

By the properties of the Rotation matrix, $r_1 \; and \; r_2$ are orthonormal so their dot product will be 0 and will have equal

$$r_1^T r_2 = h_1 \left(\mathbf{K}^{-1}\right)^T \mathbf{K}^{-1} h_2 = 0 \tag{2.11}$$

$$r_1^T r_1 = r_2^T r_2 \Longrightarrow h_1 \left(\mathbf{K}^{-1}\right)^T \mathbf{K}^{-1} h_1 = h_2 \left(\mathbf{K}^{-1}\right)^T \mathbf{K}^{-1} h_2 \tag{2.12}$$

Introduce a new matrix $\mathbf{B}$ which is defined as below.

$$\mathbf{B} = \left(\mathbf{K}^{-1}\right)^T \mathbf{K}^{-1} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{12} & b_{22} & b_{23} \\ b_{13} & b_{23} & b_{33} \end{bmatrix} = \begin{bmatrix} 1/f_x^2 & 0 & -c_z/f_x^2 \\ 0 & 1/f_y^2 & -c_y/f_y^2 \\ -c_x/f_x^2 & -c_y/f_y^2 & c_x/f_x^2 + c_y/f_y^2 + 1 \end{bmatrix}$$

Note that, $\mathbf{B}$ is a symmetric matrix. Hence any single term in the equation (2.12) can be represented by

$$h_i^T \mathbf{B} h_j \; where \; 1 \leq i, j \leq 2$$

$$h_i^T \mathbf{B} h_j = v_{ij}^T b$$

$$= \begin{bmatrix} h_{i1}h_{j1} & h_{i1}h_{j2} + h_{i2}h_{51} & h_{i2}h_{j2}h_{j3}h_{j1} + h_{i1}h_{j2}h_{i3}h_{j2} + h_{i2}h_{i3}h_{i3}h_{j3} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{12} \\ b_{22} \\ b_{13} \\ b_{23} \\ b_{33} \end{bmatrix}$$

The goal is to find matrix $\mathbf{B}$ We obtain the following equation for each view of the chess board. Here $\mathbf{b}$ is constant as it is a function of the camera but vector v changes for each view of the camera. So we can take pictures of the chessboard from the different views and stack all the vectors v on top of each other. Now, this can be solved using Singular Value Decomposition(SVD).

$$\begin{bmatrix} v_{12} \\ (v_{11} - v_{22})^T \end{bmatrix} \mathbf{b} = 0 \tag{2.13}$$

After getting matrix $\mathbf{B}$ the value for the intrinsic parameters can be computed as follows.

$$\begin{aligned} c_y &= (b_{12}b_{13} - b_{11}b_{23}) \,/\, (b_{11}b_{22} - b_{12}^2) \\ \lambda &= b_{33} - (b_{13}^2 + c_y\,(b_{12}b_{13} - b_{11}b_{23}))\,/b_{11} \\ f_x &= \sqrt{\lambda / b_{11}} \\ f_y &= \sqrt{\lambda b_{11} /\, (b_{11}b_{22} - b_{12}^2)} \\ c_x &= -b_{13}f_x^2/\lambda \end{aligned} \tag{2.14}$$

The extrinsic parameters can be obtained using the following equation.

$$\begin{aligned} r_1 &= \lambda \mathbf{K}^{-1} h_1 \\ r_1 &= \lambda \mathbf{K}^{-1} h_2 \\ r_3 &= r_1 \times r_2 \\ t &= \lambda \mathbf{K}^{-1} h_3 \end{aligned} \tag{2.15}$$

Finally, the distortion parameters can be obtained by using equation 2.16. This equation is for one corner point of the one view as these equations are linear with respect to the distortion parameters. Hence multiple rows of equations from the other corners and views can be stacked and a simple least square solution can be obtained for the distortion parameters.

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \begin{bmatrix} x_d \\ y_d \end{bmatrix} + \begin{bmatrix} 2p_1 x_d y_d + p_2\,(r^2 + 2x_d^2) \\ 2p_2 x_d y_d + p_1\,(r^2 + 2y_d^2) \end{bmatrix} \tag{2.16}$$

Figure 2.4 shows the image before and after the distortion correction.

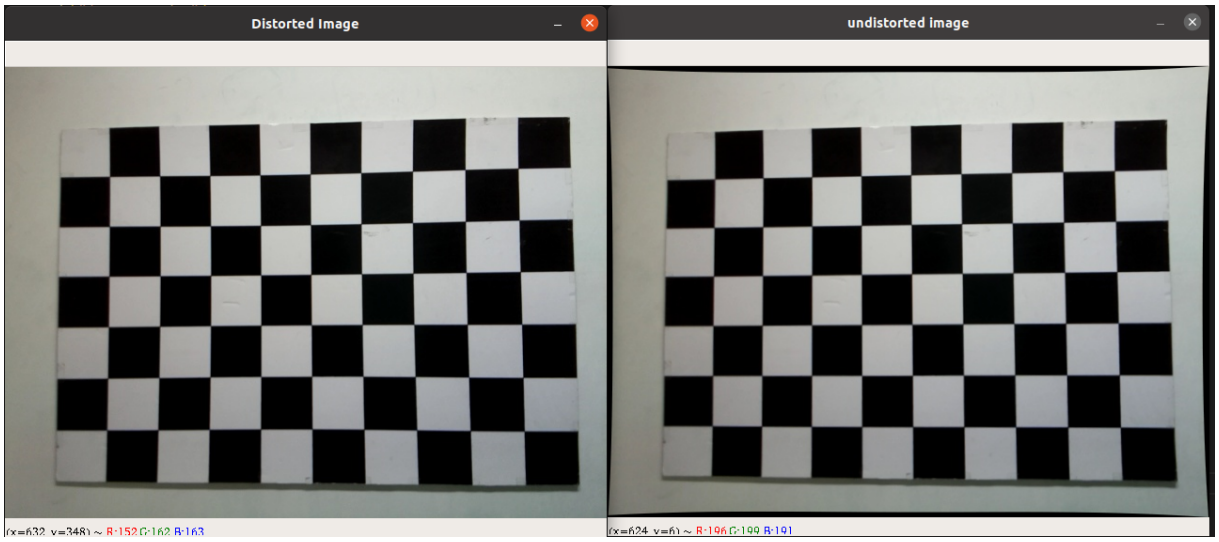


Figure 2.4: Distorted and Undistorted image

Few newer methods have been developed for camera calibration that use an Aprilgrid instead of a chessboard. These methods provide benefits such as the ability to use partially visible calibration boards and fully resolved pose of the targets [25].

In the proposed work, both the camera and IMU are used for pose estimation. Therefore, the transformation between the camera and IMU is needed. The Kalibr framework [24, 27] has been utilized for the calibration of these extrinsic parameters.

2.1.3 Quaternion Representation of Rotation

Rotation can be described using different representations, such as rotation matrices, Euler angles, and rotation vectors. However, each of these representations has its limitations. Rotation matrices require nine parameters, making them over-parameterized. Euler angles suffer from singularities, such as Gimbal lock, where a loss of one degree of freedom occurs. Other representations also encounter singularities or have drawbacks.

To overcome these challenges, complex numbers and quaternions are utilized. Complex numbers are used to represent vectors in the complex 2D plane, allowing for compact rotation representations. For 3D spatial rotation, quaternions provide an algebraic system similar to complex numbers but without singularities. Quaternions offer a concise representation for rotation and have been extensively studied and visualized. Additional resources on quaternions can be found in the article by 3Blue1Brown.Figure 2.5 provides a visualization of quaternion

rotation.

Mathematically, a quaternion is represented as:

$$\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} \tag{2.17}$$

Here, q_0 is the real part, and **i**, **j**, and **k** are the three imaginary parts of the quaternion. These imaginary parts satisfy the given relationships.

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases} . \tag{2.18}$$

The vector representation is given by

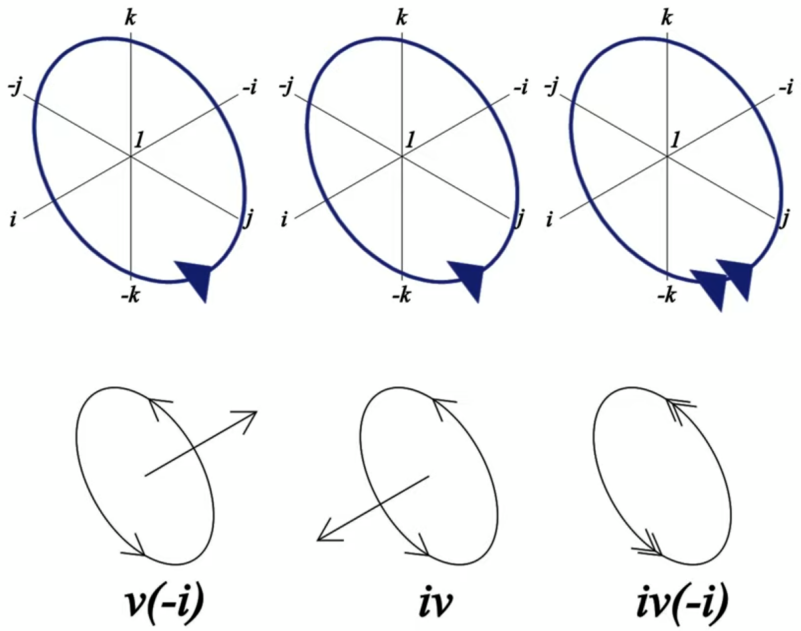


Figure 2.5: Quaternion Representation

$$\mathbf{q} = [s, \mathbf{v}]^T \tag{2.19}$$

where $s = q_0 \in \mathbb{R}, \quad \mathbf{v} = [q_1, q_2, q_3]^T \in \mathbb{R}^3$ The other representation of the quaternion which will be helpful for the rotation is

$$\mathbf{q} = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}(xi + yj + zk) \tag{2.20}$$

where θ is the angle of rotation and the $\begin{bmatrix} x & y & z \end{bmatrix}^T$ represents the vector along which the rotation is to be done. A quaternion is used to represent the rotation of any point in the 3D space. A three dimensional point $\mathbf{p} = [x, y, z]^T \in \mathbb{R}^3$, and $\mathbf{q}$ specifies the rotation. First, the given three dimensional point should be converted to an imaginary quaternion given by $\mathbf{p} = [0, x, y, z]^T = [0, \mathbf{v}]^T$, then the rotated point $\mathbf{p}'$ is given by

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}. \tag{2.21}$$

The quaternion multiplication specified here is given by

$$\mathbf{q}_a\mathbf{q}_b = \begin{bmatrix} s_a s_b - \mathbf{v}_a^T\mathbf{v}_b, s_a\mathbf{v}_b + s_b\mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b \end{bmatrix}^T. \tag{2.22}$$

2.2 Measurement Preprocessing

In the context of visual-inertial navigation, preprocessing is necessary for both visual and inertial measurements. In visual processing, the features are tracked across consecutive frames, while new features are detected in latest frame. For and IMU measurements, preintegration of an IMU data is required between the consecutive image frames.

2.2.1 Feature Tracking and Keyframe Selection

For feature detection in the new keyframe, the Shi-Tomasi Corner Detector [28] is utilized to detect features and filter out non-trackable features. After detection, the KLT optical flow algorithm [29] is used to track the detected features in consecutive image frames. A minimum number of good features is maintained in every image for the pose estimation in the further process.

As we are using a calibrated camera with known intrinsic and distortion parameters, the tracked two-dimensional features are firstly undistorted and then again projected onto the image plane after undergoing outlier rejection. The RANSAC outlier rejection with the fundamental matrix is used [30].

This step also includes keyframe selection. As a high frame rate camera is used, there is not much change in the scene between corresponding frames. To save computational resources, new features are not detected in every frame; feature detection is only performed for keyframes. We have the following criteria for keyframe selection:

- **Average parallax:** As a measure of whether the current frame should be designated as a new keyframe, the average parallax of the tracked features is compared between the current frame and the previous frame. If this average parallax exceeds a specified threshold, the current frame is identified as a new keyframe. It's important to note that the parallax can result from both translational and rotational motion. However, during rotation-only motion, feature triangulation is not feasible. To address this scenario, we employ short-term integration of gyroscope measurements from the IMU to compensate for rotation when computing the parallax.

- **Tracking quality:** In the case of fewer than a certain number of tracked features, the current frame is considered a keyframe. In this frame, new features will be detected to maintain the desired number of features. This criteria avoids complete loss of feature tracks.

2.2.2 Structure from Motion and Bundle Adjustment

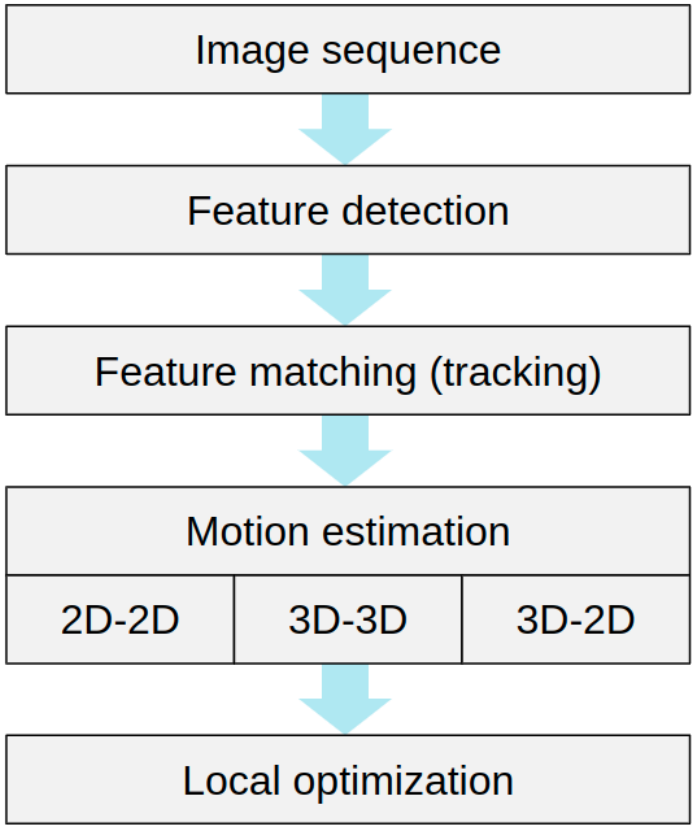


Figure 2.6: Flow of Structure from motion [1]

Figure 2.6 shows the complete flow of the Visual Odometry system.

- The SIFT features are detected and described in each frame. [31]
- KLT tracker to track features in consecutive images. [29]
- Two view geometry is applied.
- The 2D-2D correspondence with five-point algorithm. [32]
- Essential matrix is obtained which can be decomposed into R and t
- 3D-2D correspondences with PnP algorithm. [33]
- After the motion estimation between the consecutive frames a bundle adjustment is done to minimize the reprojection error between observed and predicted points

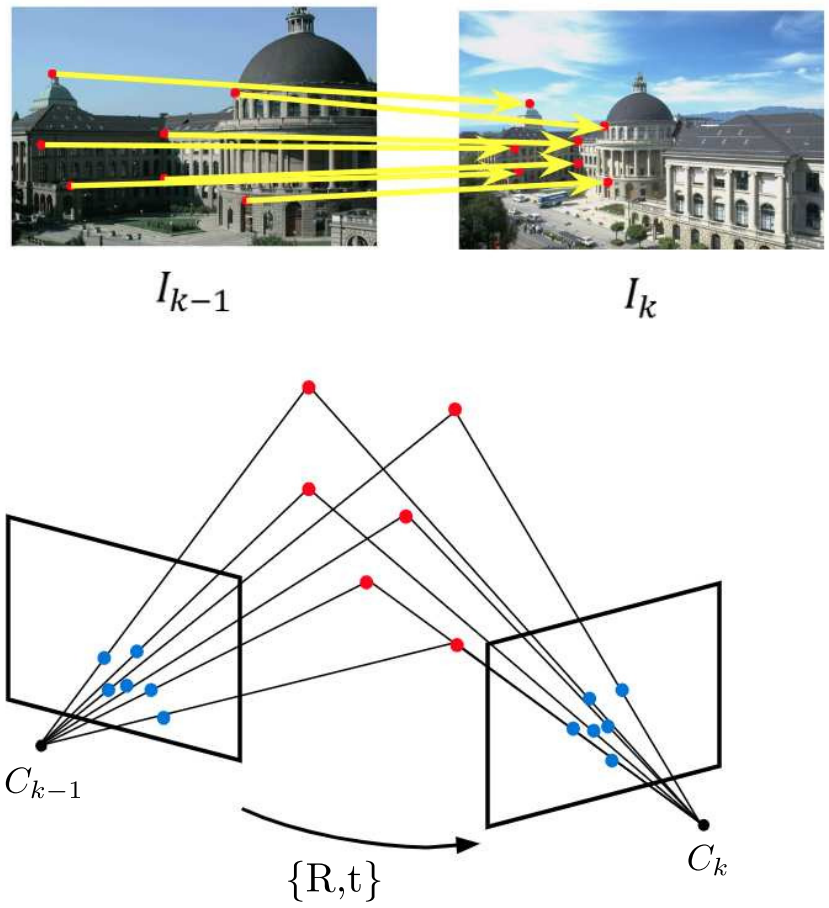


Figure 2.7: Motion Estimation [1]

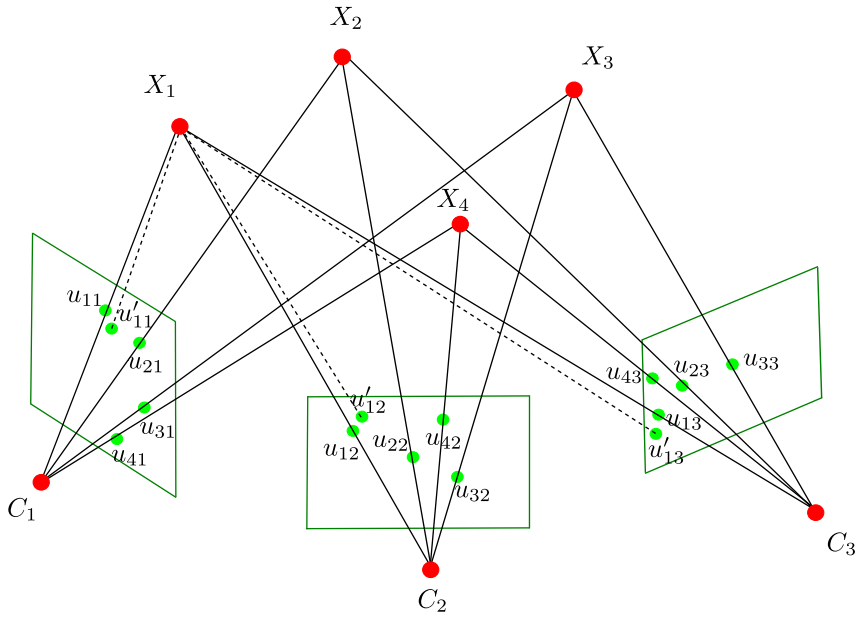


Figure 2.8: Bundle Adjustment

- Reprojection error

$$r_{ij} = u_{ij} - \pi\left(C_{j,}X_i\right) \tag{2.23}$$

where u_{ij} is observed i^{th} feature in j^{th} image and $\pi\left(C_{j,}X_i\right)$ is a non-linear operation which estimates the projection of 3D point onto an image plane [34]

- Reprojection error give the accuracy of the computed 3D points and pose

- The bundle adjustment is a non-linear least square problem that minimizes the total re-projection error. Figure 2.8

$$\min_{\mathcal{C}} \sum_{i=1}^{n} \sum_{j=1}^{m} \left(u_{ij} - \pi\left(C_j, X_i\right)\right)^2 \tag{2.24}$$

2.2.3 Notations and Frames

Different frames and rotation representations are introduced in the notation definitions. The gravity vector is aligned with the z-axis of the $(\cdot)w$, which stands for the world frame. $(\cdot)b$ stands for the body frame, which is the same as the IMU frame. $(\cdot)$ stands for the camera frame. Hamilton quaternions ($\mathbf{q}$) and rotation matrices ($\mathbf{R}$) are both used to express rotations. While rotation matrices ($\mathbf{R}$) are also utilised for the ease of rotating 3D vectors, quaternions ($\mathbf{q}$) are predominantly used in state vectors. $\mathbf{q}_b^w$ and $\mathbf{p}_b^w$ stand for the translation and rotation from the body frame to the world frame, respectively. The body frame used to take the kth

image is denoted by the subscript b_k and c_k represents the camera frame during image capture. The multiplication operation between two quaternions is denoted by $\otimes$. The gravity vector in the world frame is represented as $g_w = [0, 0, g]^T$. Finally, the symbol $\hat{(\cdot)}$ indicates the noisy measurements or estimations of a certain quantity.

2.2.4 Quaternion based IMU Preintegration

In most practical applications, the data obtained from the IMU is at a higher rate than the image frame rate, thus usually there are several IMU measurements between two consecutive frames as shown in Figure 2.9

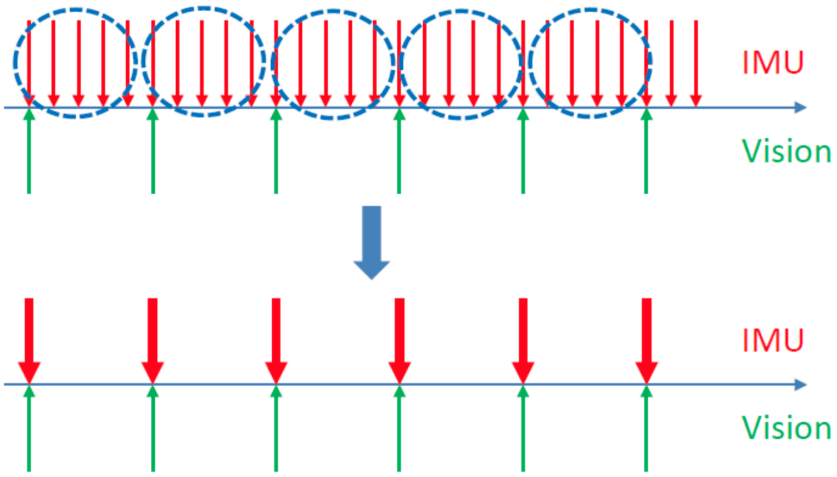


Figure 2.9: IMU Preintegration [2]

The position, velocity, and orientation states for any two consecutive image frames b_k and b_{k+1} can be propagated by the IMU measurements during the time interval $[t_k, t_{k+1}]$ in the world coordinate frame, which is given by

$$\begin{aligned}
\mathbf{p}^w_{b_{k+1}} &= \mathbf{p}^w_{b_k} + \mathbf{v}^w_{b_k}\Delta t_k + \iint_{t\in[t_k,t_{k+1}]} (\mathbf{R}^w_t(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a) - \mathbf{g}^w)\, dt^2 \\
\mathbf{v}^w_{b_{k+1}} &= \mathbf{v}^w_{b_k} + \int_{t\in[t_k,t_{k+1}]} (\mathbf{R}^w_t(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a) - \mathbf{g}^w)\, dt \\
\mathbf{q}^w_{b_{k+1}} &= \mathbf{q}^w_{b_k} \otimes \int_{t\in[t_k,t_{k+1}]} \frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}}_t - \mathbf{b}_{w_t} - \mathbf{n}_w)\mathbf{q}^{b_k}_t dt
\end{aligned} \tag{2.25}$$

where

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} -\lfloor\boldsymbol{\omega}\rfloor_\times & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix}, \lfloor\boldsymbol{\omega}\rfloor_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2.26}$$

The state propagation of IMU involves the pose and velocity of the frame b_k. When there are changes in these initial states, the measurements must be repropagated. However, in optimization-based algorithms, this propagation strategy can be computationally demanding. Therefore, we use the algorithm of preintegration. Changing the reference frame of refrence from the world frame to the local frame b_k, we can preintegrate only the components that are associated with linear acceleration and angular velocity. The preintegration process is as follows:

$$\begin{aligned}
\mathbf{R}_w^{b_k}\mathbf{p}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k}\left(\mathbf{p}_{b_k}^w + \mathbf{v}_{b_k}^w\Delta t_k - \frac{1}{2}\mathbf{g}^w\Delta t_k^2\right) + \boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\
\mathbf{R}_w^{b_k}\mathbf{v}_{b_{k+1}}^w &= \mathbf{R}_w^{b_k}(\mathbf{v}_{b_k}^w - \mathbf{g}^w\Delta t_k) + \boldsymbol{\beta}_{b_{k+1}}^{b_k} \\
\mathbf{q}_w^{b_k}\otimes\mathbf{q}_{b_{k+1}}^w &= \boldsymbol{\gamma}_{b_{k+1}}^{b_k}
\end{aligned} \tag{2.27}$$

where

$$\begin{aligned}
\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \iint_{t\in[t_k,t_{k+1}]}\mathbf{R}_t^{b_k}(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a)dt^2 \\
\boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \int_{t\in[t_k,t_{k+1}]}\mathbf{R}_t^{b_k}(\hat{\mathbf{a}}_t - \mathbf{b}_{a_t} - \mathbf{n}_a)dt \\
\boldsymbol{\gamma}_{b_{k+1}}^{b_k} &= \int_{t\in[t_k,t_{k+1}]}\frac{1}{2}\boldsymbol{\Omega}(\hat{\boldsymbol{\omega}}_t - \mathbf{b}_{w_t} - \mathbf{n}_w)\boldsymbol{\gamma}_t^{b_k}dt.
\end{aligned} \tag{2.28}$$

When considering b_k as the reference frame, the preintegration terms can be obtained solely from the IMU measurements. Equation (2.28) reveals that $\alpha_{b_{k+1}}^{b_k}, \beta_{b_{k+1}}^{b_k}, \gamma_{b_{k+1}}^{b_k}$ are exclusively related to the IMU biases rather than other states in b_k and b_{k+1}. If the change in the bias estimation of the IMU is small, we adjust $\alpha_{b_{k+1}}^{b_k}, \beta_{b_{k+1}}^{b_k}, \gamma_{b_{k+1}}^{b_k}$ using its first-order approximation with respect to bias. Otherwise, repropagation becomes necessary.

For the propagation of covariance, as a quaternion with four-dimensional is over parametrized its error term is defined as

$$\boldsymbol{\gamma}_t^{b_k} \approx \hat{\boldsymbol{\gamma}}_t^{b_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}\delta\boldsymbol{\theta}_t^{b_k}\end{bmatrix} \tag{2.29}$$

where $\delta\boldsymbol{\theta}t^{bk}$ is three-dimensional small perturbation.

The continuous error dynamics of terms in equation (2.28) can be derived as

$$
\begin{aligned}
\begin{bmatrix} \delta\dot{\boldsymbol{\alpha}}_t^{b_k} \\ \delta\dot{\boldsymbol{\beta}}_t^{b_k} \\ \delta\dot{\boldsymbol{\theta}}_t^{b_k} \\ \delta\dot{\mathbf{b}}_{a_t} \\ \delta\dot{\mathbf{b}}_{w_t} \end{bmatrix} &= \begin{bmatrix} 0 & \mathbf{I} & 0 & 0 & 0 \\ 0 & 0 & -\mathbf{R}_t^{b_k}\lfloor\hat{\mathbf{a}}_t - \mathbf{b}_{a_t}\rfloor_\times & -\mathbf{R}_t^{b_k} & 0 \\ 0 & 0 & -\lfloor\hat{\boldsymbol{\omega}}_t - \mathbf{b}_{w_t}\rfloor_\times & 0 & -\mathbf{I} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{\alpha}_t^{b_k} \\ \delta\boldsymbol{\beta}_t^{b_k} \\ \delta\boldsymbol{\theta}_t^{b_k} \\ \delta\mathbf{b}_{a_t} \\ \delta\mathbf{b}_{w_t} \end{bmatrix} \\
&+ \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\mathbf{R}_t^{b_k} & 0 & 0 & 0 \\ 0 & -\mathbf{I} & 0 & 0 \\ 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{n}_a \\ \mathbf{n}_w \\ \mathbf{n}_{b_a} \\ \mathbf{n}_{b_w} \end{bmatrix} = \mathbf{F}_t \delta\mathbf{z}_t^{b_k} + \mathbf{G}_t \mathbf{n}_t.
\end{aligned} \tag{2.30}
$$

After the zero-order hold discretization, the final preintegration terms are given by the following equations. The derivation of the following equations can be found in the [2, 35].

$$
\begin{aligned}
\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &\approx \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} + \mathbf{J}_{b_a}^{\alpha}\delta\mathbf{b}_{a_k} + \mathbf{J}_{b_w}^{\alpha}\delta\mathbf{b}_{w_k} \\
\boldsymbol{\beta}_{b_{k+1}}^{b_k} &\approx \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} + \mathbf{J}_{b_a}^{\beta}\delta\mathbf{b}_{a_k} + \mathbf{J}_{b_w}^{\beta}\delta\mathbf{b}_{w_k} \\
\boldsymbol{\gamma}_{b_{k+1}}^{b_k} &\approx \hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\mathbf{J}_{b_w}^{\gamma}\delta\mathbf{b}_{w_k} \end{bmatrix}.
\end{aligned}
$$

Chapter 3

Visual Inertial Odometry

Visual Inertial Odometry(VIO) system that combines visual and inertial sensor measurements to estimate the 6-degree-of-freedom (6-DoF) pose of a camera in real-time.Visual odometry utilizes visual information from cameras to estimate motion, while inertial navigation relies on measurements from inertial sensors such as accelerometers and gyroscopes. By fusing data from both sources, VINS Mono provides more accurate and robust pose estimation compared to using visual and inertial data.VINS Mono [2] incorporates advanced algorithms for feature detection, tracking, and sensor fusion. It leverages the rich visual information from a monocular camera and combines it with the precise motion measurements from inertial sensors. This fusion allows for reliable tracking even in challenging environments with rapid motion, occlusions, or limited visual features.

3.1 Full Architecture

The complete block diagram of monocular visual-inertial navigation system is shown in the Figure 3.1. The first section in the system is measurement preprocessing in which the features are detected and tracked which will be used in the further process. This section also performs the IMU preintegration which integrates the IMU measurements between two consecutive image frames. As the system is non-linear it needs a good initial estimation of the states which is provided by the initialization module. The initial values of the pose, velocity, gravity vector and 3-D features are calculated using SfM and IMU integration. The Tightly coupled Visual-Inertial fusion module with relocalization fuses the preintegrated IMU measurements and features observation. Loop Closure is also used to eliminate the drift.

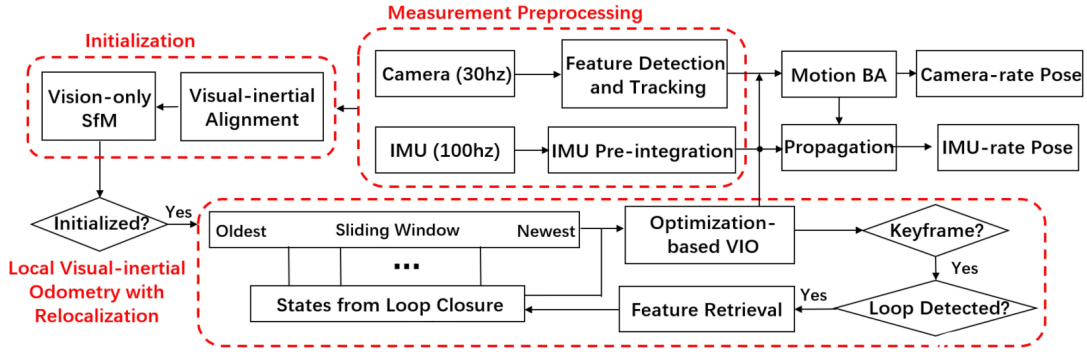


Figure 3.1: System Architecture [2]

3.2 Initialization

The VINS-mono is a highly non-linear system thus it requires a very good intialization to per-from well. These strating values are obtained from loose alignment of the IMU preintegration with the Vision-only SfM(Structure from Motion).

3.2.1 Vision only SfM

Initially, vision-only Structure-from-Motion (SfM) is employed to estimate the camera poses and feature positions, up to scale. A sliding window is maintained with multiple frames, and if stable feature tracking is achieved within the window, then using a Five-point algorithm [32] we get the rotation and up-to-scale translation between the consecutive frames. Subsequently, we set an arbitrary scale s, and then based on the depth all the features are triangulated. Then a PnP method [33] is used to estimate the poses of the other frames in window. At last a global bundle adjustment is done so that the reprojection error of the tracked features can be minimized. Here we take the first cam frame as the world frame which is denoted by $(\cdot)^{c0}$, every other pose of camera is $\bar{\mathbf{p}}_{c_k}^{c0}, \mathbf{q}_{c_k}^{c0}$ are represented with respect to this first camera pose$(\cdot)^{c0}$. As we already know the extrinsic parameters between an IMU and camera from the previous calibration process, all poses are translated from the c_0 frame to the IMU frame b.

$$\begin{aligned} \mathbf{q}_{b_k}^{c0} &= \mathbf{q}_{c_k}^{c0} \otimes \left(\mathbf{q}_c^b\right)^{-1} \\ s\bar{\mathbf{p}}_{b_k}^{c0} &= s\bar{\mathbf{p}}_{c_k}^{c0} - \mathbf{R}_{b_k}^{c0}\mathbf{p}_c^b \end{aligned} \tag{3.1}$$

here is an unknown scaling parameter.

3.2.2 Visual Inertial Alignment

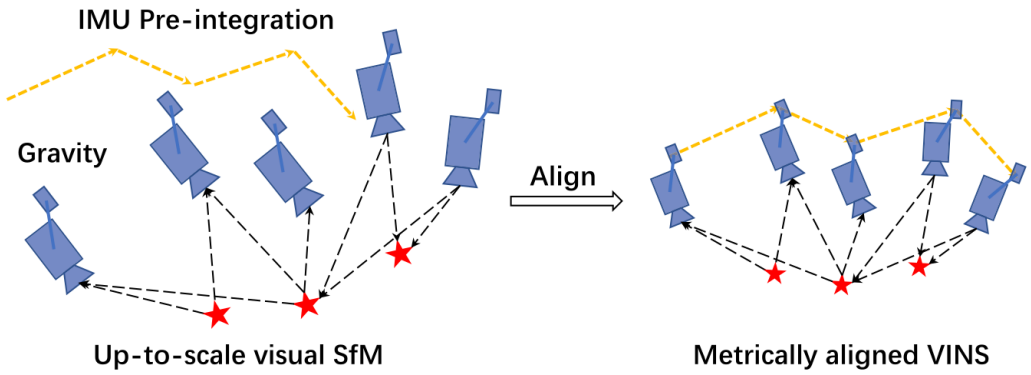


Figure 3.2: Visual Inertial Alignment [2]

Figure 3.2 shows the pictorial representation of the Visual Intertial alignment where the motion is first independently estimated using the SfM and IMU measurement and then both IMU preintegration and up-to-scale visual structure are matched.

1. Gyroscope Bias Calibration

The rotation $\mathbf{q}_{b_k}^{c_0}$ and $\mathbf{q}_{b_{k+1}}^{c_0}$ between two frames can be obtained using visual SfM, along with the relative constraint $\hat{\gamma}_{b_{k+1}}^{b_k}$ which IMU preintegration gives. We do the lineriaziation of the IMU preintegration term w.r.t bias of gyroscope and then we minimize the following cost function.

$$\begin{aligned} \min_{\delta b_w} \sum_{k \in \mathcal{B}} \left\| {\mathbf{q}_{b_{k+1}}^{c_0}}^{-1} \otimes \mathbf{q}_{b_k}^{c_0} \otimes \boldsymbol{\gamma}_{b_{k+1}}^{b_k} \right\|^2 \\ \boldsymbol{\gamma}_{b_{k+1}}^{b_k} \approx \hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2} \mathbf{J}_{b_w}^{\gamma} \delta \mathbf{b}_w \end{bmatrix} \end{aligned} \tag{3.2}$$

where $\mathcal{B}$ are the indexes of frames in the given window. The gyroscope bias $\mathbf{b}_w$ is then calibrated and used to repropagate all preintegration terms for the IMU.

2. Velocity, Gravity vector and Metric Scale Initialization

In order for the VIO to function, the other essential states must be initialized. The state vector is given by

$$\mathcal{X}_I = \left[\mathbf{v}_{b_0}^{b_0}, \mathbf{v}_{b_1}^{b_1}, \ldots \mathbf{v}_{b_n}^{b_n}, \mathbf{g}^{c_0}, s \right] \tag{3.3}$$

Let's consider two consecutive frame for that we have the following equations

$$\begin{aligned}\boldsymbol{\alpha}_{b_{k+1}}^{b_k} &= \mathbf{R}_{c_0}^{b_k}(s(\bar{\mathbf{p}}_{b_{k+1}}^{c_0} - \bar{\mathbf{p}}_{b_k}^{c_0}) + \frac{1}{2}\mathbf{g}^{c_0}\Delta t_k^2 - \mathbf{R}_{b_k}^{c_0}\mathbf{v}_{b_k}^{b_k}\Delta t_k) \\ \boldsymbol{\beta}_{b_{k+1}}^{b_k} &= \mathbf{R}_{c_0}^{b_k}(\mathbf{R}_{b_{k+1}}^{c_0}\mathbf{v}_{b_{k+1}}^{b_{k+1}} + \mathbf{g}^{c_0}\Delta t_k - \mathbf{R}_{b_k}^{c_0}\mathbf{v}_{b_k}^{b_k}).\end{aligned} \tag{3.4}$$

By using equation (3.1) and (3.4) we get the following measurement model.

$$\hat{\mathbf{z}}_{b_{k+1}}^{b_k} = \begin{bmatrix} \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} - \mathbf{p}_c^b + \mathbf{R}_{c_0}^{b_k}\mathbf{R}_{b_{k+1}}^{c_0}\mathbf{p}_c^b \\ \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} \end{bmatrix} = \mathbf{H}_{b_{k+1}}^{b_k}\mathcal{X}_I + \mathbf{n}_{b_{k+1}}^{b_k} \tag{3.5}$$

where

$$\mathbf{H}_{b_{k+1}}^{b_k} = \begin{bmatrix} -\mathbf{I}\Delta t_k & \mathbf{0} & \frac{1}{2}\mathbf{R}_{c_0}^{b_k}\Delta t_k^2 & \mathbf{R}_{c_0}^{b_k}(\bar{\mathbf{p}}_{c_{k+1}}^{c_0} - \bar{\mathbf{p}}_{c_k}^{c_0}) \\ -\mathbf{I} & \mathbf{R}_{c_0}^{b_k}\mathbf{R}_{b_{k+1}}^{c_0} & \mathbf{R}_{c_0}^{b_k}\Delta t_k & \mathbf{0} \end{bmatrix}.$$

Here $\mathbf{R}_{b_k}^{c_0}, \mathbf{R}_{b_{k+1}}^{c_0}, \bar{\mathbf{p}}_{c_k}^{c_0},$ and $\bar{\mathbf{p}}_{c_{k+1}}^{c_0}$ are obtained from SfM. So by solving the linear-least-square problem we can get the velocity, gravity vector, and scale parameter.

$$\min_{\mathcal{X}_I} \sum_{k\in\mathcal{B}} \left\| \hat{\mathbf{z}}_{b_{k+1}}^{b_k} - \mathbf{H}_{b_{k+1}}^{b_k}\mathcal{X}_I \right\|^2 \tag{3.6}$$

3.3 Tightly Coupled Monocular VIO

The initialization process gives the initial values of velocity, gravity vector, and scale parameters after that, we go for the tightly coupled monocular Visual Inertial estimation.

3.3.1 Sliding Window formulation

The the sliding window and its full state vector is given by

$$\begin{aligned}\mathcal{X} &= \left[\mathbf{x}_0,\, \mathbf{x}_1,\, \ldots\, \mathbf{x}_n,\, \mathbf{x}_c^b,\, \lambda_0,\, \lambda_1,\, \ldots\, \lambda_m\right] \\ \mathbf{x}_k &= \left[\mathbf{p}_{b_k}^w,\, \mathbf{v}_{b_k}^w,\, \mathbf{q}_{b_k}^w,\, \mathbf{b}_a,\, \mathbf{b}_g\right], k \in [0, n] \\ \mathbf{x}_c^b &= \left[\mathbf{p}_c^b,\, \mathbf{q}_c^b\right]\end{aligned} \tag{3.7}$$

As indicated by the x_k, the state of the IMU at the time of the kth image capture includes its position, velocity, orientation, and acceleration bias.The sliding window consists of a total of n keyframes and m features. $lambda_l$ represents the inverse distance of the lth feature from its first observation.

3.3.2 Cost function

The cost function used for the optimization is the same as the bundle adjustment but it also includes the inertial residuals for the minimization. Here the sum of prior and Mahalanobis norm of all measurements residuals are minimized to obtain a maximum posterior estimation. This cost function is given by

$$\min_{\mathcal{X}} \left\{ \|\mathbf{r}_p - \mathbf{H}_p \mathcal{X}\|^2 + \sum_{k \in \mathcal{B}} \left\| \mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}) \right\|_{\mathbf{P}_{b_{k+1}}^{b_k}}^2 + \sum_{(l,j) \in \mathcal{C}} \rho(\left\| \mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}) \right\|_{\mathbf{P}_l^{c_j}}^2) \right\} \tag{3.8}$$

where $\rho(s)$ is the Huber norm [36] which is given by

$$\rho(s) = \begin{cases} s & s \leq 1 \\ 2\sqrt{s} - 1 & s > 1. \end{cases} \tag{3.9}$$

The residuals for IMU and visual measurements, denoted as $\mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X})$ and $\mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X})$ respectively, are defined. In this case, $[B]$ represents all the measurements from the IMU, and $[C]$ represents the features seen morre than once. The term $\{\mathbf{r}_p, \mathbf{H}_p\}$ shows the prior information which we get from marginalization.

3.3.3 IMU Residuals

The IMU (Inertial Measurement Unit) residual represents the difference between the measured IMU readings and the predicted values based on the motion model of the system. By comparing the actual measurements with the expected values, the residual provides valuable information about the accuracy and reliability of the IMU data. The preintegrated IMU residuals are given

by

$$\begin{aligned}\mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X}) &= \begin{bmatrix} \delta\boldsymbol{\alpha}_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\beta}_{b_{k+1}}^{b_k} \\ \delta\boldsymbol{\theta}_{b_{k+1}}^{b_k} \\ \delta\mathbf{b}_a \\ \delta\mathbf{b}_g \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}_w^{b_k}(\mathbf{p}_{b_{k+1}}^w - \mathbf{p}_{b_k}^w + \frac{1}{2}\mathbf{g}^w\Delta t_k^2 - \mathbf{v}_{b_k}^w\Delta t_k) - \hat{\boldsymbol{\alpha}}_{b_{k+1}}^{b_k} \\ \mathbf{R}_w^{b_k}(\mathbf{v}_{b_{k+1}}^w + \mathbf{g}^w\Delta t_k - \mathbf{v}_{b_k}^w) - \hat{\boldsymbol{\beta}}_{b_{k+1}}^{b_k} \\ 2\Big[\ \mathbf{q}_{b_k}^{w^{-1}} \otimes \mathbf{q}_{b_{k+1}}^w \otimes (\hat{\boldsymbol{\gamma}}_{b_{k+1}}^{b_k})^{-1}\ \Big]_{xyz} \\ \mathbf{b}_{ab_{k+1}} - \mathbf{b}_{ab_k} \\ \mathbf{b}_{wb_{k+1}} - \mathbf{b}_{wb_k} \end{bmatrix}\end{aligned} \tag{3.10}$$

where $[.]_{xyz}$ extracts the vector part of a quaternion for the error-state representation. The data from gyroscope and accelerometer also used to get an online correction.

3.3.4 Camera Residuals

In the paper, [2] the author has used the fisheye camera model which uses the projection of features on a unit sphere. In our experimental setup, we have used the Pinhole Camera model for modeling our camera which assumes the projection of features on a normalized image plane. The feature measurement $\hat{z}_l^{c_j} = [\hat{u}_l^{c_j}, \hat{v}_l^{c_j}]^{\mathrm{T}}$ is the observation of the feature in the normalized image plane. The residual term is the reprojection error, which is defined by

$$\begin{aligned} r_{\mathcal{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X}) &= \begin{bmatrix} \frac{f_{x_l}^{c_j}}{f_{z_l}^{c_j}} - \hat{u}_l^{c_j} \\ \frac{f_{y_l}^{c_j}}{f_{z_l}^{c_j}} - \hat{v}_l^{c_j} \end{bmatrix} \\ \mathbf{f}_l^{c_j} &= \begin{bmatrix} f_{x_l}^{c_j} \\ f_{y_l}^{c_j} \\ f_{z_l}^{c_j} \end{bmatrix} = \mathbf{R}_0^{c_j}(\mathbf{p}_{c_i}^0 - \mathbf{p}_{c_j}^0 + \lambda_l\mathbf{R}_{c_i}^0\mathbf{u}_l^{c_i}) \end{aligned} \tag{3.11}$$

where $\mathbf{u}_l^{c_i} = [u_l^{c_j}, v_l^{c_i}, 1]^{\mathrm{T}}$ gives the first observation of the feature and we consider it to be noiseless. Noise matrix $\mathbf{P}_l^{c_j}$ gives residual covariance

3.3.5 Marginalization

To simplify the computational complexity of optimization-based VIO, marginalization is performed. This involves selectively removing IMU states and features from the sliding window and converting associated measurements into priors.

If the second-last frame is a keyframe, the oldest frame and its measurements are marginalized. If the second-to-last frame is not a keyframe, only its visual measurements are marginalized while retaining IMU measurements. This ensures accurate feature triangulation and significant accelerometer variations.

Marginalization is carried out using the Schur complement method. The marginalized states contribute to a new prior that is added to the existing prior for refined estimation.

3.4 Relocalization

The marginalization module limits the computation usage but it introduces drift in the system. For the elimination of this drift, a tightly coupled relocalization module is used which is integrated with the VIO system. The first step in the relocalization is loop-detection which identifies the places which have already occurred. By using a bag of words feature-level correspondences are generated which are then tightly integrated into the VIO module. Due to this, it adds more constraints to the system which result in a drastic decrease in the drift of the estimation. Here we use multiple observations of multiple features for the relocalization which ultimately results in higher accuracy and state estimation smoothness. Figure 3.3 shows the process of the relocalization

3.4.1 Loop Closure

The process in which the system detects the earlier visited place again is called loop detection. Here DBoW2 [37] is used which uses a bag of words for loop detection, this is one of the most used place recognition algorithm. For gettig a better recall rate additional features are detected and aroun more 500 corners are detected and described by th BRIEF descriptor [38]. These BRIEF descriptors are saved and raw images are discarded to reduce memory consumption. After the loop detection, using the feature correspondences a link is obtained between the sliding window and loop closure. For the removal of outliers, 2D-2D (using RANSAC) [30] as and 3D-

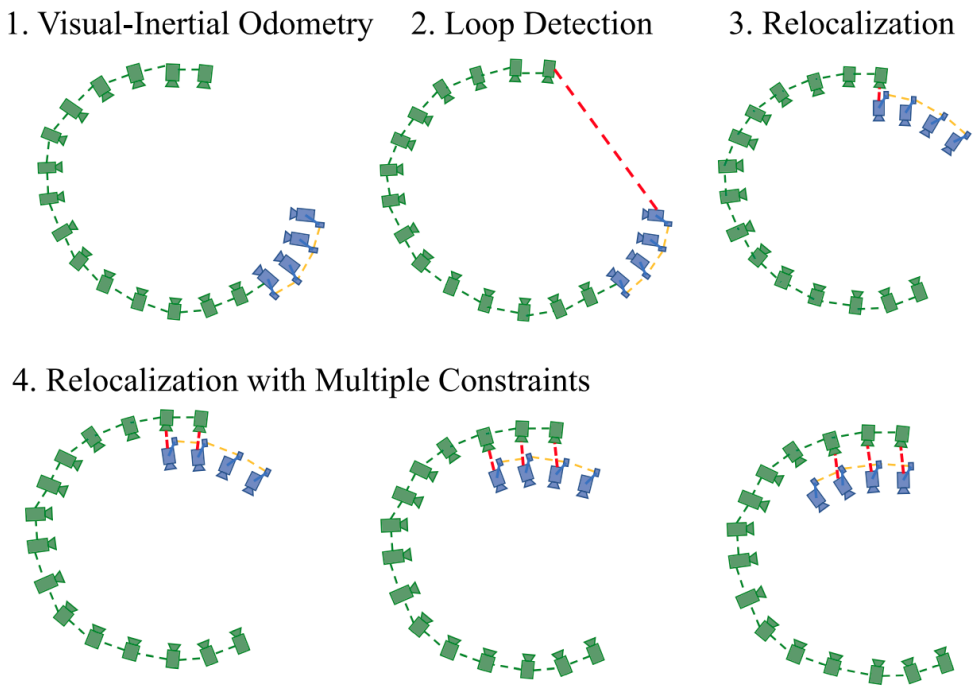


Figure 3.3: Relocalization [2]

2D(The PnP test with RANSAC) [33] are used. After the outlier rejection, the inliers are used to perform relocalization.

3.4.2 Tightly Coupled Relocalization

In relocalization we align the current sliding window and the previous poses Then all the measurements including visual measurement , IMU data and retrieved features are jointly optimized. The modified cost function is given by

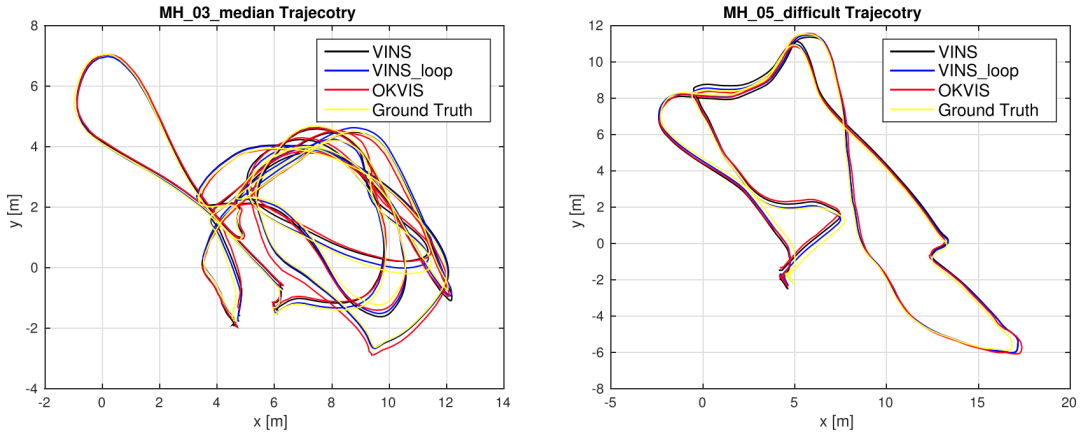
$$\min_{\mathcal{X}} \left\{ \|\mathbf{r}_p - \mathbf{H}_p\mathcal{X}\|^2 + \sum_{k\in\mathcal{B}} \left\|\mathbf{r}_{\mathcal{B}}(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \mathcal{X})\right\|^2_{\mathbf{P}_{b_{k+1}}^{b_k}} + \sum_{(l,j)\in\mathcal{C}} \rho(\left\|\mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_l^{c_j}, \mathcal{X})\right\|^2_{\mathbf{P}_l^{c_j}}) + \underbrace{\sum\nolimits_{(l,v)\in\mathcal{L}} \rho(\|\mathbf{r}_{\mathcal{C}}(\hat{\mathbf{z}}_l^{v}, \mathcal{X}, \hat{\mathbf{q}}_v^{w}, \hat{\mathbf{p}}_v^{w})\|^2_{\mathbf{P}_l^{c_v}})}_{\text{reprojection error in the loop-closure frame}} \right\} \tag{3.12}$$

where $\mathcal{L}$ is the set of observations features which we have got from loop-closure frames. (l, v) represents lth feature which is observed in the loop-closure frame v.

3.5 VIO Dataset Comparision

The benchmark comparison of this algorithm is given in [2], this algorithm is compared with the other state-of-the-art monocular-VIO algorithms like OKVIS [39]. The numerical analysis is performed to show the accuracy of the system in detail.

The VINS-mono is evaluated using the EuRoC MAV datasets [40]. This is a publically available dataset, which is collected using onboard stereo camera and IMU. For the evaluation, only the camera on left is used.



(a) (b)

Figure 3.4: VINS-mono trajectory comparison with OKVIS [2]

Figure 3.4 shows the comparison result of the VINS-mono with the state-of-the-art OKVIS on two different datasets from the EuRoC namely MH_03_medium and H_05_difficult. Figure 3.5 shows the metric comparison of VINS-mono with the OKVIS, which shows that VINS-mono with loop closure outperforms the state-of-the-art OKVIS in the long range. The detailed Benchmark comparison between different Monocular Visual Inertial Odometry Algorithm is performed and evaluated in [41].

3.6 Experimental Results

To experiment with the algorithm for further use, the hardware setup with IMU and a monocular camera is developed in Control Lab at IIT Dharwad. A dataset containing the camera and IMU measurements is collected covering the pathway of the complete Lab. This dataset is then evaluated with VINS-mono framework.

Sequence	OKVIS	VINS	VINS_loop
MH_01_easy	0.33	0.15	**0.12**
MH_02_easy	0.37	0.15	**0.12**
MH_03_medium	0.25	0.22	**0.13**
MH_04_difficult	0.27	0.32	**0.18**
MH_05_difficult	0.39	0.30	**0.21**
V1_01_easy	0.094	0.079	**0.068**
V1_02_medium	0.14	0.11	**0.084**
V1_03_difficult	0.21	**0.18**	0.19
V2_01_easy	0.090	**0.080**	0.081
V2_02_medium	0.17	0.16	**0.16**
V2_03_difficult	0.23	0.27	**0.22**

Figure 3.5: RMSE [3] in EuROC Datasets of VINS-mono [2] and OKVIS

3.6.1 Hardware Configuration

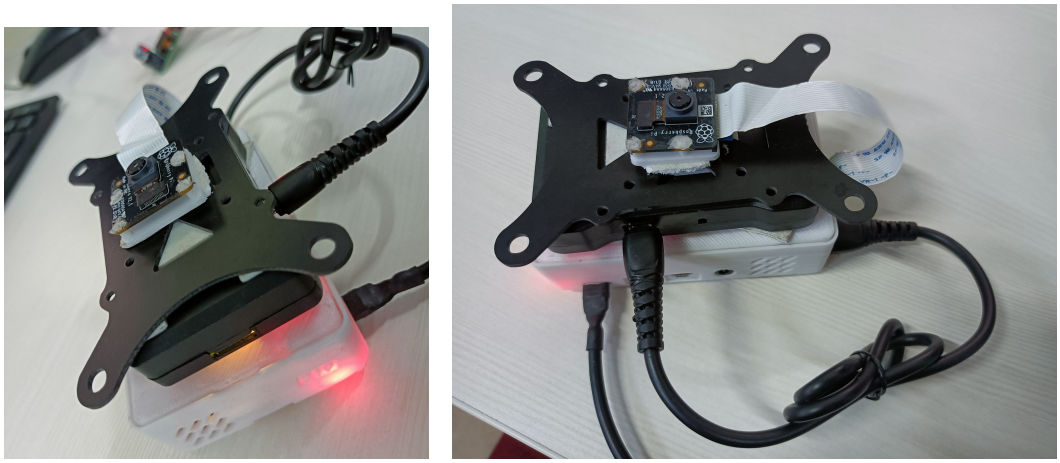
Figure 3.6 shows the hardware used for the experiment. A hand-held module for the VIO is developed which used Raspberry Pi Camera-V2 as a monocular camera and a low-cost Flight controller for the IMU data. Raspberry Pi-4 with 4GB RAM is used as SBC(Single Board Computer) for high-level tasks. The Hardware specification of the used modules are as follow:

1. **Raspberry Pi-4 :**
    - 4 GB of DDR4 RAM
    - Quad core Cortex-A72 (ARM v8) 64-bit SoC
    - Power Supply as 5V DC via USB-C connector
    - 1.5 GHz Clock frequency
2. **Pixhawk 2.4.8 FCU :**
    - 32bit STM32F427 Cortex M4 core with FPU.
    - 168 MHz clock frequency
    - 128 KB RAM and 2 MB Flash Memory

- ST Micro L3GD20H 16-bit gyroscope.
- ST Micro X4HBA 303H 14-bit accelerometer/magnetometer.

3. **: Raspberry Pi Camera version 2**

   - Sony IMX219 8-megapixel image sensor
   - Supports 1080 at 30 Hz, 720 at 60 Hz
   - 128 KB RAM and 2 MB Flash Memory



(a) (b)

Figure 3.6: Hardware Module for VIO.

3.6.2 Camera-IMU Calibration

As shown in Figure 3.7 we have used the Kalibr Multisensor framework [24, 25] with Aprilgrid for the calibration of the intrinsic parameters of the camera as well as the extrinsic parameters such as Transformation between IMU and Camera. The calibration result for the given setup is as follows which shows the calibrated intrinsic and extrinsic parameters, the reprojection error of the calibrated camera is shown in figure 3.8.

The Calibrated values of the hardware module are as follows:

- Intrinsic Camera Matrix $\mathbf{K} = \begin{bmatrix} 501 & 0 & 318 \\ 0 & 500 & 223 \\ 0 & 0 & 1 \end{bmatrix}$
- Distortion parameters $(\mathbf{k1}, \mathbf{k2}, \mathbf{p1}, \mathbf{p2}, \mathbf{p3}) = [0.15,\ -0.27,\ -1.5e^{-4},\ -7.33e^{-04},\ 0]$

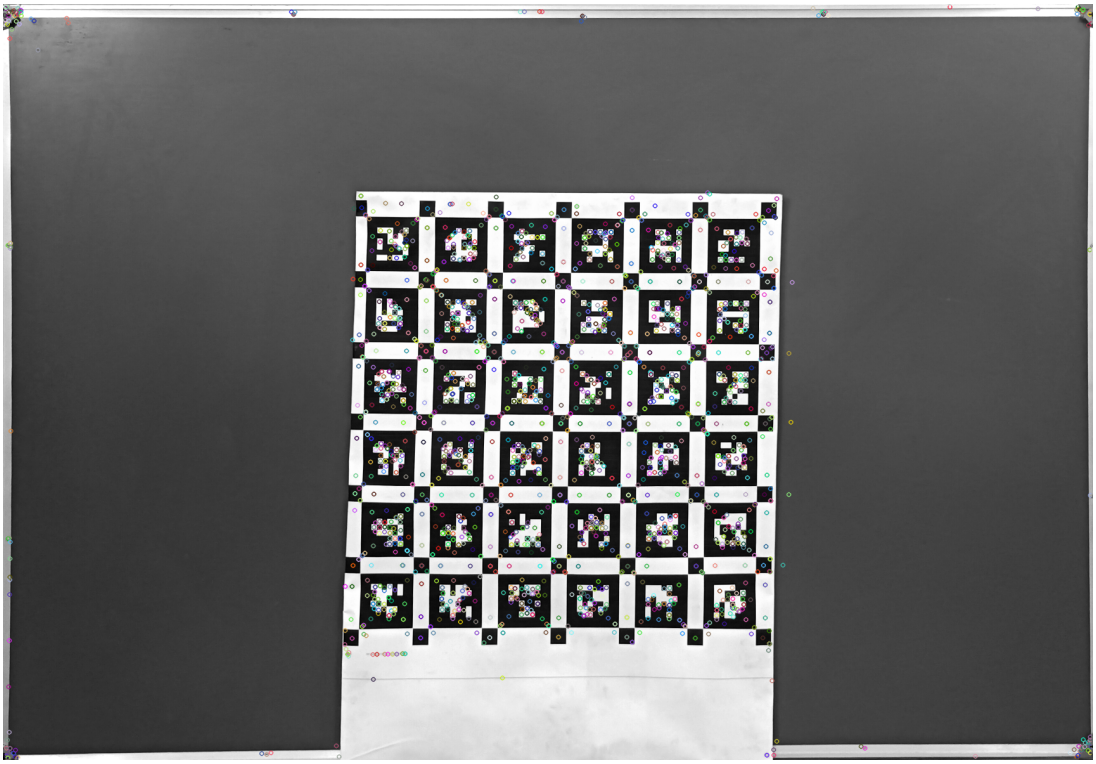


Figure 3.7: Calibration Board with April-Grid

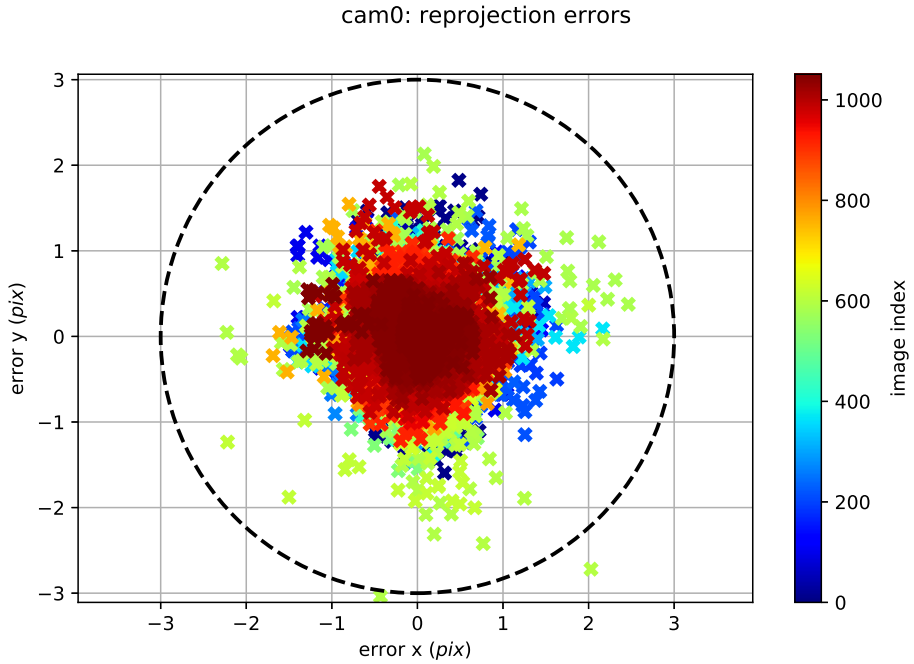


Figure 3.8: Camera Reprojection Error

- Transformation matrix $\mathbf{T}_b^c = \begin{bmatrix} 0.013 & 0.999 & 0.0532 & -0.044 \\ 0.998 & -0.014 & 0.007 & 0.007 \\ 0.008 & 0.053 & -0.998 & -0.082 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3.6.3 Visual-Inertial Odometry

For the VIO experiment, a 56.82 meters long trajectory dataset of Lab is collected and estimation is done on it using the VINS-mono implementation [2] and the previously calibrated camera parameters. Figure 3.9 shows the Trajectory tracked by the handheld VINS module in the Lab. The full video of this can be found at `https://youtu.be/uIglnuNkf7I`. Figure 3.10 shows the evaluation of the generated trajectory, due to the lack of an external tracking system, measure tape was used for generating ground truth with limited checkpoints due to which there can be some error in the ground truth generation too. The RMSE error calculated at the checkpoints for the trajectory is 0.49 and 0.44 along the x and y- axis respectively. The final drift is around 1.2 meters along the y-axis..

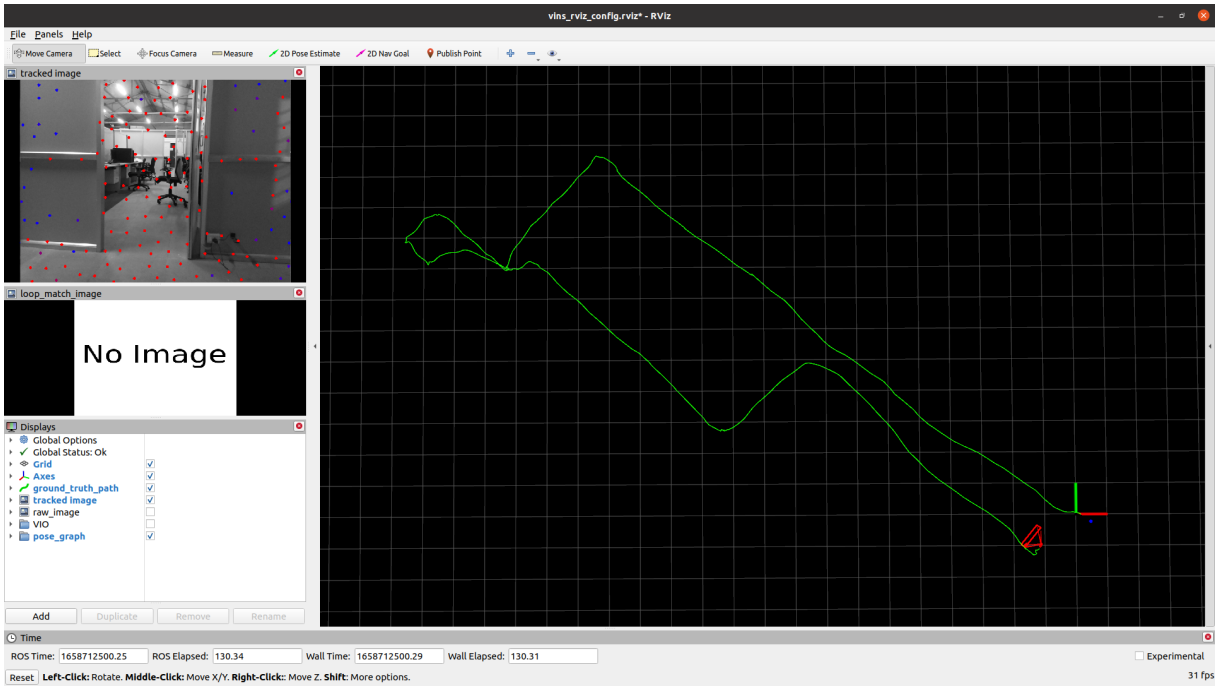


Figure 3.9: Trajectory tracked by the VINS-estimator on the Lab dataset

Note: For the trajectory evaluation the orientation and scale factor of the estimated trajectory are corrected which was the issue from the initialization process. There can also be a human error as the ground truth is collected by the measuring tape. The other reason for the error in the trajectory estimation can be due to the high reprojection error of the low-cost camera which is nearly 1.2. These results show that this approach can be used for the localization of a quadcopter with only a single monocular camera and an IMU.

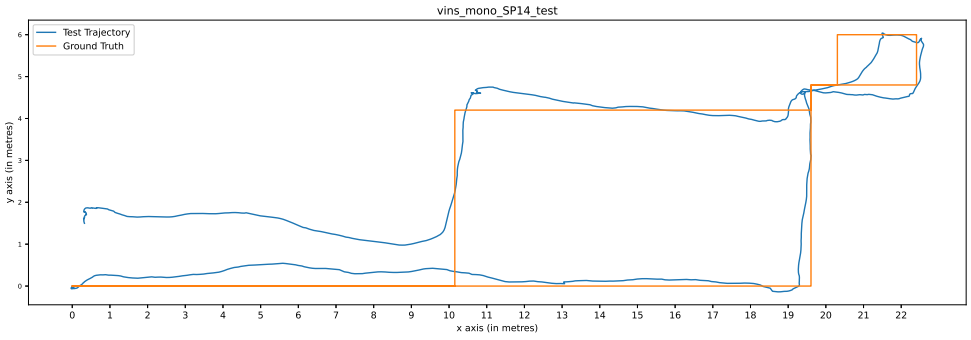


Figure 3.10: Trajectory tracked by the VINS-estimator and ground truth

Chapter 4

3D Dense Mapping

In mobile robotics, AR, CAD etc, depth reconstruction is one of the important problem. There are some sensors such as LiDARs, RGB-D cameras which can explicitly provide depth data but these sensors are either too expensive of bulky in nature which limits its usage in the small size robots such as quadcopters. Thus there is very strong interest of depth estimation using a monocular camera as almost every mobile robots are equipped with atleast a single camera. For recent applications in AR and robotics there is a need of fast computation of high resolution 3D reconstruction. Current development of the depth estimation are moving on using Artificial neural network specially CNNs to achieve 2D to 3D reconstruction. A 3D depth estimation framework named DenseDepth in [4] is used for the depth estimation from an RGB image. By using the depth estimation from CNN the 3D point cloud of the scene is created using the known camera parameters. Along with the VIO discussed in Chapter 3 and the point cloud created and Octomap is genreated of the environment.

4.1 3D Depth Estimation

The problem of 3D depth estimation is an ill-posed problem because problem such small coverage of scene, scale ambiguities, reflective or trasnculent items all contributes to the cases where where it is difficult to get geometry from apperance. In practice, the better way to capture a depth data is to use hardware assistance such as using IR-based sensors, LiDAR or stereo cameras, but these sensor are heavy and expensive which is not an appropriate choice for an indoor quadcopter. Recent methods such as CNN based 3D depth reconstruction are fair enough to get depth map from an RGB images In this work we see more recent solutions that depends on

deep neural network.

4.1.1 Multiview Reconstruction

Recently, a novel approach for multiview stereo reconstruction utilizing convolutional neural networks (CNN) has been introduced in the work [42]. Another study by [43] presented a method for joint key-frame-based camera tracking estimation of depth. In this work, our objective is to enhance the performance of monocular depth estimators.

4.1.2 Monocular Depth Estimation

Various CNN-based methods have addressed the task of monocular depth estimation, formulating it as a regression problem to predict depth maps from single RGB images [44, 45]. Although these methods have shown continuous improvements in performance, challenges remain regarding the quality and resolution of the estimated depth maps, leaving room for further enhancements.

4.1.3 Transfer Learning

Transfer learning has been proven to be an effective technique in various contexts, including 3D reconstruction. In a study by Zamir et al [46], it shows the exaamination of efficiency of transfer learning, with a focus on tasks related to 3D reconstruction. Here we leverages the concept of transfer learning by utilizing image encoders that were initially developed for image classification. We discovered that using encoders that preserve the spatial resolution of the input resulted in more accurate depth estimations, particularly when combined with skip connections.

4.1.4 Encoder-Decoder

Encoder-decoder networks have been a major contributor in solving various vision-related problems such as image segmentation, optical flow estimation, and image restoration. Recently, the use of these architectures has yielded impressive results for depth estimation, both in supervised and unsupervised settings. These methods often incorporate one or more encoder-decoder networks as a component of a larger network. Here, we utilize a simple, single encoder-decoder architecture that incorporates skip connections.

4.2 Network Architecture

Figure 4.1 shows the basic encoder-decoder network architecture.

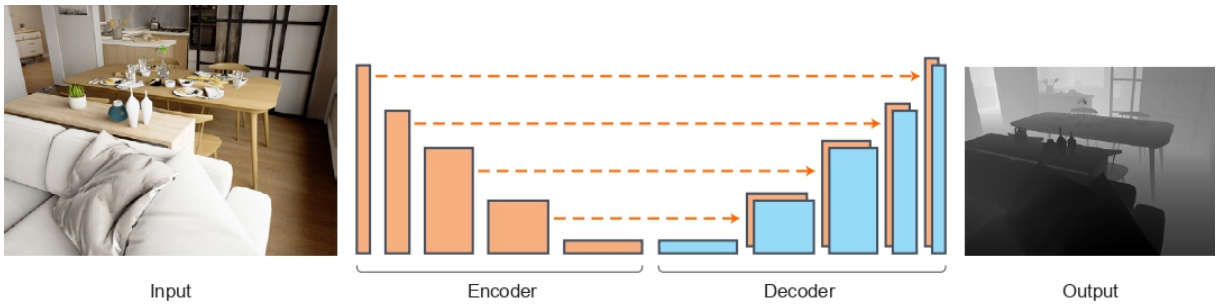


Figure 4.1: Network Architecture [4]

4.2.1 Architecture

The encoder-decoder network with skip connections used in Table 4.1 is composed of a modified DenseNet-169 network. Here the top layers are removed,which is serving as the encoder. The decoder starts with a 1x1 conv layer, and also have equal number of output channels same as the encoder. It then includes a series of upsampling layer, where each consisting of 2x bilinear upsampling layer and two 3x3 convolutional layers. The output filters of these layers are set to half the number of input filters, and the first convolutional layer operates on the concatenation of the previous layer's output and a pooling layer from the encoder, ensuring matching spatial dimensions. All the upsampling block is followed by a leaky ReLU activation function. [4]

4.2.2 Loss Function

The loss function commonly used for depth regression problems involves comparing the true depth map y with the predicted depth map y'. The selection of the loss function can have a very high impact on the training speed and overall performance of depth estimation. Various loss function can be found in the existing literature. In this work, our goal is to use a function that strikes a balance between accurately reconstructing depth images by reducing the disparity in depth values and gives penalties to the distortions. These distortions usually relates to the object boundaries within the scene.

The loss function used for the network is defined as follows:

$$L(y, \hat{y}) = \lambda L_{\text{depth}}(y, \hat{y}) + L_{\text{grad}}(y, \hat{y}) + L_{\text{SSIM}}(y, \hat{y}). \tag{4.1}$$

The first loss term, L_{depth}, represents the point-wise L_1 loss computed on the depth values:

$$L_{\text{depth}}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} |y_p - \hat{y}_p|. \tag{4.2}$$

The second loss term, L_{grad}, corresponds to the L_1 loss calculated on the image gradient $\boldsymbol{g}$ of the depth image:

$$L_{\text{grad}}(y, \hat{y}) = \frac{1}{n} \sum_{p}^{n} |\boldsymbol{g}_{\text{x}}(y_p, \hat{y}_p)| + |\boldsymbol{g}_{\text{y}}(y_p, \hat{y}_p)|, \tag{4.3}$$

where $\boldsymbol{g}_{\text{x}}$ and $\boldsymbol{g}_{\text{y}}$ represent the differences in the x and y components of the depth image gradients between y and $\hat{y}$, respectively.

Lastly, L_{SSIM} incorporates the Structural Similarity (SSIM) term, which is a widely-used metric for image reconstruction tasks and has proven to be effective for depth estimation CNNs:

$$L_{\text{SSIM}}(y, \hat{y}) = \frac{1 - \text{SSIM}(y, \hat{y})}{2}. \tag{4.4}$$

The value of λ is set to 0.1.

4.3 Depth to Point Cloud

A depth image (or depth map) is an image where each pixel represents the distance of that point in the image from the sensor's coordinate system. A point cloud is a set of points in 3D space that can be represented by a set of coordinates (x,y,z). The process of converting a depth image to a point cloud involves mapping each pixel in the depth image to a 3D point in space by using its x,y,z coordinates. This process is also known as depth to point cloud conversion, it allows the point cloud to be visualized in 3D space and can be used for various applications such as robot navigation, object recognition, and 3D reconstruction. Figure 4.2 shows the flow of conversion from RGB to Pointcloud.

4.3.1 Point cloud computing

To compute the point cloud, the process involves converting the depth pixel from the 2D coordinate system of the depth image to the 3D coordinate system of the depth camera, which includes

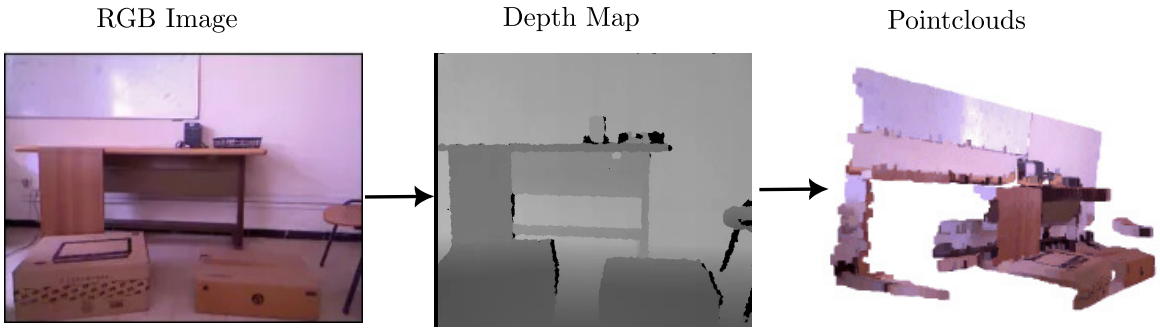


Figure 4.2: Depth to Pointcloud

the coordinates (x, y, and z). The computation of the 3D coordinates is achieved through the utilization of the following formulas. In these formulas, depth(i, j) represents the depth value located at the intersection of row i and column j:

$$\begin{cases} z = \text{depth}(i, j) \\ x = \dfrac{(j - c_x) \times z}{f_x} \\ y = \dfrac{(i - c_y) \times z}{f_y} \end{cases} \tag{4.5}$$

Figure 4.3 shows the point cloud generated from a monocular camera using the discussed CNN based architecture.

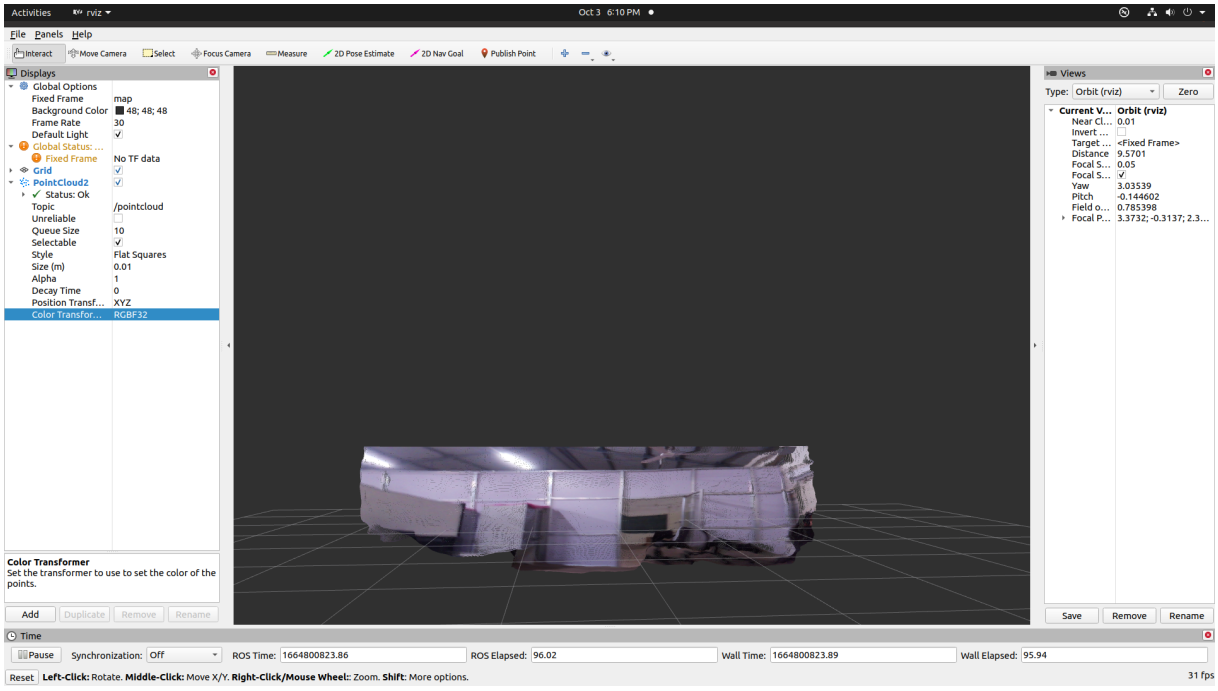


Figure 4.3: Point Cloud generated from monocular image.

4.4 Octomap

An octomap is a data structure and algorithm for efficient 3D mapping, specifically designed for robots operating in unknown or partially known environments. It uses an octree data structure to represent 3D space, dividing it recursively into eight voxels (3D pixels) until the desired level of resolution is reached. Occupancy probability is assigned to each voxel, representing the likelihood that the voxel is occupied by an object. Octomaps are often used in robotics for tasks such as localization, mapping, and object detection.

4.4.1 Octrees

An octree is a tree data structure used for efficient 3D mapping and spatial partitioning. It is a type of space-partitioning tree in which each internal node has exactly eight children. The tree is built by recursively subdividing the space into eight octants, until a desired level of resolution is reached. The leaf nodes of the tree represent small volumes of space, called voxels, which can be used to represent the occupancy of a 3D environment. Figure 4.4 shows the octree structure.

An octree can be used to represent a 3D environment in a more memory-efficient way than a 3D grid or voxel map. Because the tree structure allows for efficient traversal, it can be used for various tasks such as collision detection, ray tracing, and spatial partitioning of objects. The tree structure also allows for fast insertion and deletion of objects in the 3D environment, making it well suited for dynamic environments. Octrees have many applications in fields such as computer graphics, robotics, and video games.

4.4.2 Probabilistic sensor fusion

This method utilizes occupancy grid mapping, as first proposed by Moravec and Elfes in 1985, to integrate sensor readings. The probability of a leaf node being occupied, given the sensor measurements up to time t, is represented by P(n—z1:t). This probability is estimated using the sensor data given by the following equation

$$P\left(n \mid z_{1:t}\right) \quad = \left[1 + \frac{1 - P\left(n \mid z_t\right)}{P\left(n \mid z_t\right)} \frac{1 - P\left(n \mid z_{1:t-1}\right)}{P\left(n \mid z_{1n-1}\right)} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{4.6}$$

where z_t is the current measurement,$P(n)$ is prior probability and $P(n \mid z_{1:t-1})$ is the previous estimate.$P(n \mid z_t)$ shows the n^{th} voxel to be occupied.

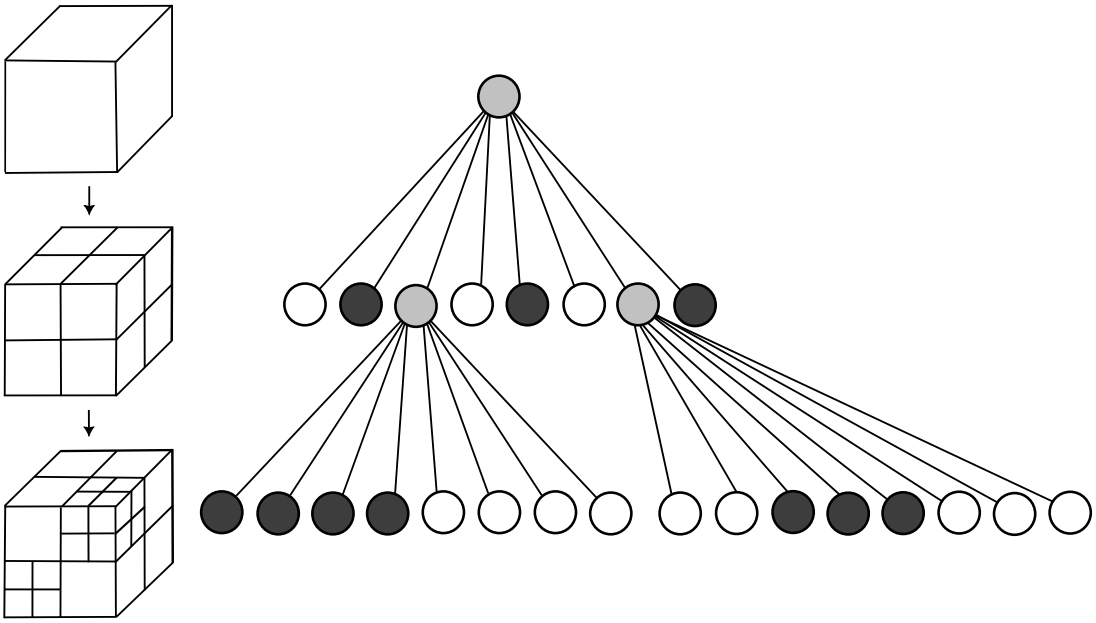


Figure 4.4: Octree

By assuming a uniform prior probability, where $P(n) = 0.5$, the equation can be reformulated using log-odds notation as:

$$\mathrm{L}(n \mid z_{1:t}) = \mathrm{L}(n \mid z_{1:t-1}) + \mathrm{L}(n \mid z_t) \tag{4.7}$$

where

$$\mathrm{L}(n) = \log\left[\frac{P(n)}{1 - P(n)}\right] \tag{4.8}$$

This updated formulation of the update rule enables faster updates by replacing multiplications with additions.

4.5 Results

In the results section, we present our monocular 3D reconstruction and mapping using a Raspberry Pi camera for image acquisition and a Lab PC for dense reconstruction. Our approach leverages the monocular camera's images to capture the scene, while the Lab PC performs computation to reconstruct and map the environment. For the program part the prior Pytorch Implementation of the avilable framework is modified for the GPU utilization which increased its fps by 15x the updated repo is available at `https://github.com/Avi241/DenseDepth`. By utilizing this setup, we achieve accurate and comprehensive reconstructions, enabling us to

capture intricate details and structures in the scene. Further this system is also setuped on Jetson Xavier NX for onboard execution.The following figures shows the construction of 3D Octomap of the Control Lab at IIT Dharwad.

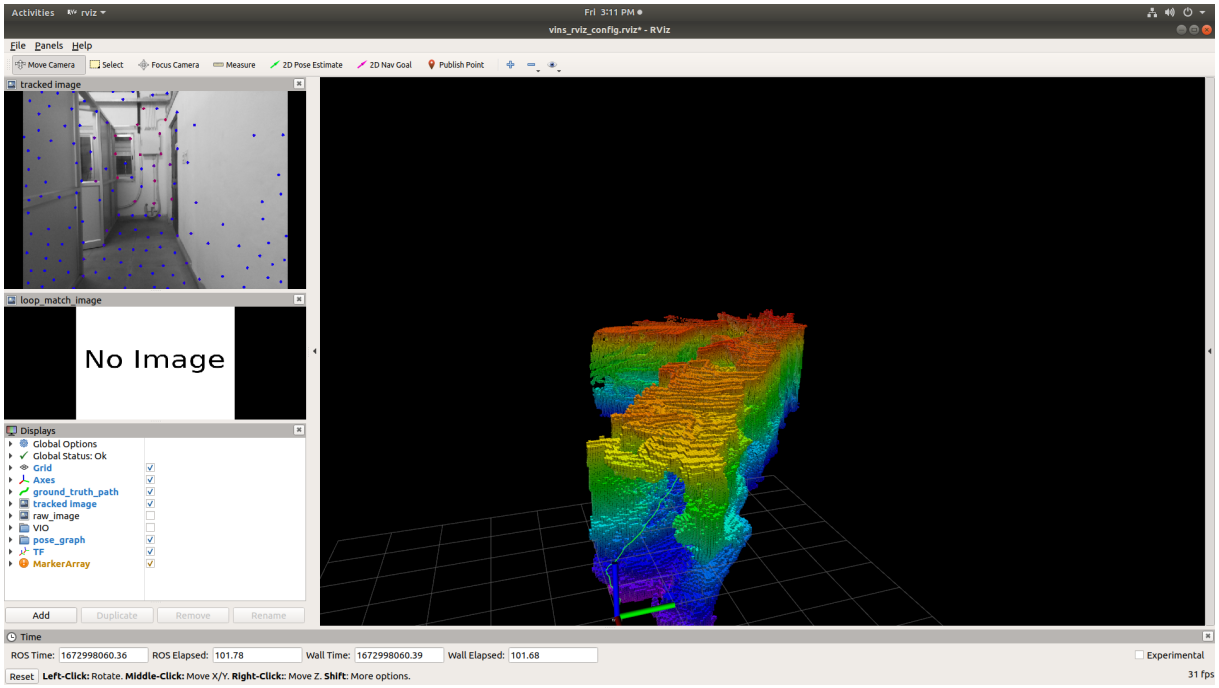


Figure 4.5: Octomap in Lab using monocular RGB camera

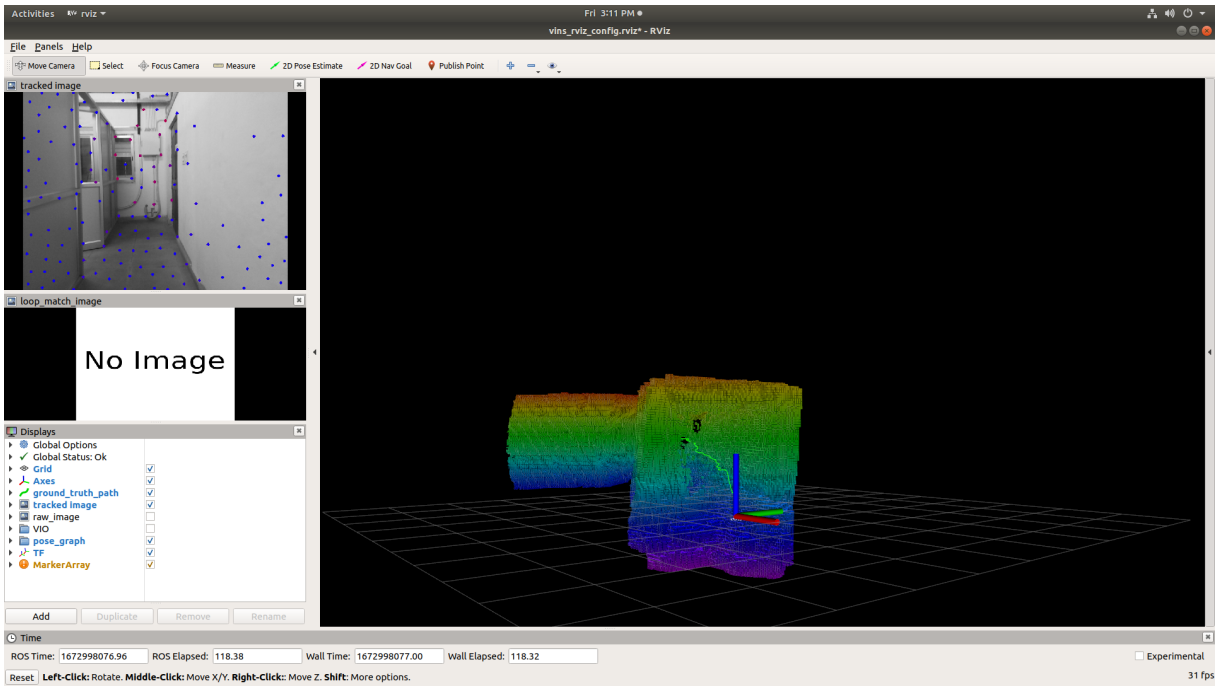


Figure 4.6: Octomap in Lab using monocular RGB camera

LAYER	OUTPUT	FUNCTION
INPUT	$480 \times 640 \times 3$	
CONV1	$240 \times 320 \times 64$	DenseNet CONV1
POOL1	$120 \times 160 \times 64$	DenseNet POOL1
POOL2	$60 \times 80 \times 128$	DenseNet POOL2
POOL3	$30 \times 40 \times 256$	DenseNet POOL3
...	...	...
CONV2	$15 \times 20 \times 1664$	Convolution 1×1 of DenseNet BLOCK4
UP1	$30 \times 40 \times 1664$	Upsample 2×2
CONCAT1	$30 \times 40 \times 1920$	Concatenate POOL3
UP1-CONVA	$30 \times 40 \times 832$	Convolution 3×3
UP1-CONVB	$30 \times 40 \times 832$	Convolution 3×3
UP2	$60 \times 80 \times 832$	Upsample 2×2
CONCAT2	$60 \times 80 \times 960$	Concatenate POOL2
UP2-CONVA	$60 \times 80 \times 416$	Convolution 3×3
UP2-CONVB	$60 \times 80 \times 416$	Convolution 3×3
UP3	$120 \times 160 \times 416$	Upsample 2×2
CONCAT3	$120 \times 160 \times 480$	Concatenate POOL1
UP3-CONVA	$120 \times 160 \times 208$	Convolution 3×3
UP3-CONVB	$120 \times 160 \times 208$	Convolution 3×3
UP4	$240 \times 320 \times 208$	Upsample 2×2
CONCAT3	$240 \times 320 \times 272$	Concatenate CONV1
UP2-CONVA	$240 \times 320 \times 104$	Convolution 3×3
UP2-CONVB	$240 \times 320 \times 104$	Convolution 3×3
CONV3	$240 \times 320 \times 1$	Convolution 3×3

Table 4.1: Network Layers [4]

Chapter 5

3D Motion Planning

3D motion planning refers to the process of generating a feasible and optimal path for a robot or an object in a three-dimensional space. The EGO framework, proposed by Zhou et al. in [5], is an gradient local planning approach used for motion planning which does not use Euclidean Signed Distance Function(ESDF). It aims to provide a smooth, collision-free, and dynamically feasible spline path. Earlier,for local optimal solutions, gradient-based planners use pre-built ESDF maps to assess gradient magnitude and direction.

In contrast, the EGO framework avoids the need for a pre-computed ESDF map by calculating the collision cost as the difference between the collision path and a guiding collision-free path. By eliminating the computation time required for ESDF maps, the overall processing time for local planning is significantly reduced. The trajectory is guided out of obstacles using an estimated gradient. Figure 5.1 illustrates a typical scenario in autonomous aeriAL navigation in which the drone targets to locally avoid collisions. During the navigation process, the trajectory is confined to a limited area within the ESDF updating range.

5.1 Collision Avoidance

Figure 5.1 and 5.2 shows the collision avoidance flow. Here, the decision variables are represented by control points, denoted as $\mathbf{Q}$, which are part of a $\mathbf{B}$-spline curve. Each control points $\mathbf{Q}$ contains unique information about its surroundings. Initially, a B-spline curve Φ is provided that satisfies terminal constraints but may not be collision-free. An optimization process is then initiated, where any colliding segments detected in each iteration are addressed by generating a collision-free path Γ. For each control point $\mathbf{Q_i}$ of the colliding segment, an anchor point p_{ij}

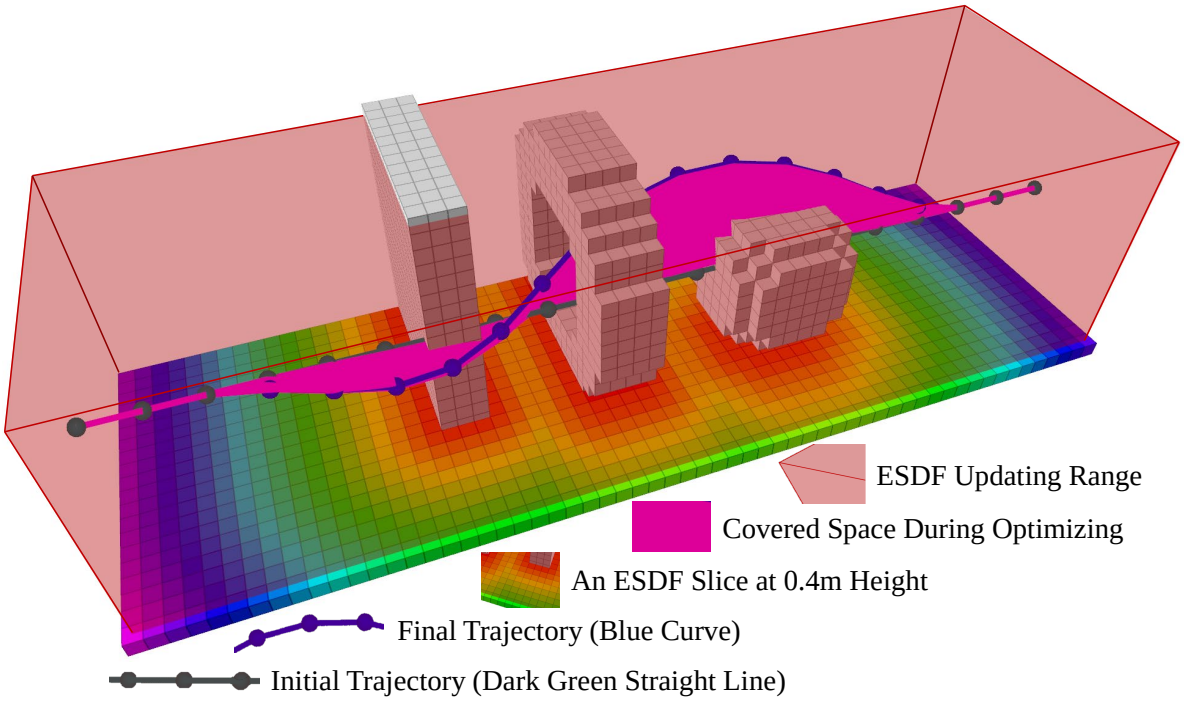
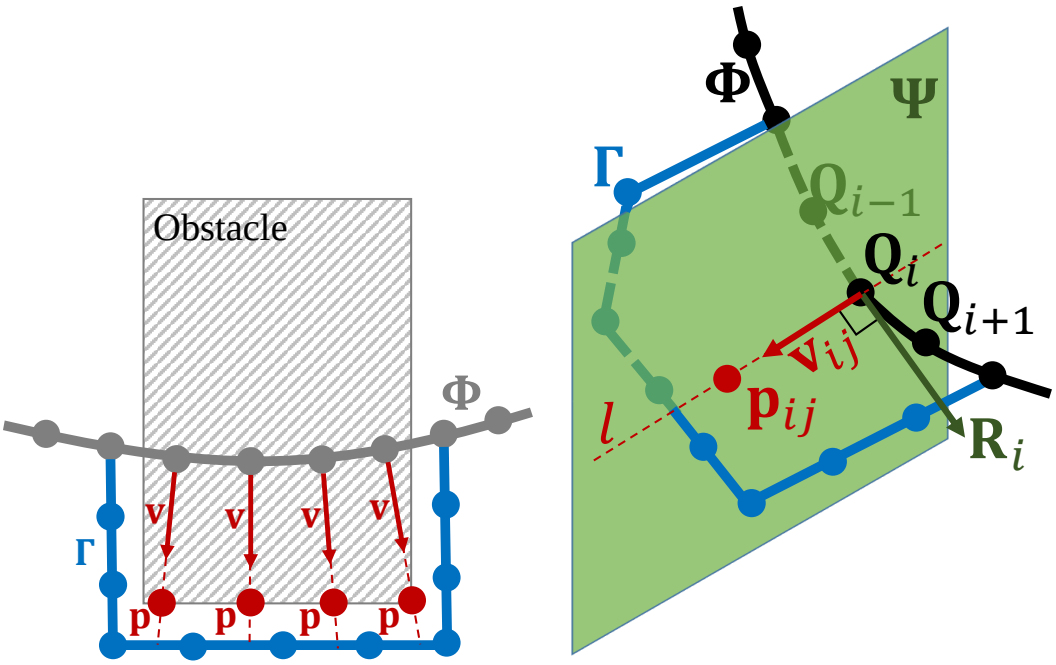


Figure 5.1: Trajectory by Ego Planner [5]

is assigned on the obstacle surface along with a corresponding repulsive direction vector v_{ij}, as depicted in Figure 5.2(a). The indices of the control points and p,v pairs are denoted by i and j, respectively, with each p,v pair only belonging to one control point. For simplicity, we will omit the subscript ij without any ambiguity. Figure 5.2(b) shows the procedure of generating p,v pair. The obstacle distance from $\mathbf{Q_i}$ to the j^{ij} obstacle is defined as

$$d_{ij} = (\mathbf{Q}_i - \mathbf{p}_{ij}) \cdot \mathbf{v}_{ij}$$

To avoid generating duplicate p,v pairs before the trajectory escapes from the current obstacle, a criterion is employed. This criterion determines whether an obstacle, where the control point Q_i is located, is considered newly discovered based on the condition that $d_{ij} > 0$ for all valid j. By applying this criterion, only the necessary obstacles that contribute to the final trajectory are taken into account during the optimization process, resulting in a significant reduction in operation time.



(a) p,v Pairs (b) p,v Generations

Figure 5.2: p,v pairs [5]

5.2 Simulation and Results

5.2.1 Proposed Approach

We propose a system that uses a monocular camera and an IMU to achieve navigation in GPS denied environment. The entire proposed system architecture of the autonomous aerial navigation system for quadcopters and the framework used for the specific task in the proposed approach is shown in Figure 5.3. The full pipeline is divided into five significant modules.

Camera-IMU calibration is performed to obtain the intrinsic and extrinsic parameters which are used by other modules. VINS-mono uses the raw IMU data and an RGB image from the quadcopter for the state estimation, which outputs the odometry of the quadcopter in 3D space. 3D reconstruction is performed using the monocular RGB image, and point clouds are generated with the help of known camera parameters. Using these point clouds and odometry data, an octomap creates a 3D map of the environment represented by a voxel which is further used by a path planner to generate a collision-free trajectory for safe navigation.

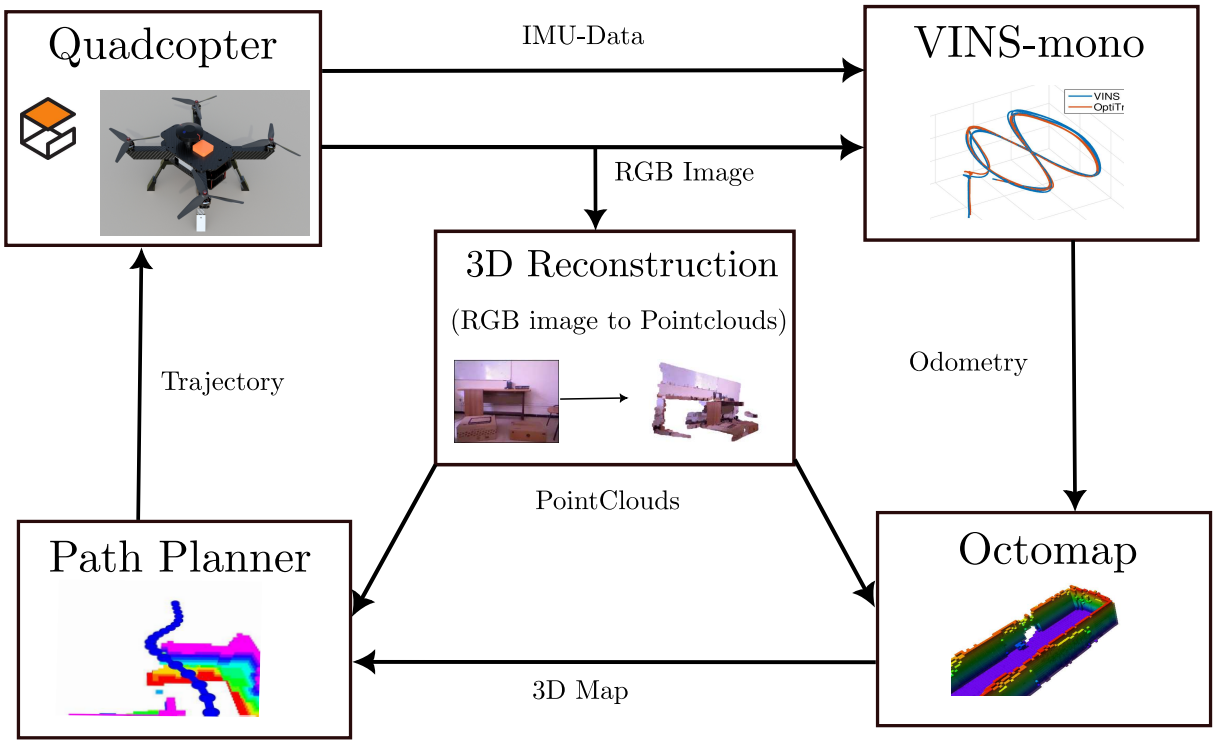


Figure 5.3: Full System Pipeline

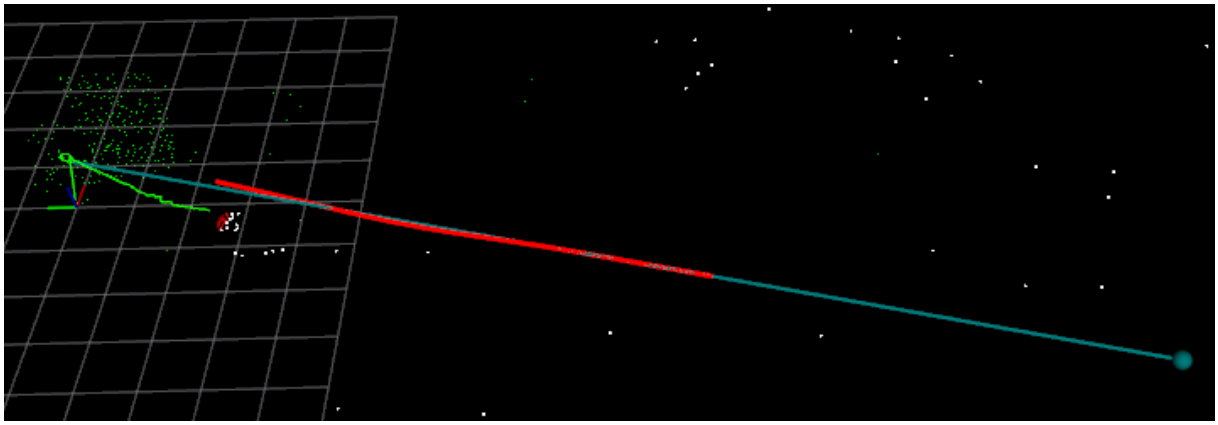
5.2.2 Results

We evaluate our system's performance in localization and planning by running navigation tests in indoor simulation environments. The simulation environments have a few obstacles and blockades that test the system's performance. The video of the simulation can be seen at `https://youtu.be/z69deIPDz2o`

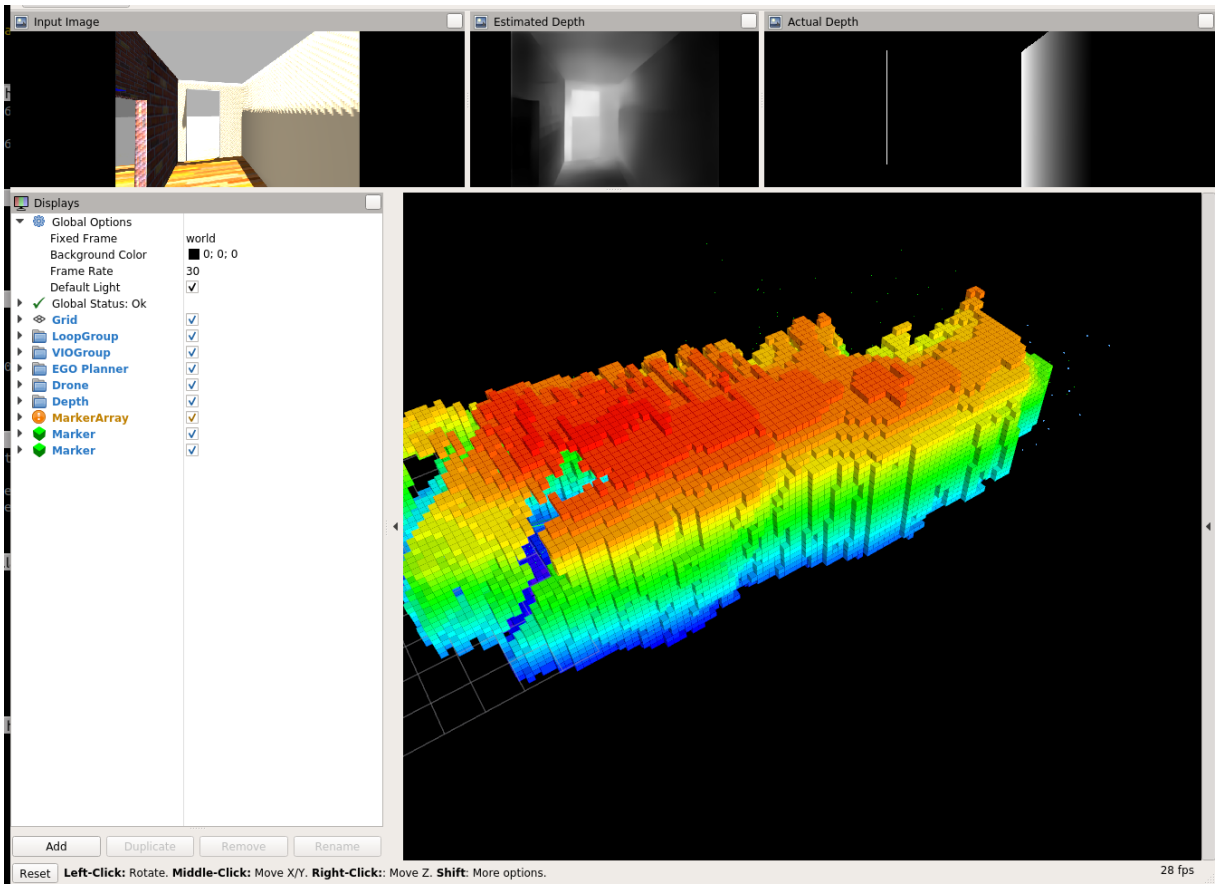
The system was able to navigate to the given target avoiding obstacles encountered accurately. The flight of the drone through the simulation environment in one such test is depicted by Figure 5.4. The figure shows the octomap generated from the mapping module and the trajectory of the planned path. The blue trajectory indicates the global path planning, the red trajectory indicates the local path planning and the green trajectory indicates the actual flight trajectory.

5.2.3 Computation Analysis

As the system is going to be running on an Edge Device, the computation analysis of all the modules were perfomed to ensure that each modules is in the limit of the computation of the device used.



(a) Ego-planner trajectory



(b) Octomap with estimated odometry and pointcloud

Figure 5.4: Simulation result of navigation

Device Specifications

The specifications of the Edge device use is as follows.

- 6-core NVIDIA Carmel ARMv8.2 64-bit CPU
- 384-core NVIDIA Volta™ GPU with 48 Tensor Cores

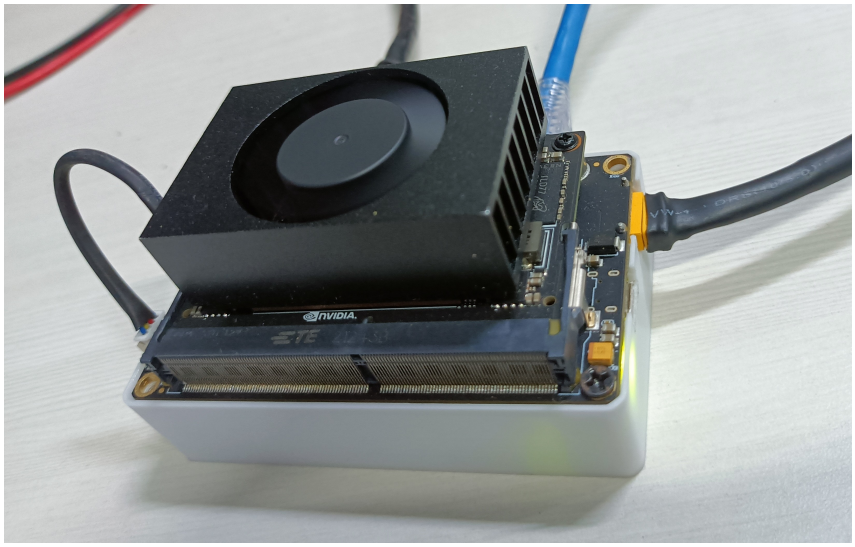


Figure 5.5: Nvidia Jetson Xavier Nx

- 8 GB 128-bit LPDDR4x 59.7GB/s RAM
- 21 TOPS AI performance. Weight- 350 gms

Node Graph

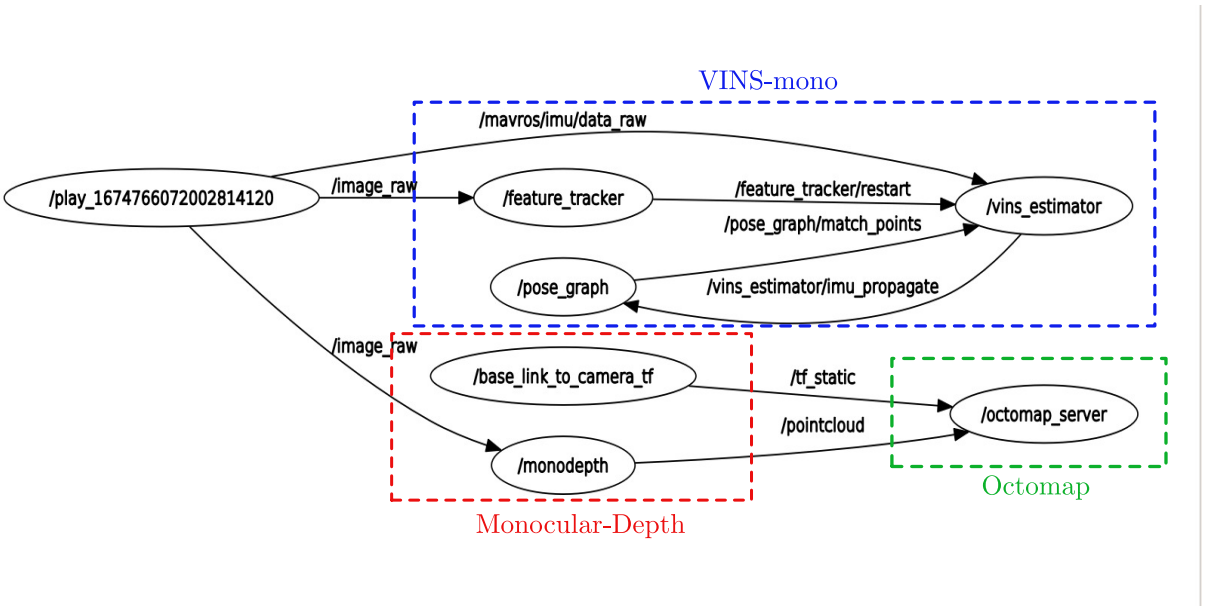


Figure 5.6: Node Graph

The computation graph in Figure 5.6 represents a system built on ROS (Robot Operating System) for autonomous navigation. It consists of multiple nodes working together.The feature tracker node identifies and tracks visual features in input image frames, allowing for

motion estimation and tracking. The VINS estimator node implements the VINS-mono algorithm, fusing visual and inertial measurements to estimate the camera's pose in 3D space. The pose graph node optimizes the estimated poses for global consistency, incorporating loop closures to enhance accuracy.The monodepth node estimates depth from monocular images, which is essential for mapping, navigation, and object recognition. Together, these nodes enable the VINS-mono system to perform visual odometry, pose estimation, and depth estimation, providing accurate localization and mapping capabilities using a monocular camera.

Computation Usage

The figure 5.7 represents the CPU usage of each node, while the figure 5.8 represents the memory usage of each node. It is worth noting that both CPU usage and memory usage are within the limits of the device being used.By monitoring CPU usage, we can assess the computational load of each node in the system. If the CPU usage remains within acceptable limits, it indicates that the nodes are efficiently utilizing the available processing power without overwhelming the system.Similarly, monitoring memory usage is essential to ensure that the nodes are not consuming excessive memory resources. When memory usage stays within the limits of the device, it suggests that the nodes are managing their memory requirements effectively, preventing potential memory-related issues such as crashes or slowdowns.Overall, the figures indicate that the CPU and memory usage of each node are well within the limits of the device being used, which is a positive sign for the system's performance and stability.

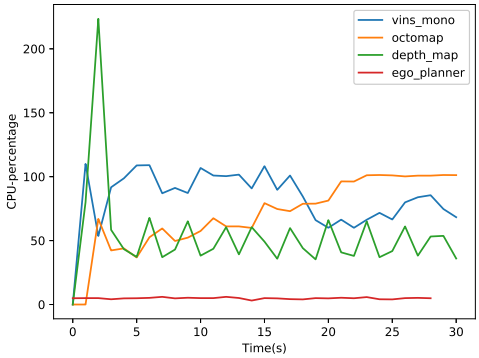


Figure 5.7: CPU Usage Analysis

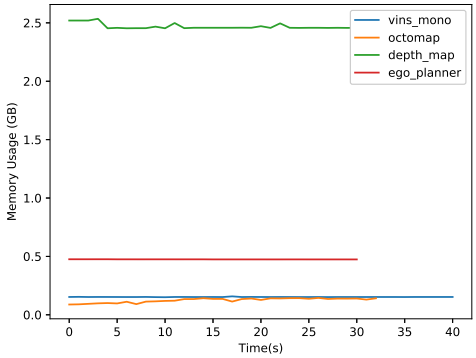


Figure 5.8: Memory Usage Analysis

Figure 5.9: Computation Usage

Publication

This work is submitted in the *62nd IEEE Conference on Decision and Control, Dec. 13-15, 2023, Singapore (Under Review)*

Chapter 6

Thermal Inertial Navigation

This chapter focuses on the deployment of visual odometry using an LWIR camera, emphasizing the autonomous navigation capability of aerial robots in environments heavily affected by obscurants like smoke and fog, where GPS signals are unavailable. To enable this capability, the proposed approach suggests fusing LWIR thermal camera data with time-synchronized IMU cues. This choice is justified by the validation that alternative solutions such as LiDAR or visible spectrum camera-based approaches often provide unreliable information in such environments. Visible spectrum camera images suffer from a white-out effect caused by light scattering through obscurants, while LiDAR data becomes less reliable, resulting in sparse and noisy point clouds. In contrast, the proposed approach demonstrates the robust performance of a well-calibrated and fused thermal navigation architecture, even in environments with dense obscurants like fog or smoke.

6.1 Evaluation of Navigation Sensors

The paper [6] presents a comprehensive study evaluating various navigation sensors used in robotics. Specifically, the study involved testing all available sensors within a smoke-filled box. Figure 6.1 provides a visualization of the working wavelengths of the different navigation sensors employed in robotics. It is worth noting that the size of the smoke particles ranged from 0.1 to 1.2 microns. The results indicate that sensors operating in the infrared (IR) range, specifically within the range of 750 to 1000 nm, exhibit high immunity to scattering caused by suspended smoke and soot particles. Consequently, the LWIR camera emerges as the most suitable choice for navigation in obscurant-filled environments based on its ability to effectively

navigate through such challenging conditions.

Instrument	Type	Operating wavelength/ frequency range
Hokuyo UTM-30LX	Single-echo LIDAR	0.905 μm
IBEO Lux	Multi-echo LIDAR	0.905 μm
SensComp 6500	Sonar	50 kHz, acoustic
MaxBotix 7092	Sonar	42 kHz, acoustic
HP Webcam	Camera	~0.4 μm to 0.7 μm, visual
Xbox Kinect™ Camera	Camera	~0.4 μm to 0.7 μm, visual
Xbox Kinect™ Depth Sensor	Depth sensor (IR laser projector and CMOS camera)	0.830 μm
FLIR SC655	Thermal IR camera	7.5 μm to 13.5 μm
FLIR Tau320	Thermal IR camera	8 μm to 14 μm
ATN Night Vision Gen 1 Monocular	Night vision	~0.4 μm to 0.8 μm
Endress Hauser Micropilot M Level Meter	Radar	26 GHz

Figure 6.1: Working frequency range of different sensors [6]

Figure 6.2 shows the result of the experiments from [6] with varying smoke density and temperature. As part of the self-evaluation process, a fire experiment was conducted in the Fire Research laboratory at IIT Dharwad. The experiment involved lighting a 0.1 m and 0.3 m pool fire, and the available sensor was tested under these conditions. Figure 6.3 presents the results of the experiment, highlighting the distortion observed in the 2D LiDAR when confronted with a 0.3 m pool fire. The figure visually illustrates the impact of the fire on the performance of the LiDAR sensor, indicating potential limitations when operating in such high-temperature and high-intensity fire scenarios.

6.2 Hardware Configuration

In order to conduct further experiments with the algorithm, a hardware setup comprising an IMU and a thermal camera has been developed at the Control Lab located at IIT Dharwad. Figure 6.4 provides a visual representation of the hardware utilized in the experiment. A hand-held module specifically designed for Thermal Inertial Odometry (TIO) has been created, incorporating the LWIR FLIR Tau2 as the thermal camera, and a low-cost flight controller to capture IMU data. For high-level tasks, a Raspberry Pi-4 with 4GB RAM has been employed as the

Instrument	Dense smoke, low temperature	Light smoke, high temperature
Electromagnetic instruments		
LIDAR instruments	Attenuation at 4 m visibility; failure at 1 m visibility	No effect
Visual cameras	Attenuation at 8 m visibility; failure at 1 m visibility	No effect
Kinect™ depth sensor	Poor results even with >8 m Visibility (combination of particle blocking and sensor being flooded by light from fire)	Sensor flooded by light from fire during whole test
Night vision	Failure at about 4 m visibility	Sensor flooded by light from fire during whole test
Thermal cameras	No effect	No effect
Radar	No effect	No effect
Other		
Sonar	Some attenuation with temperature change	Attenuation with temperature change

Figure 6.2: Evaluation of Navigation Sensor [6]

Single Board Computer (SBC) within the setup. This hardware configuration enables the execution of the desired algorithm and facilitates the exploration of the TIO capabilities for future applications and research.

The Hardware specification of the used modules are as follow:

1. **Raspberry Pi-4 :**
    - 4 GB of DDR4 RAM
    - Quad core Cortex-A72 (ARM v8) 64-bit SoC
    - Power Supply as 5V DC via USB-C connector
    - 1.5 GHz Clock frequency
2. **Cube Orange Flight Controller :**
    - 32bit ARM® STM32H753 Cortex®-M7.
    - 400 Mhz/1 MB RAM/2 MB Flash

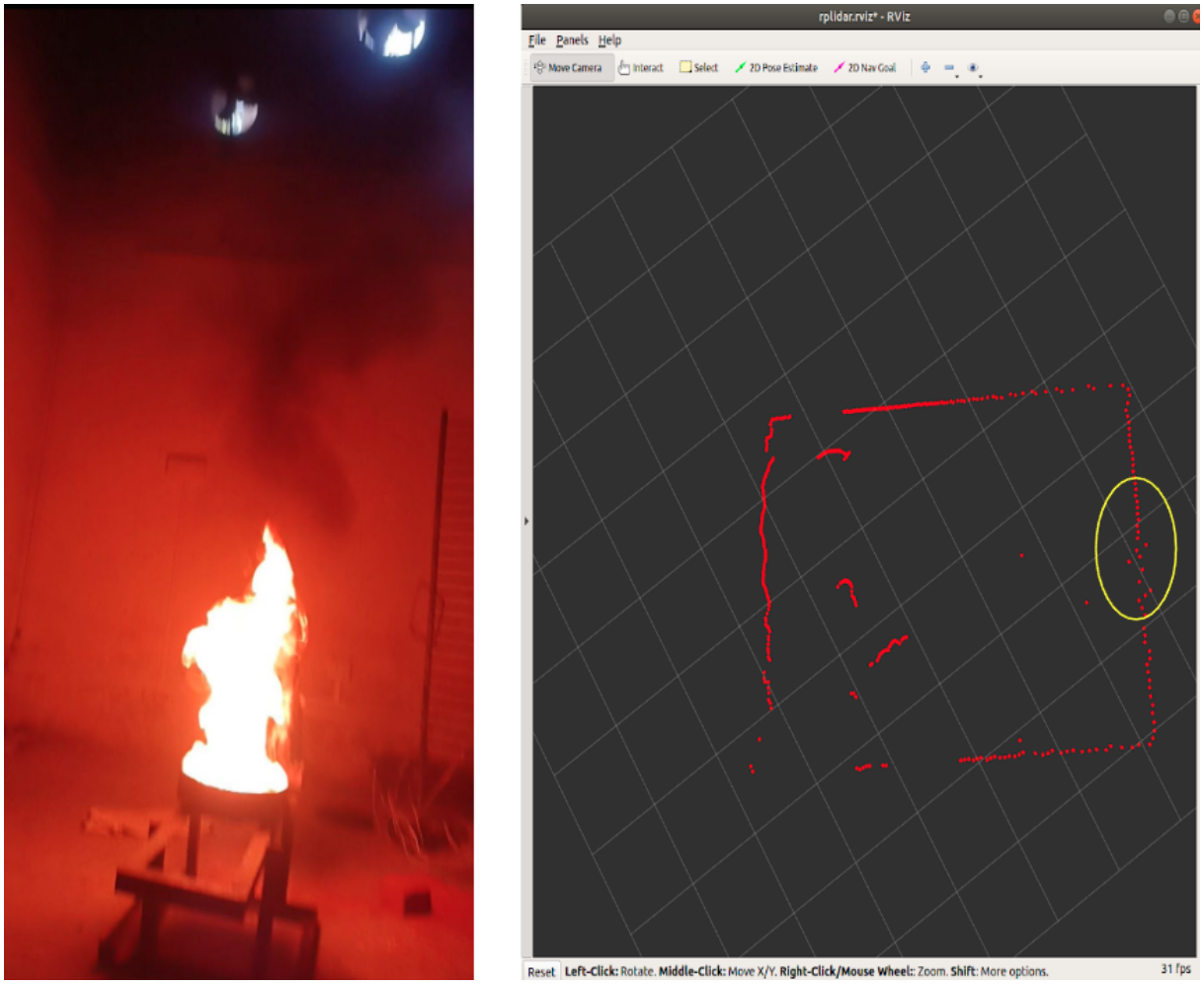


Figure 6.3: 0.3 m Pool Fire and 2D Lidar results

- 32 bit STM32F103 failsafe co-processor
- ICM 20649 integrated accelerometer / gyro, MS5611 barometer
- InvenSense ICM20602 IMU,ICM20948 IMU/MAG, MS5611 barometer

3. **LWIR FLIR Tau2**

- 17 μm pixel size
- Supports 30/60 Hz (NTSC) 25/50 Hz (PAL)
- Temperature range : -25 °C to +100 °C
- Resolution : 640 x 480 (NTSC); 640 x 512 (PAL)

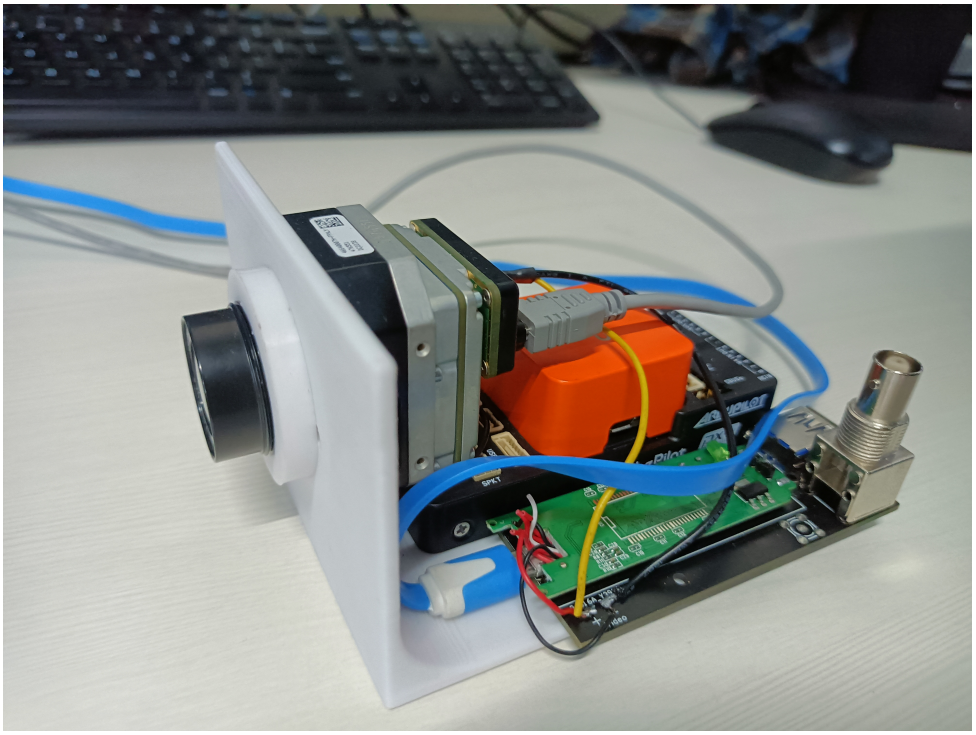


Figure 6.4: Thermal Inertial Odometry Module

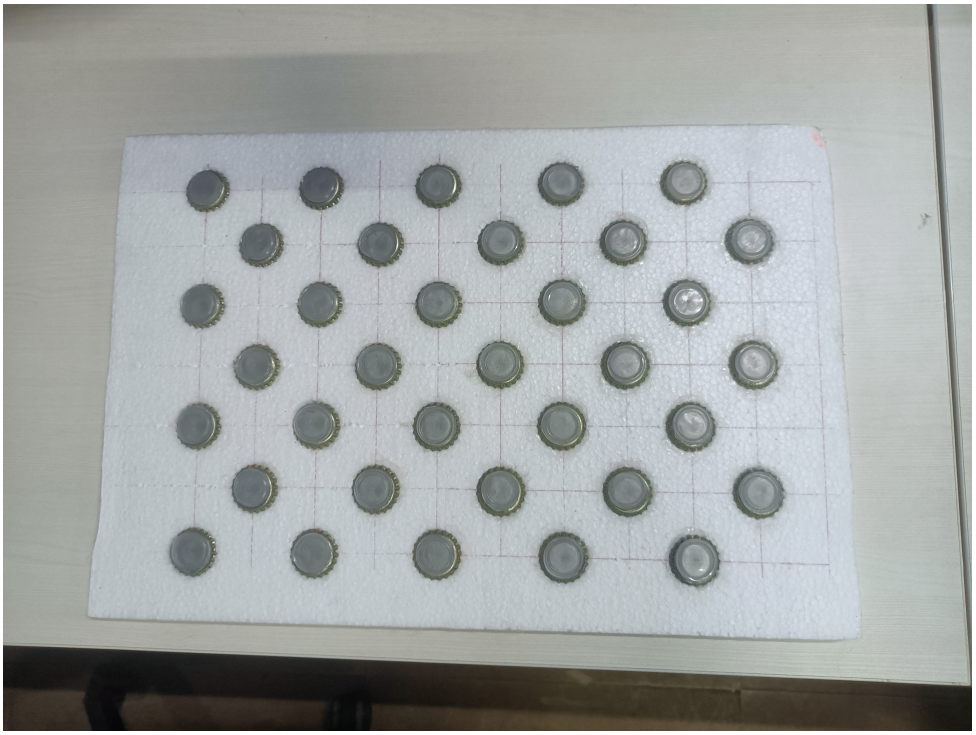


Figure 6.5: Calibration Board with Metal Circular grids

Figure 6.6: Calibration with Kalibr framework

6.3 Thermal Camera Calibration

In our research, we designed and fabricated an asymmetric circle calibration board using two different materials: metal discs and a foam board. These materials were chosen due to their distinct emissivity and thermal radiation properties. Figure 6.5 depicts the physical layout of the calibration board.

To calibrate the intrinsic parameters of the camera, we employed the Kalibr Multisensor framework [24, 25] with the use of an Aprilgrid. Figure 6.6 showcases the utilization of this framework during the calibration process. The resulting calibration for the given setup yielded the following calibrated intrinsic parameters.

Furthermore, the reprojection error of the calibrated camera is visualized in Figure 6.7, providing insights into the accuracy and performance of the calibration process.

The Calibrated values of the hardware module are as follows:

- Intrinsic Camera Matrix $\mathbf{K} = \begin{bmatrix} 414 & 0 & 340 \\ 0 & 422 & 280 \\ 0 & 0 & 1 \end{bmatrix}$

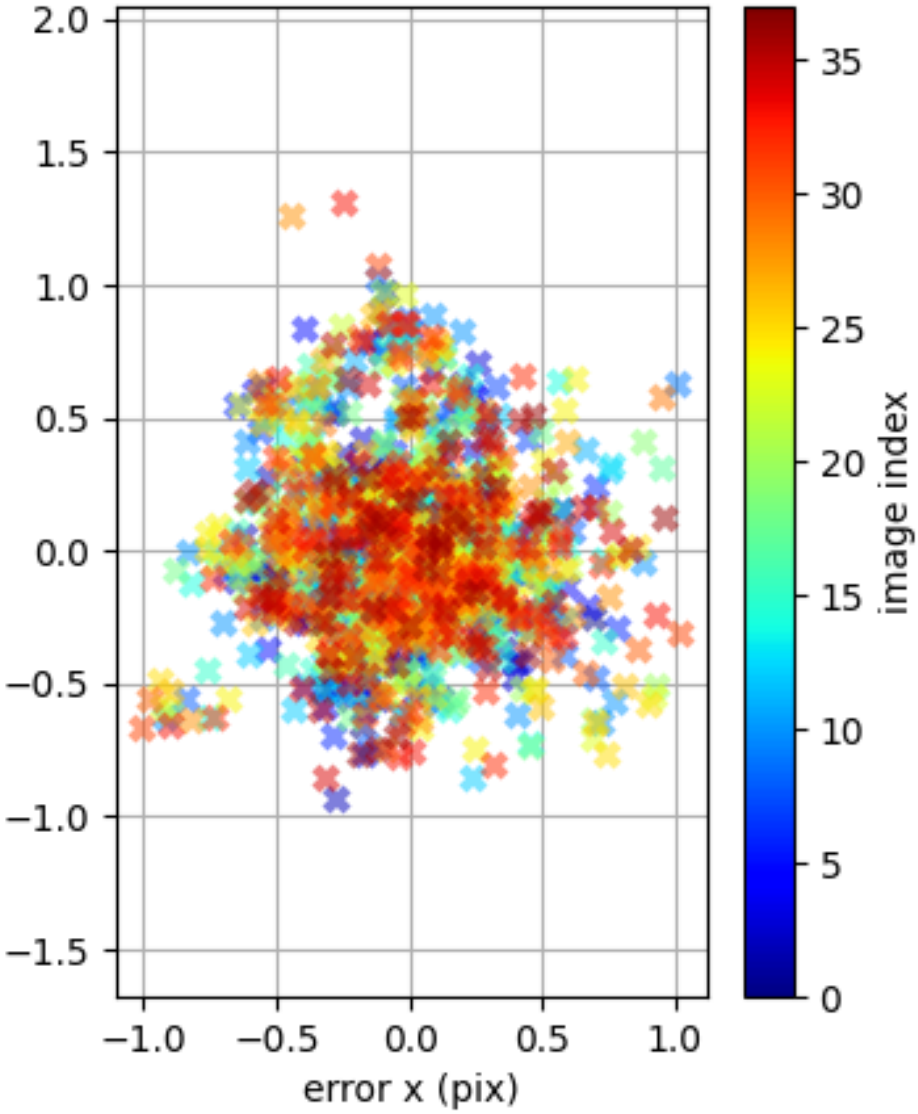


Figure 6.7: Thermal Camera Reprojection Error

- Distortion parameters $(\mathbf{k1}, \mathbf{k2}, \mathbf{p1}, \mathbf{p2}) = [-0.348,\ 0.093,\ -0.011,\ -0.008]$

6.4 Feature detection and Tracking

In thermal inertial localization, Long Wave Infrared Cameras (LWIR) are utilized instead of visual cameras, providing radiometric information about the observed environments. LWIR cameras prove to be superior choices over visual cameras or LiDAR for navigation in environments affected by obscurants. Leveraging the existing hardware, feature detection and tracking techniques have been implemented specifically on the thermal images. This approach enables the extraction of relevant features and facilitates accurate pose estimation in thermal-based lo-

calization systems.

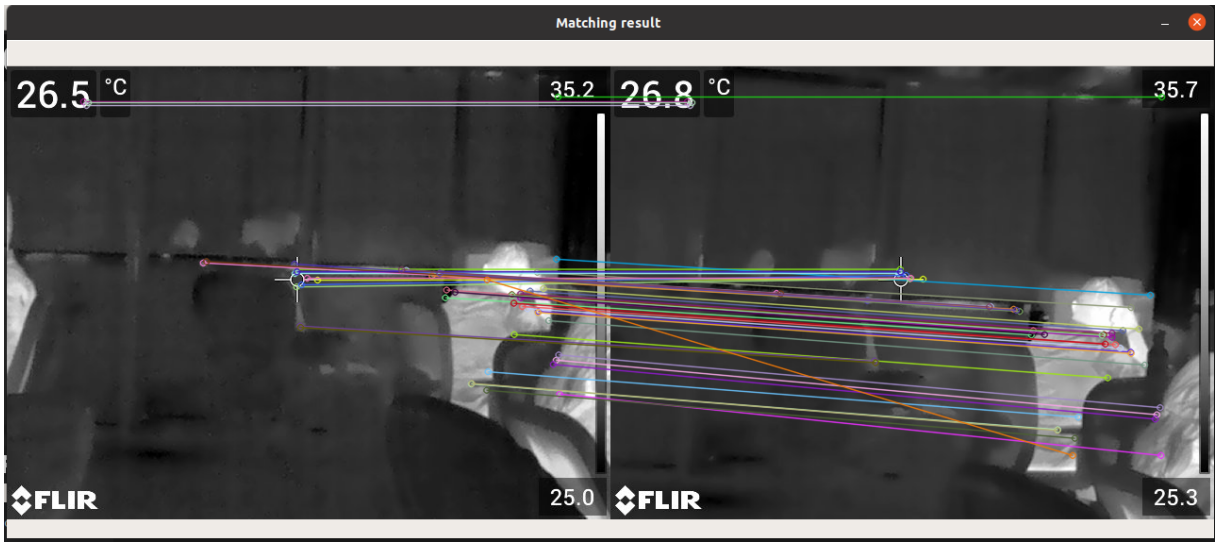


Figure 6.8: ORB feature detector and matching on thermal Images

Figure 6.8 illustrates the practical implementation of the ORB feature detector and matching technique on a gathered thermal image. This demonstration provides evidence that detecting and tracking features within thermal images is indeed achievable, thereby establishing the viability of employing such images for pose estimation. Figure 6.9 showcases the outcomes of the KLT tracker applied to the Thermal Images. For a comprehensive visual representation of the entire process, a complete video recording is accessible at `https://youtu.be/5PAQ3zsI-K4`.

6.5 Visual Odometry using Thermal Camera

By employing the identical approach as that of monocular visual odometry, it is plausible to conduct pose estimation utilizing Thermal Images. Regrettably, the received LWIR camera Tau 2 incurred internal damage, necessitating its return to the manufacturer for repairs. Consequently, the Thermal Odometry was demonstrated on an openly accessible dataset that comprised thermal images of a scene with a drone navigating within an environment. The ensuing steps outline the methodology employed in Thermal Odometry:

- Capture images: I_t and I_{t+1}
- Undistort the captured images.

Figure 6.9: ORB feature detector and matching

- Use SURF detector algorithm to detect features in image I_t.
- Track these features using an optical flow methodology, remove points that fall out of frame or are not visible in I_{t+1}.
- Trigger a new detection of points if the number of tracked points falls behind a threshold. Set to 100 in this implementation.
- Nister's 5-point algorithm with RANSAC to find the essential matrix.
- Estimate R, t from the essential matrix.

Figure 6.11 depicts the utilization of Thermal Images for the implementation of Visual Odometry. The presented figure illustrates the feasibility of performing pose estimation through the employment of a thermal Camera. Figure 6.10 showcases the tracked position of the camera within a three-dimensional spatial context. For a comprehensive view of the entire process, a video recording of the implementation can be accessed at `https://youtu.be/viwudOpqGm0`.

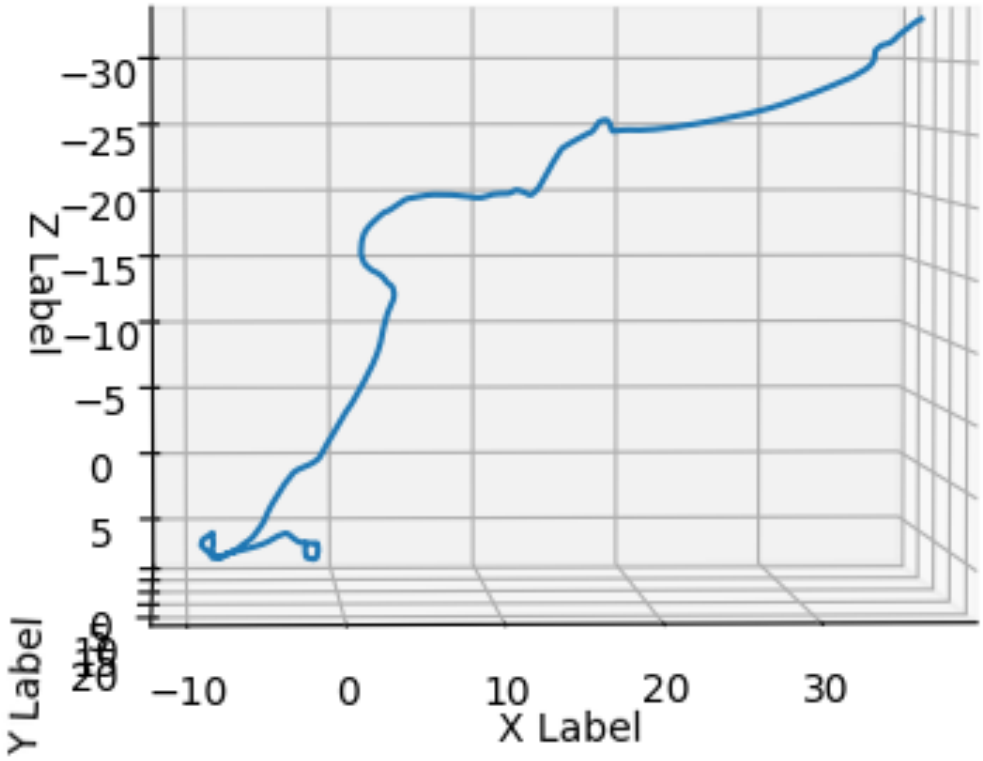


Figure 6.10: Thermal Odometry Plot

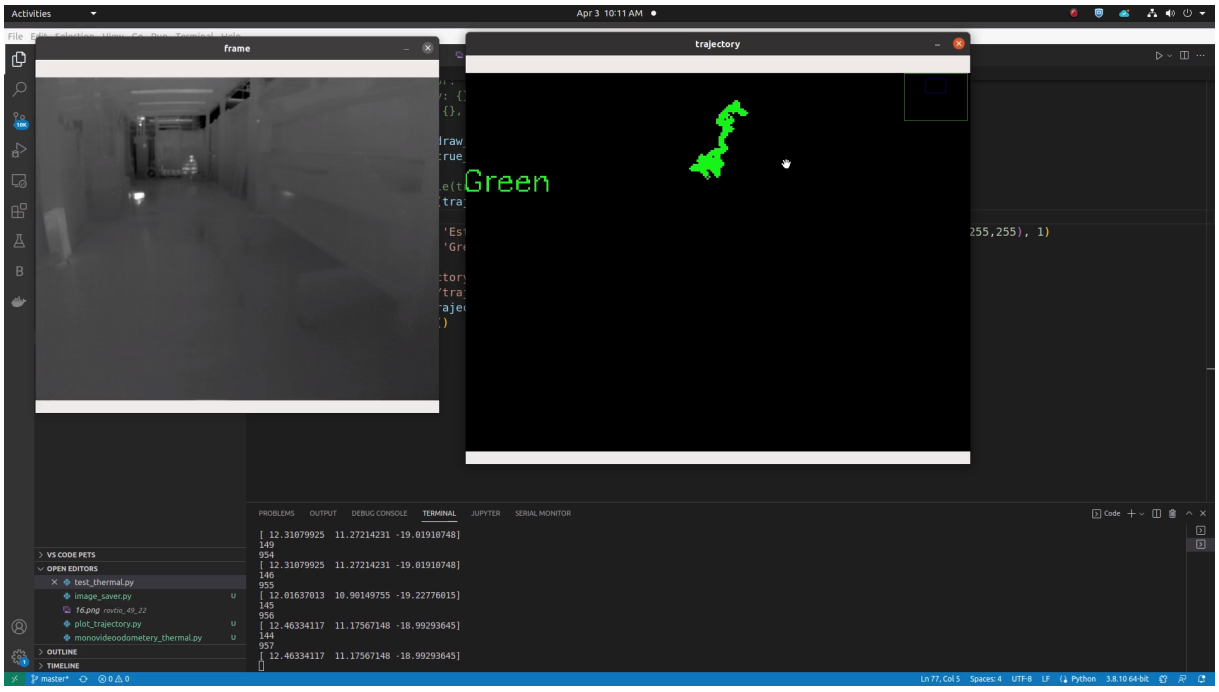


Figure 6.11: Thermal Odometry

Chapter 7

Drone Design and Development

In this chapter, we delve into the design and development of the drone used in our project. We begin by presenting the CAD model of the drone and showcasing the successfully fabricated drone using CNC milling. We then discuss the materials used in the fabrication and insulation of the drone, highlighting the use of carbon fiber for the drone body due to its exceptional strength-to-weight ratio and various desirable properties. We also explore the use of aramid fiber for insulation, emphasizing its high temperature resistance and low thermal conductivity. The electronic components onboard the drone, including the flight controller, ESC, BLDC motors, and radio receiver, are detailed, along with their specifications. Furthermore, we describe the experiments performed in a smoke-filled environment, presenting thermal measurements and flight test results to demonstrate the drone's performance in such conditions. This chapter provides valuable insights into the drone's design, materials, and performance, laying the foundation for the subsequent chapters' discussions and analysis.

7.1 Design and Fabrication

The CAD model of the drone, as shown in Figure 7.1, was designed using Fusion 360, a powerful and widely used CAD design software. Fusion 360 offers a range of tools and features that enable engineers to create intricate and precise 3D models.

The design of the drone was carefully planned to minimize accessories and ensure a lightweight structure. This approach was adopted to enhance the drone's maneuverability, agility, and overall flight performance. By reducing unnecessary components and optimizing the design for weight reduction, we aimed to maximize the drone's efficiency and flight time.

Furthermore, the decision to fabricate the drone using CNC milling was driven by the desire to achieve high precision and accuracy in the manufacturing process. CNC milling allows for the creation of complex geometries with tight tolerances, ensuring the faithful reproduction of the CAD design. However, it's important to note that while CNC milling offers excellent precision, it may not be suitable for large-scale mass production due to its time-consuming nature.

By leveraging the capabilities of Fusion 360 and employing a design strategy focused on minimal accessories and lightweight construction, we aimed to develop a drone that excels in performance, efficiency, and agility.

Figure 7.1: CAD Model of the drone

Figure 7.2: Fabricated drone

7.2 Materials Used in Fabrication and Insulation of the Drone

7.2.1 Carbon Fiber for Drone Fabrication

Carbon fiber was used for fabricating the drone body due to its high strength-to-weight ratio, making it perfect for drone fabrication. The crystal alignment in the structure matrix of carbon fiber gives it a high strength-to-volume ratio as well. Some other properties that make carbon fiber exceptional for use in the UAV industry are as follows:

- High specific toughness
- Good vibration damping
- High dimensional stability
- Low coefficient of thermal expansion

- Low abrasion
- Fatigue resistance
- Chemical inertness
- High corrosion resistance

Throughout the experimentation on flight dynamics, the drone encountered multiple crashes. However, despite these incidents, both the electronic components and the quadcopter structure as a whole demonstrated exceptional resilience and durability. This remarkable level of robustness fulfilled the stringent requirement for the drone's ability to withstand challenging conditions and unexpected impacts.

7.2.2 Aramid Fiber used for Insulating the Drone and Electronic Components from High-Temperature Smoke

The insulation on the drone had the following requirements:

- High temperature resistance or low thermal conductivity - for minimum heat flux penetration
- Less density - due to payload restrictions on the quadcopter
- Less thickness - for better drone aerodynamics
- Easy removal after every flight

Various materials were tested, including Intumescent coatings, Polyurethane foam, Aramid fiber, and Aluminized glass fiber, under a cone calorimeter setup and on the drone. It was observed that Aramid fiber and Aluminized glass fiber were the only materials that met all the above-mentioned requirements simultaneously. Although Aluminized glass fiber is not currently being used due to some design-related hurdles, further research is being conducted as it is also a viable candidate for providing insulation. Some properties that make Aramid fiber exceptional are mentioned in Table 7.2.2.

Design iterations were performed on the insulation part, and they are still ongoing to achieve a design that minimizes interference with the flight dynamics.

Properties	Magnitude
Density	0.48 kg/m^3
Thickness	1.7 mm
Maximum sustainable temperature	300°C

Table 7.1: Aramid fiber properties

Figure 7.3: CAD Model of the drone covered in Aramid fiber

Figure 7.4: Actual drone with the insulation sleeve on

7.3 Electronic Components Onboard the Drone

Table 7.3 presents the specifications of the components used in the drone, including the flight controller, ESC, BLDC motors, radio receiver, carbon fiber plates, carbon fiber beams, and propellers.

7.4 Experiments performed in Smoke

The drone was completely manufactured using carbon fiber parts that can withstand temperatures up to 700°C in our institute's CNC milling machine. The drone was tested by flying it manually in a smoke environment, and the following measurements were obtained:

The graph in Figure 7.5 represents the maximum temperature recorded anywhere on the drone's body at a given time. As shown, the hottest parts of the drone were its motors and the electronic speed controller (mounted below the top carbon fiber plate).

Figure 7.6 shows a thermal image of the drone in-flight, depicting the high-temperature zones on the UAV.

S.no.	Item	Specifications
1	Flight Controller	PixHawk Cube Orange
		Input Voltage - 3.3/5V
2	ESC	T-Motor VELOX V50A
		Input voltage - 10V to 27V
		Peak output current - 55A
3	BLDC motors	R2207 2207 Brushless Motor
		2450 kV
4	Radio receiver	FLY SKY FS IA6B
		Operating Frequency - 2.4GHz
		No. of channels - 6
		Radio range ¿ 500m
5	Carbon Fiber plates	Carbon fiber Sheet plate
		300mm x 200mm x 2mm
6	Carbon Fiber beams	Square carbon fiber Tube
		20mm(OD) x 18mm(ID) x 500mm(L)
7	Propellers	Injection molded Propellers
		6x4.5

Table 7.2: Specifications of the components used in the drone

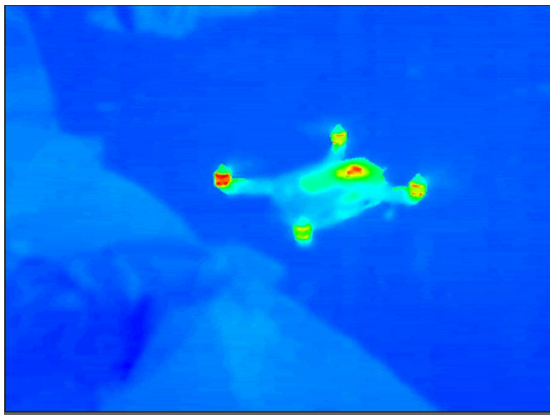


Figure 7.5: Peak temperature on the drone as recorded by LWIR thermal camera

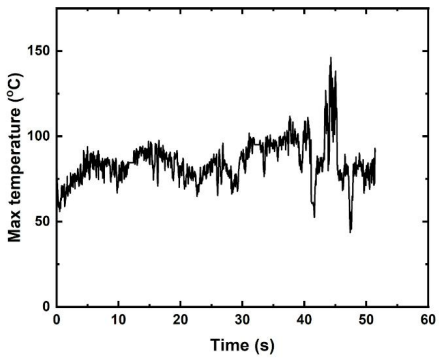


Figure 7.6: In-flight drone thermal image depicting the high-temperature zones on the UAV

It was observed from this test that after flying for over 5 minutes inside a smoke-filled environment, the maximum temperature reached by any part of the drone was approximately 91°C. The hottest components onboard were the electronic speed controller (ESC) and the BLDC motors.

Table 7.4 provides general information about the test performed, including the size of the room, size of the fire pool, fuel used, emissivity value set in the IR camera, and total flight time in the smoke environment.

Size of the room	12m x 13m x 15m
Size of the fire pool	0.3m
Fuel used for experiment	Diesel
Emissivity value set in IR camera	0.5
Total time flew in smoke	378 s

Table 7.3: General information about the test performed

In order to assess the drone's performance we conducted a flight test in Althold mode while monitoring the readings of the barometer in the UAV logs. The flight was conducted with the drone equipped with the necessary smoke insulation measures. Figure 7.7 presents the UAV log data obtained during the flight test. Upon analyzing the logs, we observed that there were no noticeable anomalies or abnormalities in the barometer readings. This indicates that the drone's barometer functioned properly and remained unaffected by the presence of smoke during the flight. These findings suggest that the drone is capable of flying in smoke-filled environments without significant operational issues. The full video is available at `https://youtu.be/Cp9LwxokHGk`

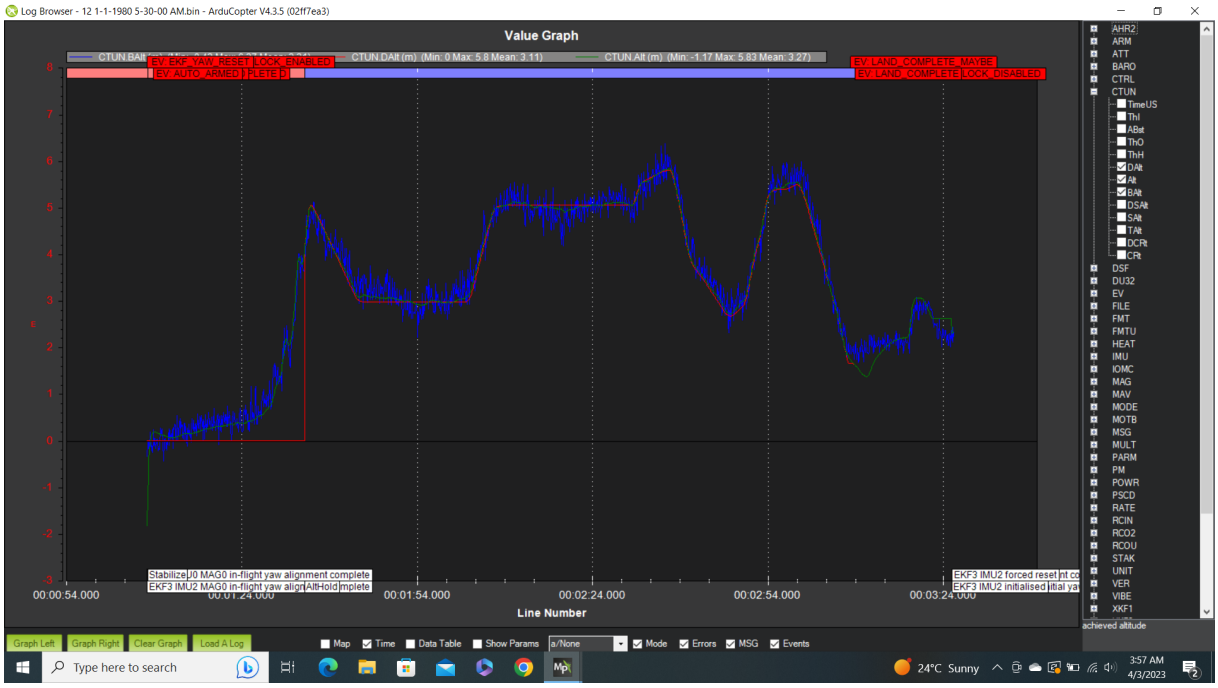


Figure 7.7: UAV logs indicating the drone's performance in smoke-filled environment

Chapter 8

Conclusion and Future Work

This chapter provides a concise summary of the work described in this thesis and discusses potential avenues for future research.

8.1 Conclusion Remarks

The primary objective of this thesis was to address the challenge of autonomous aerial navigation in GPS-denied indoor environments, with a particular focus on obscurant-filled conditions. The developed system pipeline for autonomous navigation in indoor settings utilized a monocular camera and an IMU. A Visual-Inertial Navigation approach was employed for quadcopter localization, incorporating a neural network-based 3D depth estimation and mapping technique, followed by motion planning. For obscurant-filled environments, an LWIR camera resistant to scattering from smoke and dust particles was utilized, enabling pose estimation through thermal imaging. The presented simulation results validate the effectiveness of the proposed approaches.

Furthermore, the work on autonomous navigation using a monocular RGB camera has been published as an open-source ROS package under the MIT license, accessible at `https://github.com/Avi241/movinav`. Additionally, a Docker image containing all the pre-built programs has been developed to facilitate easy setup of the system.

8.2 Future Scopes

While the presented work covers the full system pipeline for visual camera-based navigation and includes partial progress in thermal camera-based navigation, there are several future directions

worth exploring, including:

- Implementation of the developed pipeline on a real drone to evaluate its performance in practical scenarios.
- Extension of the thermal navigation component to encompass the complete system pipeline, enabling robust navigation in obscurant-filled environments.
- Comprehensive testing and validation of the system in various obscurant-filled conditions to assess its reliability and effectiveness.

These future endeavors have the potential to further enhance the capabilities of autonomous aerial navigation, particularly in challenging indoor environments.

Appendix A

MASCOT

A.1 Introduction

The exploration of any environment can be boosted by deploying Multiple Agents in the Environment which can coordinate and explore the environment in the least amount of time and with higher accuracy. To deploy this kind of control, work on distributed control simulation is done, which is described in this chapter.

Distributed control of multi-agent systems has been a very active area of research over the last couple of decades, starting from the asymptotic consensus laws proposed in [47], various distributed control laws for finite-time consensus [48], formation control [49], trajectory tracking [50], applications like survey, search and rescue [51], non-cooperative tasks like pursuit-evasion [52] etc. The primary reason for the popularity of multi-agent systems is that multi-agent systems are much more capable than individual agents and can achieve complex objectives using simple agents. Most of such control algorithms are designed assuming simple dynamics of agents, for example, single integrator, double integrator [53], non-holonomic Dubin's vehicle model, etc. These agent models are popular mainly because the dynamics of various robots and autonomous vehicles, under a few assumptions, may be simplified to such models.

Extensive simulations are necessary for appropriate tuning of the controllers before actual deployment on physical systems to ensure that the systems remain stable and the controller will work as desired. A good testing platform with a simple user interface can address this problem. MASCOT is designed to be such a testing platform. In the current version, MASCOT uses quadcopters as agents to simulate multi-agent systems. The lower-level controllers of the quadcopters are tuned so that the user can consider the agents to have double integrator

dynamics in the primary directions and test the control algorithms. By adjusting the control parameters of the lower-level controllers, practical scenarios like relaxing the assumptions may be created.

There are few MATLAB-based quadcopter simulators that are either for the single quadcopter or lack the environmental dynamics and visuals [54, 55]. MATLAB provides a few toolboxes which help for testing the algorithms but there are limitations of MATLAB such as inter-process communications, physical dynamics, code porting from simulation to hardware, etc. Some authors have demonstrated the control algorithms using realistic simulation environments and /or hardware test-beds. [56, 57]. However, these efforts are limited to specific control algorithms and are not easily accessible to other researchers. Moreover, such hardware testbeds come with physical limitations such as a number of agents, space, cost, as well as the potential risk of damage to the vehicles and the surroundings.

MASCOT is developed using open-source software, the Robot Operating System (ROS) [58] and Gazebo. Gazebo with its ODE (Open Dynamics Engines) provides a simulation environment with various real-world physical dynamics such as collision, gravity, mass, inertia, wind effects, and many more which are very tedious to configure in MATLAB. Moreover, ROS supports low-level drivers and programming in C++, making it very convenient to directly deploy the simulation codes on physical systems, which are not straightforward in MATLAB. ROS also provides a rich set of open-source packages for different applications which helps researchers to modify and use the code as per their needs. MASCOT retains all of the advantages of ROS while simplifying the user interface by providing simple text files for configuring the agents and control laws for simulations. We have also integrated Novint Falcon Haptic controller for the Human-in-loop functionality. This platform also supports hybrid simulation where crazyflie with an optitrack tracking system is used as an hardware agents and AR-parrot drone running in Gazebo are the virtual agents. The communication between the real and virtual agents are handled on ROS network. While this chapter describes the simulations of multi-agent systems with double integrator agents, simulating systems with single integrator agents, non-holonomic agents and heterogeneous systems are equally easy on MASCOT.

The rest of the chapter is organized as follows. In Section A.2, we discuss the quadcopter dynamics. Section A.3 describes the structure and features of the designed simulation testbed along with the tools used in the development. In Section A.4, various use cases of the testbed are discussed with the simulations carried out using MASCOT.

This work is submitted for publication in "The Eighth Indian Control Conference 2022" titles as as "ROS-Based **M**ulti-**A**gent **S**ystems **CO**ntrol Simulation **T**estbed (MASCOT)"

A.2 Preliminaries

In this section, first, we describe the preliminaries of the dynamics of a quadcopter and review how a quadcopter can be represented using double integrator dynamics. A few preliminaries used in distributed control of multi-agent systems are also discussed.

A.2.1 Frame of References

Figure A.1 shows the standard definitions [59] of the reference frames used in this simulation. $\{E\}$ is the *Earth reference frame* having its z-axis facing upward in the space. The body coordinate frame, $\{B\}$ is attached to the center of the quadcopter having its z-axis facing downwards in the space. $\{B'\}$ reference frame is attached to the body having its origin same as $\{B\}$ and $x^{B'}y^{B'}$ plane is parallel to $x^{E}y^{E}$.

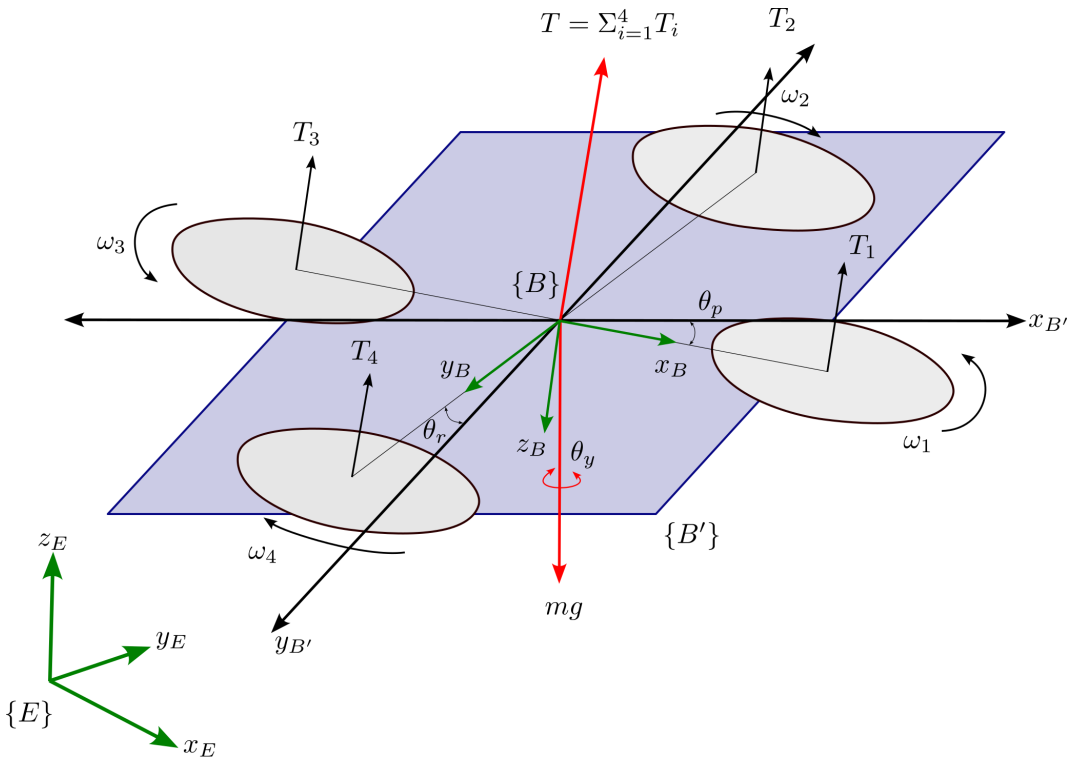


Figure A.1: Quadcopter reference frames setup

The angles θ_p (pitch), θ_r (roll), and θ_y (yaw) represents the rotation of the body along x,

y, and z-axis of the reference frame respectively. As $x^{B'}y^{B'}$ plane is parallel to $x^E y^E$ thus $\{B'\}$ can only rotate along the z-axis with angle θ_p. The change in orientation between $\{B'\}$ and $\{\mathrm{E}\}$ is given by a rotation matrix

$$\mathbf{R}_{B'}^{E} = \begin{bmatrix} c\theta_y & s\theta_y & 0 \\ -s\theta_y & c\theta_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A.2.2 Quadcopter Dynamics

The upward thrust along the $-z^B$ axis is given by

$$T_i = b\bar{\omega}_i^2$$

where ω_i is the angular velocity of the rotor and $b > 0$ is a lift constant that depends on the air density, number of blades, cube of rotor blade radius, and chord length of the blade. According to Newton's second law of motion, the translation dynamics of the quadcopter in $\{E\}$ is given by

$$m\dot{\mathbf{v}}^E = \begin{bmatrix} 0 & 0 & mg \end{bmatrix}^T - \mathbf{R}_B^E \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T - B\mathbf{v} \tag{A.1}$$

where $\mathbf{v}$ is the velocity of quadcopter in $\{\mathrm{E}\}$ given by $\mathbf{v} = \begin{bmatrix} {v_x}^E & {v_y}^E & {v_z}^E \end{bmatrix}^T \in \mathbb{R}^3$, B is aerodynamics friction and $T = \Sigma T_i$ is total upward thrust generated by the rotors in $-z^B$ direction, g is the gravitational acceleration, and m is the mass of quadcopter. In (A.1), first term represents the gravitational force acting in $-z^E$ direction, second term represents the total upward thrust generated in $-z^B$ direction in $\{B\}$ and rotated in $\{E\}$.

The rotation in each axis is obtained by varying pairwise differences in rotor thrust. Torque about x and y axes is given by

$$\tau_x = dT_4 - dT_2 = db(\bar{\omega}_4^2 - \bar{\omega}_2^2)$$

$$\tau_y = dT_1 - dT_3 = db(\bar{\omega}_1^2 - \bar{\omega}_3^2)$$

The torque applied to each motor is opposed by aerodynamic drag and exerts a reaction torque on the frame which is given by $Q_i = k\bar{\omega}_i$ where k depends on the same factors as b. The total reaction torque about z-axis is given by

$$\tau_z = Q_1 - Q_2 + Q_3 - Q_4 = k\left(\bar{\omega}_1^2 + \bar{\omega}_3^2 - \bar{\omega}_2^2 - \bar{\omega}_4^2\right) \tag{A.2}$$

The total torque applied to the quadcopter body is $\mathbf{\Gamma} = \begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}$. By Euler's equation of motion rotational acceleration is given is by

$$J\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \mathbf{\Gamma} \tag{A.3}$$

where J is the 3×3 inertia matrix of the quadcopter and ω is a angular velocity vector of the quadcopter body frame. Overall quadcopter motion equation [59] is obtained by integrating equations (A.1) and (A.2) which is given by

$$\begin{bmatrix} \mathbf{T} & \mathbf{\Gamma} \end{bmatrix}^T = A \begin{bmatrix} \bar{\omega}_1^2 & \bar{\omega}_2^2 & \bar{\omega}_3^2 & \bar{\omega}_4^2 \end{bmatrix}^T \tag{A.4}$$

where $A = \begin{bmatrix} -b & -b & -b & -b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{bmatrix}$ is a constant invertible matrix since $b, k, d > 0$.

A.2.3 Quadcopter Dynamics as Double Integrators

The overall motion in the plane $x^{B'}y^{B'}$ is obtained by pitching and rolling the quadcopter by an angle θ_p and θ_r. The total force acting on a quadcopter is given by

$$\mathbf{f}^{B'} = \mathbf{R}x\left(\theta r\right)\mathbf{R}y\left(\theta p\right)\begin{bmatrix} 0 & 0 & T \end{bmatrix}^T \tag{A.5}$$

where,

$$\mathbf{R}x\left(\theta r\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\theta_r & s\theta_r \\ 0 & -s\theta_r & c\theta_r \end{bmatrix},\ \mathbf{R}y\left(\theta p\right) = \begin{bmatrix} c\theta_p & 0 & s\theta_p \\ 0 & 1 & 0 \\ -s\theta_p & 0 & c\theta_p \end{bmatrix}$$

are the Rotation matrix about x and y axes respectively.

Simplifying (A.5) we get $\mathbf{f}^{B'}$ as

$$\mathbf{f}^{B'} = \begin{bmatrix} T\sin\theta_p & T\sin\theta_r\cos\theta_p & T\cos\theta_r\cos\theta_p \end{bmatrix}^T$$

for small θ_p and θ_r the above equation can be approximated by

$$\mathbf{f}^{B'} \approx \begin{bmatrix} T\theta_p & T\theta_r & T \end{bmatrix}^T$$

i.e $f_x^{B'} \approx T\theta_p$ and $f_y^{B'} \approx T\theta_r$

According to Newton's Second law of motion $F = ma$, we can write

$$ma_x^{B'} = T\theta_p$$

$$\theta_p = \frac{m}{T} a_x^{B'}, \;\; \theta_r = \frac{m}{T} a_y^{B'}$$

where $a_x^{B'}$ is $a_y^{B'}$ is acceleration of quadcopter in $\{B'\}$ frame.
As the controller will receive the position and velocity in $\{E\}$ frame so it will compute the acceleration in $\{E\}$ frame which needs to be rotated to the $\{B'\}$ frame.

$$\mathbf{f}^{B'} = \mathbf{R}_E^{B'} \mathbf{f}^{\mathbf{E}}$$

$$m\mathbf{a}^{B'} = \mathbf{R}_E^{B'} m\mathbf{a}^{\mathbf{E}}$$

$$\mathbf{a}^{B'} = \mathbf{R}_E^{B'} \mathbf{a}^{\mathbf{E}}$$

This equation shows the motion of the quadcopter in $x^{B'}y^{B'}$ plane.

Remark 1. *While the discussion above describes the motion of the quadcopter in $x^{B'}y^{B'}$ plane as a double integrator, motion along $-z^{B'}$ is directly dependent on the thrust generated by the rotors in $\{B\}$ frame given by $T = \Sigma T_i$, which is naturally a double integrator.*

A.2.4 Preliminaries of Distributed Control

A homogeneous multi-agent system is a collection of n agents with identical dynamics communicating with each other over a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of vertices $\mathcal{V} = \{\alpha_i,\ i = 1, ..., n\}$ represents the agents and the edges $(\alpha_i, \alpha_j) \in \mathcal{E}$ denote the communication from agent α_i to α_j. The communication graph $\mathcal{G}$ is *undirected* if $(\alpha_i, \alpha_j) \in \mathcal{E} \implies (\alpha_j, \alpha_i) \in \mathcal{E}$ otherwise it is *directed*. A *path* in a graph is a sequence of edges. An undirected graph is said to be *connected* if it has a path from each agent to every other agent. A directed graph is said to contain a *directed spanning tree* if there is an agent α_r (called *root*) such that there is a directed path from α_r to every other agent. Set of neighbours for each agent is given by $\mathcal{N}_i := \{j \in \mathcal{V} : (\alpha_i, \alpha_j) \in \mathcal{E}\}$. The Laplacian matrix $\mathcal{L}$ of a graph $\mathcal{G}$ is given by $\mathcal{L}_n = [lij] \in \mathbb{R}^{n \times n}; l_{ij} = -a_{ij}, i \neq j, l_{ii} = \sum_{j=1, j\neq i}^{n} a_{ij}$, where a_{ij} is the *weight* of the edge (α_i, α_j). Some control algorithms use leader-follower configuration of the multi-agent systems in which, the set of agents is divided into a set of leader $\mathbf{L}$ and a set of followers $\mathbf{F}$.

In this paper, agents are assumed to have double-integrator dynamics, given by

$$\ddot{\mathbf{x}}_i^E(t) = \mathbf{f}_i^E(t) \tag{A.6}$$

where $\mathbf{x}_i^E = \begin{bmatrix} \mathbf{p}_i^E \\ \mathbf{v}_i^E \end{bmatrix} \in \mathbb{R}^{2d}$ is the state vector of α_i and $\mathbf{f}_i^E \in \mathbb{R}^d$ denotes the input to α_i. The vectors $\mathbf{p}_i^E$ and $\mathbf{v}_i^E$ denote the position and velocity of α_i, respectively, in $\{E\}$ frame.

$$f_i^E = \begin{cases} \sum_{j=1}^{n} \\ 0 \text{ if } a_i \in \mathcal{A} \end{cases} \tag{A.7}$$

where a_{ij} is the (i,j) entry of the adjacency matrix $A_n \in \mathbb{R}^{n \times n}$ corresponding to the communication graph $\mathcal{G}_n$ and β is a positive constant.This control law is the modified version of the control law mentioned in [60].The derivation of this law is provided in [56].The leader position is seprately controlled by a waypoint Navigation Controller.The control law discussed here drives **n** quadcopters to consensus i.e for any initial p_i^E and v_i^E and for all i,j = 1,....,n i.e $||p_i^E(t) - p_j^E(t)|| \to 0$ and $v_i^E(t) \to 0$ as $t \to \infty$.

Remark 2. *The heading control loop given in the next section is designed such that $\theta_y \to \theta_y^*$ relatively fast and is held constant. Thus, matrix $\mathbf{R}_{B'}^E$ can be considered as constant.*

A.2.5 Preliminaries of Bearing based Formation Control

Consider the multi-agent system of $n-$ agents $a_1, a_2, \ldots, a_n$ with dynamics specified by (A.6). The communication between the agents is specified by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the agents form the node set $\mathcal{V}$ and communication between the agents forms the edge set $\mathcal{E}$. An edge $(a_i, a_j) \in \mathcal{E}$ if there is a communication between agents a_i and a_j. For the communicating agents, the edge and bearing vectors are defined respectively as,

$$e_{ij} = p_j - p_i \text{ and } g_{ij} = \frac{e_{ij}}{\|e_{ij}\|} \tag{A.8}$$

The formation $\mathcal{F}$ of the multi-agent system can be specified using absolute positions, inter-agent distances or inter-agent bearings. To specify a formation using inter-agent bearings alone, the formation need to satisfy certain conditions specified by *Bearing Rigidity Theory.* Let g_{ij}^* denote the desired bearing between agent a_i and a_j. The orthogonal projection matrix P_{ij}^* corresponding to the desired bearing is defined as

$$P_{ij}^* = I_d - g_{ij}^* g_{ij}^{*T}. \tag{A.9}$$

A weighted Laplacian matrix L_g^* for the formation can be defined as

$$L_g^* = .. \tag{A.10}$$

The Bearing rigidity theory

A.2.6 Preliminaries of Human-in-the-loop Control

Human-in-the-loop (HITL) refers to a type of system in which a human operator is involved in the control loop of a robotic system. In a robotics simulator, human-in-the-loop typically means that a human operator can interact with the simulated robotic system in real time and make decisions or provide input that affects the behavior of the simulated robot. There are many potential

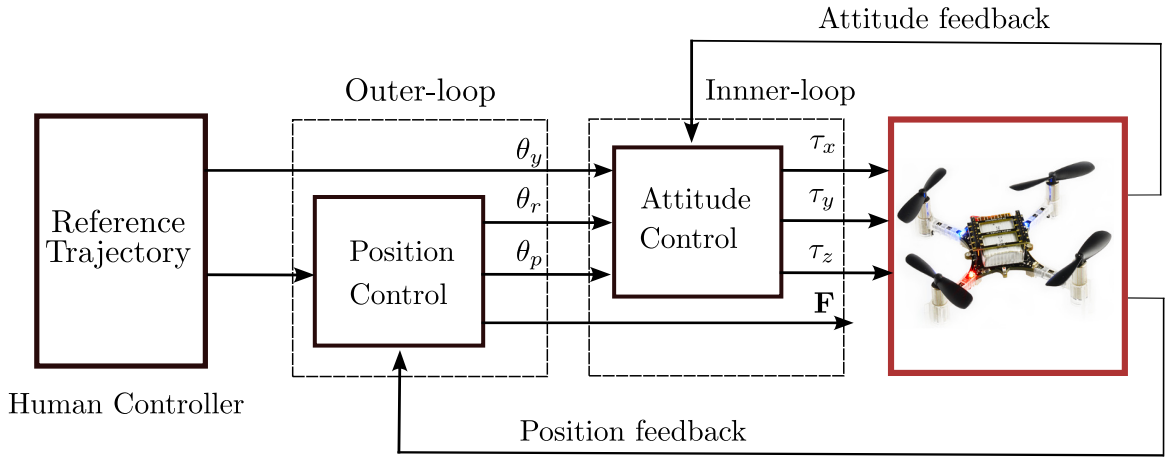


Figure A.2: Human-in-the-loop control Block diagram

applications for human-in-the-loop robotics simulations. For example, a robotics simulator with human-in-the-loop functionality could be used to test and evaluate the performance of a new robotic system, or to train operators to work with the system in a safe and controlled environment. It could also be used to test and validate control algorithms or other software that will be used to operate the real-world robotic system.

A.3 MASCOT: Structure and Features

A.3.1 Tools Used

Robot Operating System (ROS)

ROS is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS provides a distributive architecture that eliminates the problem of a single node handling all the tasks. ROS supports low-level drivers and programming in various languages such as Python, C++, Java, and Lisp. This makes it convenient to directly deploy the codes written for simulation on the physical systems. Visualization and debugging

tools such as Rviz and rqt make it very easy to keep track of the process [58]. The basic architecture of ROS is shown in Fig. A.3.

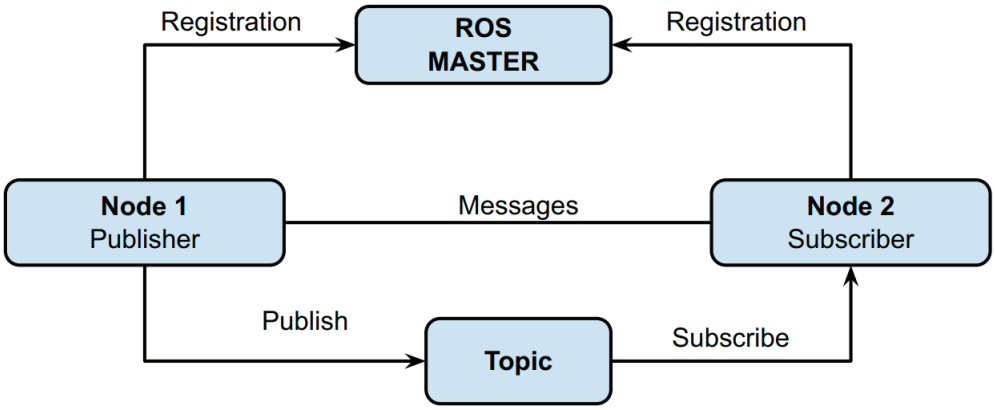


Figure A.3: ROS Architecture

Gazebo

The gazebo is a 3D simulation platform that provides robots, sensors, and environment models which are required for robot development. It uses Open Dynamics Engine (ODE) for real-time simulations and graphics. Gazebo supports a wide range of sensors such as LRF,2D/3D camera, Depth Camera, and Force-Torque Sensor. APIs provided in Gazebo enable users to create robots, sensors, and environment control as a plug-in.

Falcon Haptic Controller

We are using Novint Falcon as an input for Human-in-the-loop simulation.The Novint Falcon is a haptic device that is designed to provide users with the ability to touch and feel virtual objects in a computer-generated environment. It is typically used in gaming, virtual reality, and simulation applications.The Novint Falcon uses a series of motors and linkages to create force feedback and motion, allowing users to experience the sensation of touch and motion in virtual environments. It has three degrees of freedom (DOF), allowing it to move in the x, y, and z axes. It also has a number of buttons and controls that can be used to interact with the virtual environment.

One of the unique features of the Novint Falcon is its ability to create a wide range of haptic sensations, including force feedback, vibrations, and thermal effects. This allows it to

provide a more immersive and realistic experience for users, making it useful for a variety of applications, including gaming, training, and simulation. Overall, the Novint Falcon is a pow-

Figure A.4: Falcon Haptic Controller

erful haptic device that allows users to touch and feel virtual objects in a computer-generated environment, providing a more immersive and realistic experience. It is useful for a variety of applications, including gaming, training, and simulation, and its ability to create a wide range of haptic sensations makes it particularly useful for creating more realistic and engaging virtual environments. The kinematics and dynamics of this device can be found at [61].

Crazyflie Quadcopter

The harware quadcopter used in the testbed for hybrid simulation is Crazyflie 2.1. The Crazyflie 2.1 is a small and lightweight drone developed by Bitcraze, a Swedish robotics company. The Crazyflie 2.1 is equipped with a number of sensors and features that allow it to navigate and stabilize itself in flight. It has an onboard camera, gyroscopes, accelerometers, and magnetometers, as well as a barometer and ultrasound rangefinder. It also has a wireless communication system, allowing it to be controlled remotely using a radio controller or a computer.The Crazyflie 2.1 quadcopter measures 92 millimeters between diagonally opposed motor shafts and weighs 27 grams with a battery. It contains a 32-bit, 168- MHz ARM microcontroller with floating-point unit that is capable of significant onboard computation. Software and hardware are both open-

source. The Crazyflie communicates with a PC over the Crazyradio PA, a 2.4 GHz USB radio that transmits up to two megabits per second in 32-byte packets. The Crazyflie 2.1 is capable

Figure A.5: Crazyflie Quadcopter

of flying for up to 7 minutes on a single battery charge, and can reach speeds of up to 10 meters per second. It is small and lightweight, weighing just 27 grams, and is capable of flying indoors and outdoors. We have added Optitrack markers and flow deck which increases its weight to 33 grams and reduces the flight time to 6 minutes. [62]

Optitrack Localization System

The Optitrack Localization System is a motion capture system that utilizes infrared cameras and reflective markers to track the three-dimensional position and orientation of objects in real-time. This technology has been widely adopted in research and commercial applications, including motion analysis, robotics, and virtual reality. In our study, we employed the Optitrack system with six Flex13 infrared cameras. These cameras were placed at strategic locations around the tracking volume to ensure comprehensive coverage and accurate tracking of the markers. The high accuracy and precision of the Optitrack system, with tracking errors typically less than 1

millimeter, made it an ideal choice for our research. Additionally, the system's ease of setup and use made it a convenient and reliable tool for our study. Figure A.6 shows the setup of the Localization system. The Motive software from Optitrack is used for the pose estimation of the agents in the workspace which is communicated to the PC running ROS over the VRPN network and further control is handeled on the PC running ROS.

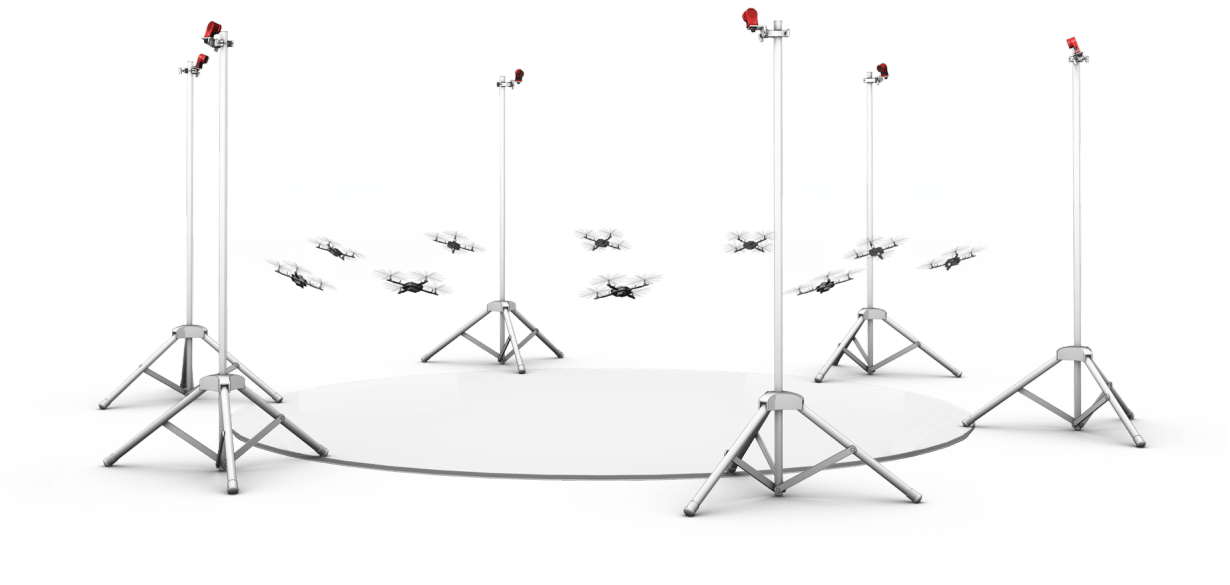


Figure A.6: Optitrack System with siz Flex13 cameras

Quadcopter Simulation Package

The quadcopter model used in MASCOT is based on the tum-simulator AR Parrot Drone Gazebo simulation package [63] developed by the Computer Vision Group at the Technical University of Munich. The lower level controller is modified and tuned so that the quadcopters can be treated as a double integrator system for control algorithm implementation, and added topics and controls required for the implemented model. A general control script is written for the implementation of the different use cases discussed in Section A.4;

A.3.2 Control Block

Figure A.7 shows the Control block diagram of the system. The low-level control of the quadcopter is achieved by four independent controllers given below:

- Heading Control
- Altitude Control

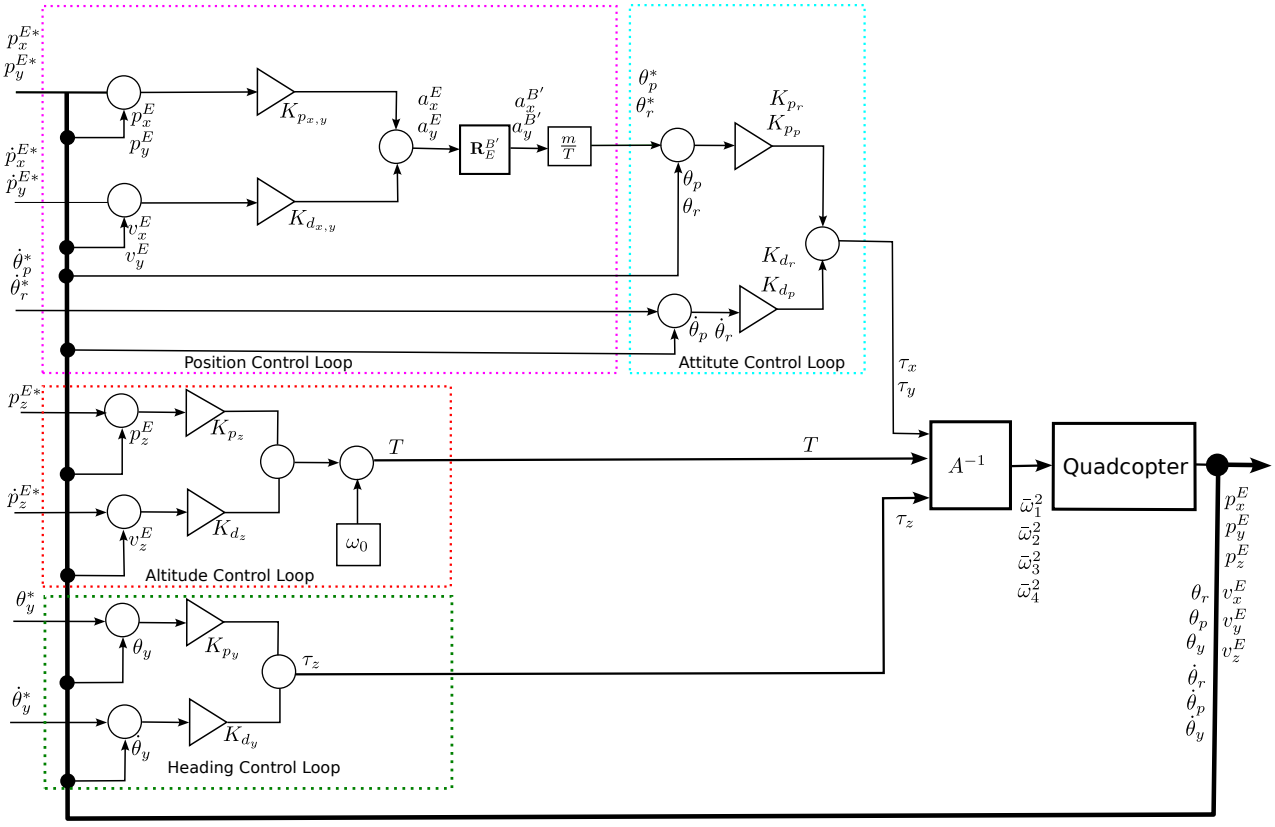


Figure A.7: Control Block diagram

- Pitch and Roll angle control
- Position Control

Altitude Control

The Altitude of the quadcopter is achieved by the total Thrust T generated by all the rotors. This thrust is controlled by the following proportional-derivative controller.

$$T = K_{p_z}\left(p_z^* - p_z\right) + K_{d_z}\left(\dot{p}_z^* - \dot{p}z\right) + \omega_0$$

where p_z is the current altitude, p_z^* is the desired altitude, K_p, K_d are proportional, derivative gains respectively and ω_0 is the rotor speed bias which generates a trust to counter the weight of quadcopter

Heading Control

The heading is a yaw angle θ_y which represents the angle between x^B and x^E about z^Eaxis. To control this angle we generate a required torque τ_z which is given by the following proportional-

derivative controller.

$$\tau_z = K_{p_y}\left(\theta_y^* - \theta_y\right) + K_{d_y}\left(\dot{\theta}_y^* - \dot{\theta}_y\right)$$

where θ_y is a current yaw angle, θ_y^* is the desired yaw angle and K_p,K_d are proportional,derivative gains respectively.

Controlling θ_r and θ_p

The roll angle θ_r represents the angle between $y^{B'}$ and y^B about $x^{B'}$ axis.To control this angle we generate a required torque τ_r which is given by the following proportional-derivative controller.

$$\tau_x = K_{p_r}\left(\theta_r^* - \theta_r\right) + K_{d_r}\left(\dot{\theta}_r^* - \dot{\theta}_r\right)$$

where θ_r is a current roll angle,θ_r^* is the desired roll angle and K_{p_r}, K_{d_r} are proportional,derivative gains respectively.

The pitch angle θ_y represents the angle between $x^{B'}$ and x^B about $y^{B'}$ axis.To control this angle we generate a required torque τ_p which is given by the following proportional-derivative controller.

$$\tau_y = K_{p_p}\left(\theta_p^* - \theta_p\right) + K_{d_p}\left(\dot{\theta}_p^* - \dot{\theta}_p\right)$$

where θ_p is a current pitch angle,θ_p^* is the desired pitch angle and K_{p_p}, K_{d_p} are proportional,derivative gains respectively.

Remark 3. *With the change in θ_p and θ_r there is also a reduction in the vertical component of the total thrust T by the factor of the cosine of θ_p and θ_r. As per our assumption for a very small angle, this reduction is not much which is rapidly corrected by an Altitude control loop.*

Position Control

The position of the quadcopter in $x^{B'}y^{B'}$ plane is controlled independently by the proportional-derivative controller for each axis.

$$a_x = K_{p_x}\left(p_x^* - p_x\right) + K_{d_x}\left(\dot{p}_x^* - \dot{p}x\right)$$

$$a_y = K_{p_y}\left(p_y^* - p_y\right) + K_{d_y}\left(\dot{p}_y^* - \dot{p}y\right)$$

Note: Here the a_x and a_y computed is in $\{\mathrm{E}\}$ as feedback position and velocity is also in $\{\mathrm{E}\}$ frame.

A.3.3 Architecture and Features of MASCOT

The simulation architecture of MASCOT is shown in Fig. A.8. In this system, the Gazebo sim-

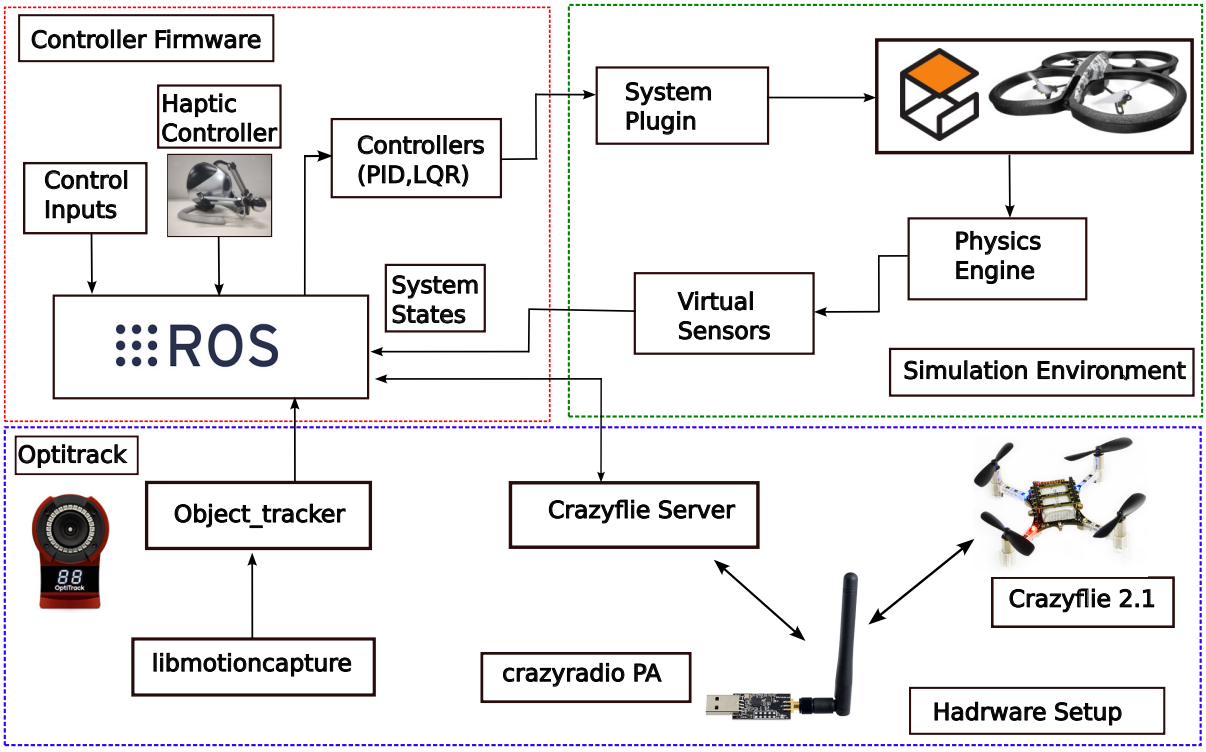


Figure A.8: System Architecture

ulator simulates the model and its sensors with the system plugin. Gazebo's internal scheduler provides the ROS interfaces. This ROS interface enables a user to manipulate the properties of the simulation environment over ROS, as well as spawn and introspect on the state of models in the environment. ROS works as a middleware that runs the independent controller of individual robots spawned in Gazebo environments, ROS provides the intercommunication channel within the nodes, where all the controller nodes communicate with each other and Gazebo over the TCPROS protocol and exchange the information. The control node is written in python for the high-level control of the quadcopter the low-level control and plugin are developed in C++ which provides the tuned low-level control of the quadcopter. If needed user can modify or tune this low-level control.

A.3.4 Configuring the Hybrid-Simulation

For simulating the multi-agent systems using MASCOT, users can provide configuration parameters and control laws using a plain text configuration file, which is used by the internal programs for the initialization and simulation of the quadrotor.

The configurable parameters required are as follows:

- **Robot:** Details of the Robots to be simulated
  - **Number:** No. of Agents to be spawned in Gazebo.
  - **IntialPosition:** Flag to enable random initial position or specified initial position.
  - **Position:** Initial Position of Robot (x,y,z) and mode of operation real or virual
- **Control:** Controls laws to be used
  - **Custom-Control:** Flag to Enable Custom Control.
  - **Tutorial Examples:** Flag to Enable Tutorial.
    1. **Waypoint Navigation:**
       * **P-Gain:** Proportial Gain , Default = 1.0
       * **D-Gain:** Differential Gain , Default = 1.0
    2. **Formation Control:** Flag to Enable Formation Control.
    3. **Formation Control Shape:** Shape of the formation Square, Triangle, Hexagon.
    4. **Falcon:** Enable falcon haptic controller for human-in-the-loop control.
    5. **Consensus:** Flag to Enable Consensus Control.
       * **Leader:** Specify the robot number that to be leader, 0-for leaderless consensus.
       * **Communication Graph:** Flag for communication graph
       * **L-mat:** Laplacian Matrix.
    6. **Min-max Consensus:** Enable Min-max Consensus Control.
- **Output:** Configuration of output
  - **Velocity :** Enable it to generate the velocity plot.
  - **Position :** Enable it to generate the position plot.

– **Save-plot :** Enable to save the plots.

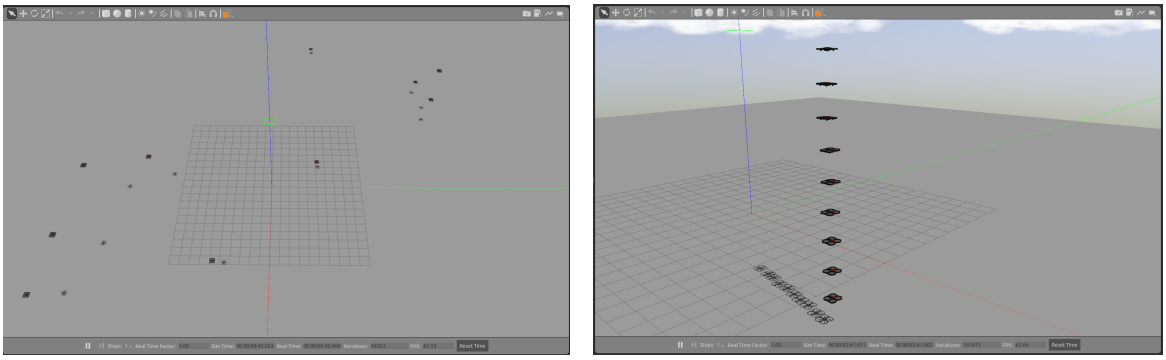
– **Show-plot :** Enable it to show plots.

– **Save-data :** Enable to save as numpy-array

A user needs to specify their control laws by setting the Custom-Control flag (set as default) as plain text using python syntax. The user can use the states of the quadcopter such as position, velocity, acceleration, and orientation in the control algorithm and can publish the acceleration command which is an output of the control algorithm. Few demos are made available as tutorials for the users which are described in Section A.4 below.

A.4 Examples

In this section, the simulation results of MASCOT are demonstrated using a few examples of linear as well as non-linear distributed control laws designed for multi-agent systems. A fewer number of agents are used for clarity of figures. All the Algorithms are tested on the Personal Computer with Intel i7 running at 2.8 GHz and 16 GB RAM. The OS used is Ubuntu 20.04.4 with ROS Noetic and Python 3.8.10.



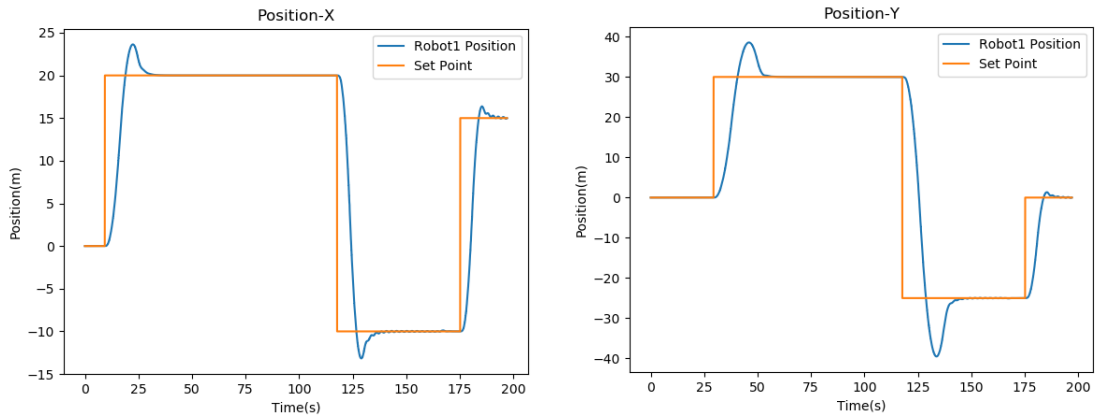
(a) Initial position of quadcopters at t = 0 s

(b) Final position of quadcopters after consensus at t = 80 s

Figure A.9: Visualization in Gazebo

A.4.1 Waypoint Navigation

The quadcopter is initialized at a $(0, 0, 1)$ position and the different waypoints are given in the $x^E y^E$ plane. From Figure A.10, it can be inferred that the quadcopter achieves its goal position with a very small overshoot, which depends on the parameters of the controller.



(a) Position in X-axis

(b) Position in Y-axis

Figure A.10: Quadcopter Position Plots.

A.4.2 Consensus Algorithms

We demonstrate two consensus algorithms that have linear control laws [60]: 1) for leaderless asymptotic consensus, and 2) leader-follower configuration i.e. consensus tracking In a leaderless case, four agent system is considered which are initialized at random positions, while in leader-follower configuration a ten-agent system is considered with $\alpha_4 \in \mathbf{L}$.

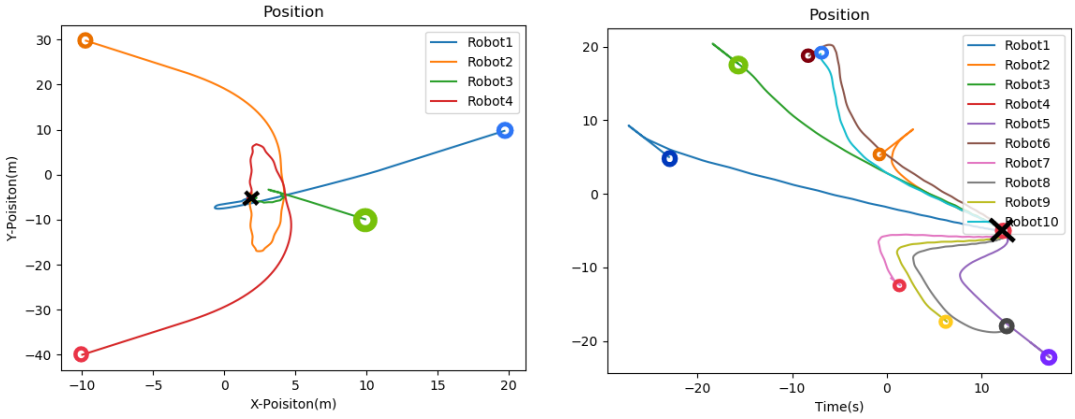
The control algorithms used is as follows:

$$\mathbf{f}_i^E = \begin{cases} \sum_{j=1}^{n} a_{ij} \left(\mathbf{p}j^E - \mathbf{p}i^E \right) - \beta \mathbf{v}_i^E \text{ if } \alpha_i \in \mathbf{F} \\ \\ 0 \text{ if } \alpha_i \in \mathbf{L} \end{cases} \tag{A.11}$$

where β is a positive constant. Note that, in a leaderless case, all the agents are assumed to be followers. All the quadcopters run their controllers simultaneously and plugin under their respective namespace. Figure A.9 shows the visualization of the convergence of quadcopters in the Gazebo simulator. For the leaderless case, Fig. A.11(a) shows the trajectory followed by all the quadcopters to reach the consensus point. Fig. A.12 shows the convergence of the position of the quadcopter in the x^E and y^E axis. For leader-follower configuration Fig. A.11(b) shows the trajectory followed by the follower and Fig. A.13 show the convergence of the position of the Quadcopter in the x^E and y^E-axis.

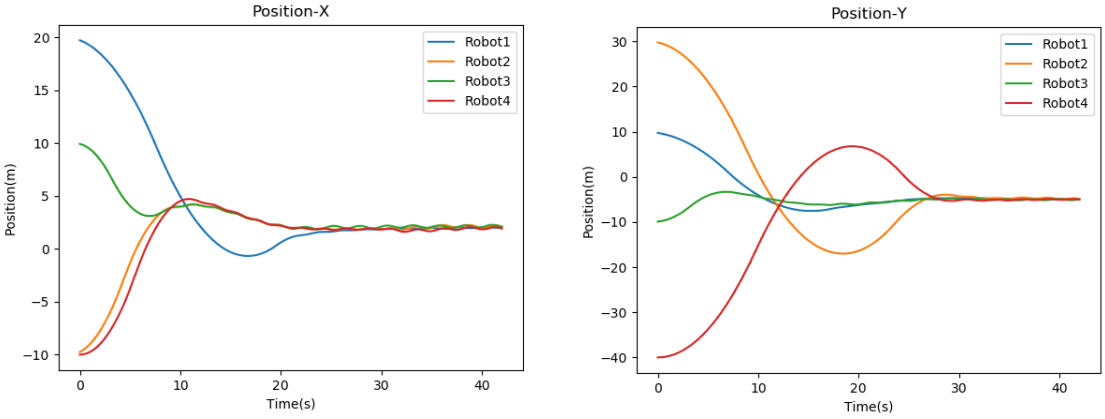
A.4.3 Min-max time Consensus Control

To demonstrate the simulations of non-linear distributed control laws, a min-max time consensus tracking algorithm proposed in [64] is used. This control algorithm extracts a directed



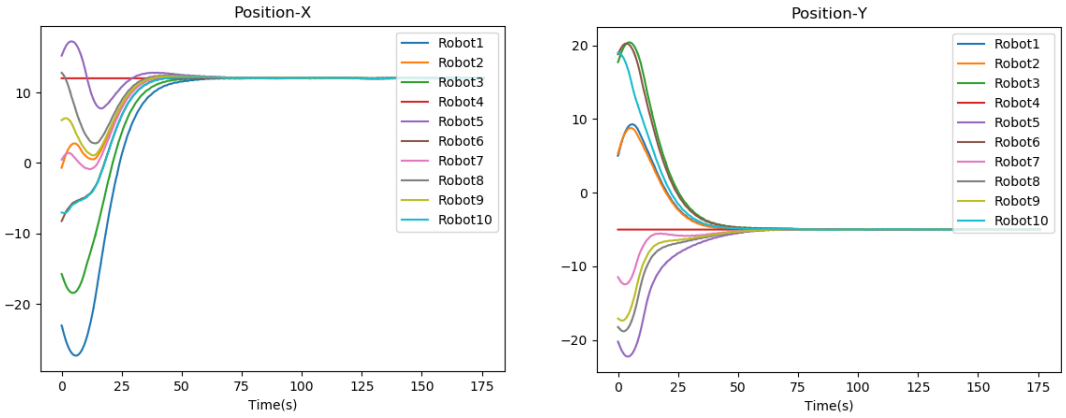
(a) Leaderless Control plot (b) Leader Follower plot

Figure A.11: Consensus Control Position Plots.



(a) Position in X-axis (b) Position in Y-axis

Figure A.12: Quadcopter Leaderless Control Plots.



(a) Position in X-axis (b) Position in Y-axis

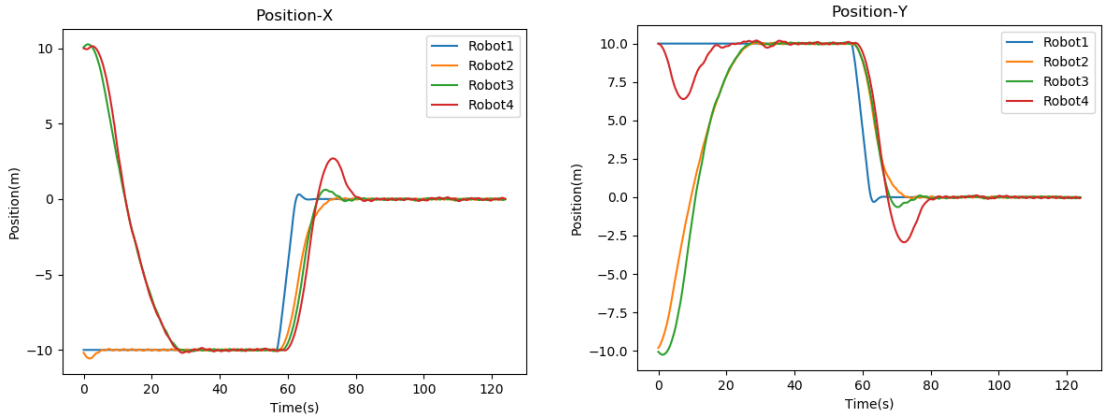
Figure A.13: Quadcopter Leader Consensus Control Plots.

spanning tree from the communication graph between the agents and treats each edge as a leader-follower pair. The root of the spanning tree gives the reference trajectory for consensus tracking. The agents are assumed to have bounded inputs. Using the difference of states between the leader (α_p) and the follower (α_c) in each edge, a distributed bang-bang control law is given for the following agents:

$$\mathbf{f}_c^E = \beta_c \text{sign}(2(\beta_c - \beta_p)(\mathbf{p}_c - \mathbf{p}_p) + (\mathbf{v}_c - \mathbf{v}_p)^2 \text{sign}(\mathbf{v}_c - \mathbf{v}_p)) \tag{A.12}$$

where β_p and β_c are the input bounds of α_p and α_c respectively.

For simulation, a four-agent system is used with α_1 as the root agent. Figure A.14 shows the test result for this control.



(a) Position in X-axis

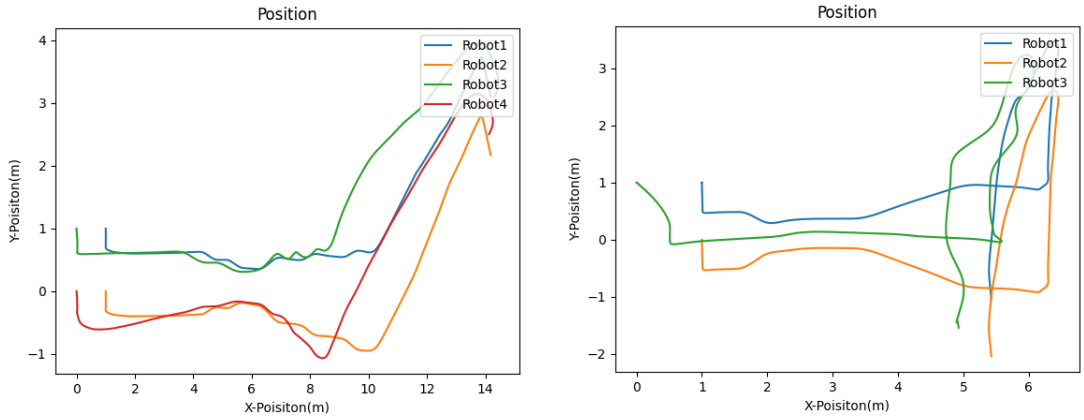
(b) Position in Y-axis

Figure A.14: Quadcopter minmax Consensus Control Plots.

A.4.4 Formation Control

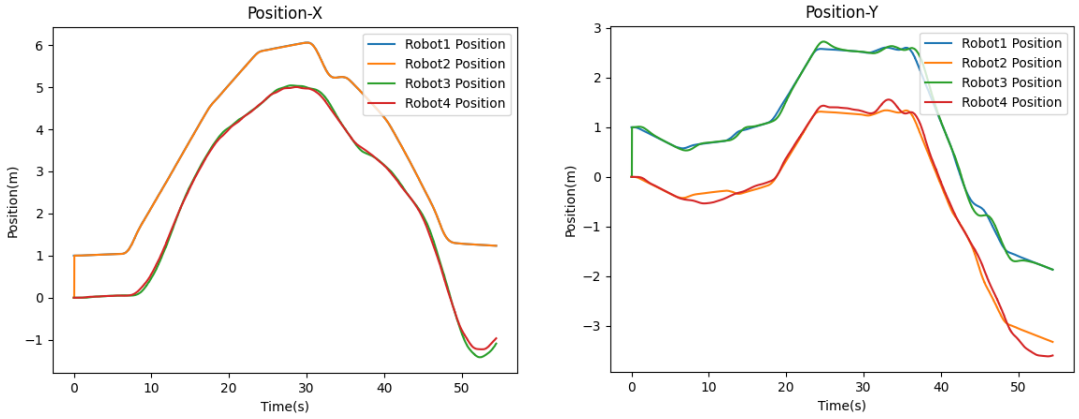
Human-in-the-loop

The bearing based formation control for Square and Triangle shape is show in figure A.15 where for the square shape formation in figure A.15(a) Robot1 and Robot2 are leaders and Robot3 and Robot4 are followers. In figure A.15(b) Robot1 and Robot2 are leader and Robot3 is Follower. The trajectory and scale of the formation is controlled using Novint Falcon Haptic controller as Human in the loop control. Figure A.16 shows the trajectory of the quadcopters in X and Y axis respectively in the Square formation. Figure A.17 shows the trajectory of the quadcopters in X and Y axis respectively in Triangle formation.



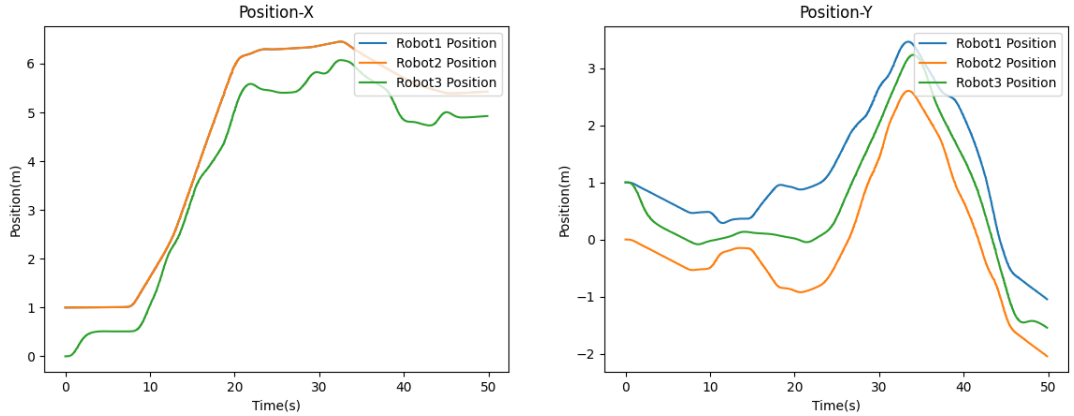
(a) Square Formation (b) Triangle Formation

Figure A.15: Formation Control Position Plots.



(a) Position in X-axis (b) Position in Y-axis

Figure A.16: Quadcopter Square Formation Control Plots.

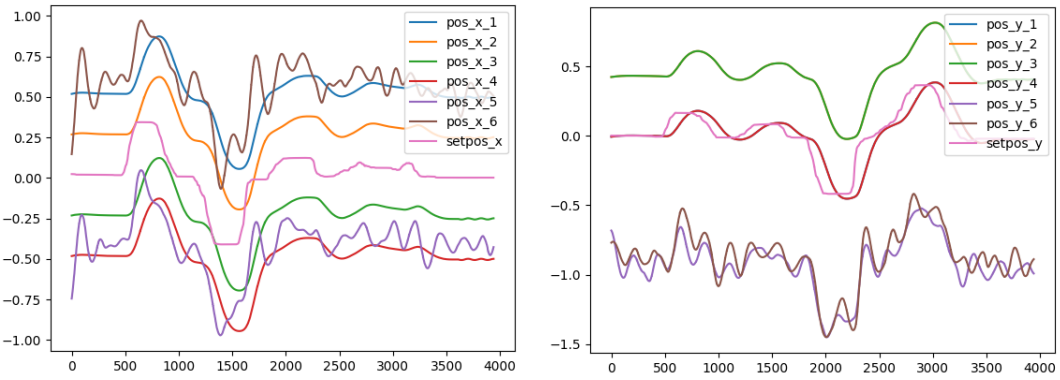


(a) Position in X-axis (b) Position in Y-axis

Figure A.17: Quadcopter Triangle Formation Control Plots.

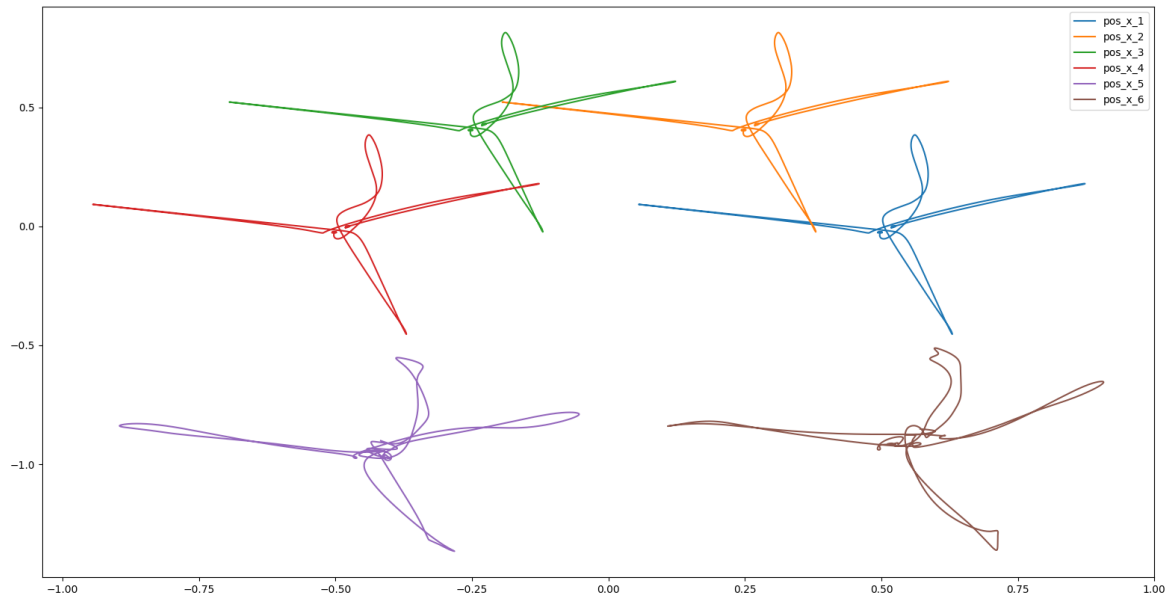
Hybrid

In hybrid simulation a formation control for a Hexagon shape is demonstrated where the agents are located to the vertices of Hexagon and the command for the motion is provided with falcon haptic controller. This experiment shows that the system is capable of handling human-in-the-loop control in an hybrid setup where few of the agents are in real world such as crazyflie and others in the simulation environment of Gazebo. In the below setup 5th and 6th agents of the Hexagon shape are in the real-world and the other 4 agents are in simulation. The virtual and real drones can communicate with each other with ROS network. Figure A.18 shows the plot for the hybrid simulation.



(a) Position in X-axis

(b) Position in Y-axis



(c) Position Position in XY-axis

Figure A.18: Quadcopter Hybrid Hexagon Formation Control Plots.

Publication

The part of this work is accepted for publication in the *2022 Eighth Indian Control Conference (ICC), IIT Madras, India*

List of Publication(s)

[1] Arvind Pandit, Akash Njattuvetty, and Ameer K. Mulla, "ROS-Based **M**ulti-**A**gent **S**ystems **CO**ntrol Simulation **T**estbed (MASCOT)", *in 2022 Eighth Indian Control Conference (ICC), IIT Madras, India (Published)*

[2] Arvind Pandit, Om Patil, and Ameer K. Mulla, "**MoVINav**-A **Mo**nocular **V**isual **I**nertial **Nav**igation Framework", *in 62nd IEEE Conference on Decision and Control, Dec. 13-15, 2023, Singapore (Under Review)*

References

[1] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Automat. Mag.*, vol. 18, pp. 80–92, 12 2011.

[2] 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. 4, pp. 1004–1020, 2018.

[3] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.

[4] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[5] 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, 2020.

[6] J. Starr and B. Lattimer, "Evaluation of navigation sensors in fire smoke environments," *Fire Technology*, vol. 50, 11 2013.

[7] C. Papachristos, S. Khattak, and K. Alexis, "Uncertainty-aware receding horizon exploration and mapping using aerial robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4568–4575.

[8] C. Papachristos, K. Alexis, L. R. G. Carrillo, and A. Tzes, "Distributed infrastructure inspection path planning for aerial robotics subject to time constraints," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2016, pp. 406–412.

[9] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel, and R. Siegwart, "Structural inspection path planning via iterative viewpoint resampling with application

to aerial robotics," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6423–6430.

[10] T. Oksanen and A. Visala, "Coverage path planning algorithms for agricultural field machines," *J. Field Robotics*, vol. 26, pp. 651–668, 08 2009.

[11] D. Zermas, D. Teng, P. Stanitsas, M. Bazakos, D. Kaiser, V. Morellas, D. Mulla, and N. Papanikolopoulos, "Automation solutions for the evaluation of plant health in corn fields," 09 2015.

[12] H. Balta, J. Bedkowski, S. Govindaraj, K. Majek, P. Musialik, D. Serrano, K. Alexis, R. Siegwart, and G. De Cubber, "Integrated data management for a fleet of search-and-rescue robots," *Journal of Field Robotics*, vol. 34, pp. 539–582, 07 2016.

[13] T. Tomić, K. Schmid, P. Lutz, A. Dömel, M. Kassecker, E. Mair, I. Grixa, F. Ruess, M. Suppa, and D. Burschka, "Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue," *Robotics Automation Magazine, IEEE*, vol. 19, pp. 46 –56, 09 2012.

[14] S. Bijjahalli, R. Sabatini, and A. Gardi, "Advances in intelligent and autonomous navigation systems for small uas," *Progress in Aerospace Sciences*, vol. 115, p. 100617, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0376042120300294

[15] K. Schmid, P. Lutz, T. Tomić, E. Mair, and H. Hirschmüller, "Autonomous vision-based micro air vehicle for indoor and outdoor navigation," *Journal of Field Robotics*, vol. 31, 07 2014.

[16] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, 04 2013.

[17] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor mav," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4557–4564.

[18] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, 03 2015.

[19] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 15–22.

[20] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 2609–2616.

[21] G. Loianno, Y. Mulgaonkar, C. Brunner, D. Ahuja, A. Ramanandan, M. Chari, S. Diaz, and V. Kumar, "Smartphones power flying robots," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 1256–1263.

[22] C. Papachristos, F. Mascarich, and K. Alexis, "Thermal-inertial localization for autonomous navigation of aerial robots through obscurants," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, pp. 394–399.

[23] M. He and R. R. Rajkumar, "Using thermal vision for extended vins-mono to localize vehicles in large-scale outdoor road environments," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, 2021, pp. 953–960.

[24] ETHZ-ASL, "Kalibr camera calibration toolbox," 2022, [Online; accessed 22-July-2022]. [Online]. Available: https://github.com/ethz-asl/kalibr

[25] A. Richardson, J. Strom, and E. Olson, "Aprilcal: Assisted and repeatable camera calibration," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1814–1821.

[26] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[27] J. Rehder, J. Nikolic, T. Schneider, T. Hinzmann, and R. Siegwart, "Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4304–4311.

[28] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[29] Bruce D. Lucas and Takeo Kanade, "An iterative image registration technique with an application to stereo vision." in *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981*. William Kaufmann, 1981.

[30] A. M. Andrew, "Multiple view geometry in computer vision," *Kybernetes*, 2001.

[31] A. Vedaldi, "An implementation of sift detector and descriptor," *University of California at Los Angeles*, vol. 7, 2006.

[32] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[33] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International Journal of Computer Vision*, vol. 81, 02 2009.

[34] Y. Chen, Y. Chen, and G. Wang, "Bundle adjustment revisited," *arXiv preprint arXiv:1912.03858*, 2019.

[35] S. Shen, N. Michael, and V. Kumar, "Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft mavs," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5303–5310.

[36] P. J. Huber, *Robust Estimation of a Location Parameter*. New York, NY: Springer New York, 1992, pp. 492–518.

[37] D. Galvez-Lopez and J. Tardos, "Bags of binary words for fast place recognition in image sequences," *Robotics, IEEE Transactions on*, vol. 28, pp. 1188–1197, 10 2012.

[38] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," vol. 6314, 09 2010, pp. 778–792.

[39] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, 02 2014.

[40] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, 01 2016.

[41] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2502–2509.

[42] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "Deepmvs: Learning multi-view stereopsis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2821–2830.

[43] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 822–838.

[44] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth international conference on 3D vision (3DV)*. IEEE, 2016, pp. 239–248.

[45] D. Xu, E. Ricci, W. Ouyang, X. Wang, and N. Sebe, "Multi-scale continuous crfs as sequential deep networks for monocular depth estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5354–5362.

[46] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3712–3722.

[47] T. Li and J.-F. Zhang, "Asymptotically optimal decentralized control for large population stochastic multiagent systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 7, pp. 1643–1660, 2008.

[48] S. Li, H. Du, and X. Lin, "Finite-time consensus algorithm for multi-agent systems with double-integrator dynamics," *Automatica*, vol. 47, no. 8, pp. 1706–1712, 2011.

[49] F. Dörfler and B. Francis, "Formation control of autonomous robots based on cooperative behavior," in *2009 European Control Conference (ECC)*. IEEE, 2009, pp. 2432–2437.

[50] V. Cichella, I. Kaminer, V. Dobrokhodov, E. Xargay, R. Choe, N. Hovakimyan, A. P. Aguiar, and A. M. Pascoal, "Cooperative path following of multiple multirotors over time-varying networks," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 945–957, 2015.

[51] D. S. Drew, "Multi-agent systems for search and rescue applications," *Current Robotics Reports*, vol. 2, no. 2, pp. 189–200, 2021.

[52] C. H. Yong and R. Miikkulainen, "Coevolution of role-based cooperation in multiagent systems," *IEEE Transactions on Autonomous Mental Development*, vol. 1, no. 3, pp. 170–186, 2009.

[53] A. Ajorlou, A. Momeni, and A. G. Aghdam, "A class of bounded distributed control strategies for connectivity preservation in multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 12, pp. 2828–2833, 2010.

[54] E. Soria, F. Schiano, and D. Floreano, "Swarmlab: a matlab drone swarm simulator," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 8005–8011.

[55] P. Lu and Q. Geng, "Real-time simulation system for uav based on matlab/simulink," in *2011 IEEE 2nd International Conference on Computing, Control and Industrial Engineering*, vol. 1, 2011, pp. 399–404.

[56] A. Joshi, N. Limbu, I. Ahuja, A. K. Mulla, H. Chung, and D. Chakraborty, "Implementation of distributed consensus on an outdoor testbed," in *2016 European Control Conference (ECC)*, 2016, pp. 2146–2151.

[57] A. Joshi, V. Sawant, D. Chakraborty, and H. Chung, "Implementation of min-max time consensus tracking on a multi-quadrotor testbed," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 1073–1078.

[58] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[59] P. I. Corke and O. Khatib, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer, 2011, vol. 73.

[60] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*. Springer, 2008, vol. 27.

[61] N. Karbasizadeh, A. Aflakiyan, M. Zarei, M. T. Masouleh, and A. Kalhor, "Dynamic identification of the novint falcon haptic device," in *2016 4th International Conference on Robotics and Mechatronics (ICROM)*, 2016, pp. 518–523.

[62] W. Hönig and N. Ayanian, *Flying Multiple UAVs Using ROS*. Cham: Springer International Publishing, 2017, pp. 83–118. [Online]. Available: https://doi.org/10.1007/978-3-319-54927-9_3

[63] T. C. V. Group, "Tum simulator," https://github.com/tum-vision/tum_simulator, 2012.

[64] A. K. Mulla and D. Chakraborty, "Min–max time consensus tracking with communication guarantee," *IEEE Transactions on Automatic Control*, vol. 63, no. 1, pp. 132–144, 2017.