Politecnico di Torino

Master's Degree in Mechatronic Engineering

Enhancing UAV Autonomous Indoor Flight with Visual Odometry Techniques

Riccardo Catania

Supervised by
Prof. Alessandro Rizzo
Dr. Stefano Primatesta
Dr. Orlando Tovar Ordoñez

July 13, 2023

Abstract

In the rapidly evolving industrial landscape, Unmanned Aerial Vehicles (UAVs) have emerged as a pivotal component in enhancing operational efficiency and safety. These autonomous systems are particularly crucial in environments where direct human intervention is challenging or risky. The ultimate goal of this project is to assist human operators in specific missions by collecting and processing data. The drone should be able to provide stable flight in GNSS (Global Navigation Satellite System) denied environments exploiting the visual odometry algorithms and specific sensors. This thesis is a part of the FIXIT project, an initiative by the Competence Industry of Manufacturing 4.0 in Turin, Italy. The FIXIT project aims to establish a cooperative system between a UAV and an Autonomous Ground Vehicle (AGV), where the UAV can perform autonomous flights in industrial environments and dock on a moving rover. The drone is equipped with a Jetson Nano companion computer, a CubeOrange with Ardupilot, and a Lidar. The software stack includes Ubuntu 20.04,ROS Noetic, and other necessary software. An Intel Realsense d435i depth camera is also part of the drone's equipment, which is crucial for the implementation of a visual odometry algorithm. The focus of this thesis is on the development of a localization strategy based on visual odometry exploiting the depth camera and other sensors to increase localization accuracy under various conditions. The RTAB-Map ROS algorithm is used for state estimation, and its parameters are tuned to optimize the speed of state estimation while maintaining an acceptable level of accuracy. To ensure smooth drone movement, a Python script was developed to fuse data from RTAB-Map, Lidar, and IMU using a filtering technique. This approach enables the drone to operate autonomously in indoor environments with high levels of magnetic fields, where GPS signals may not be reliable. Experimental tests were conducted with the drone in an indoor GPS-denied environment to validate the effectiveness of the proposed solutions. Future work includes testing more computationally expensive forms of data fusion algorithms and implementing a YOLO algorithm for object recognition. The potential applications of this work extend to security and inventory management in factories, among others. The findings of this thesis contribute to the ongoing efforts to enhance the adaptability and flexibility of UAVs in the context of Industry 4.0.

Contents

Chapter 1

Introduction

1.1 UAV Applications and Visual Odometry

Unmanned Aerial Vehicles (UAVs), more commonly known as drones, have seen a significant rise in their application across various sectors. One of the most promising areas of their use is in the industrial sector. Industries are increasingly turning to UAVs to enhance operational efficiency, safety, and productivity. They are being used for tasks such as inspection of infrastructure, monitoring of production processes, and even in logistics for the delivery of goods within large industrial complexes.

The ability of drones to operate autonomously is a significant factor driving their adoption in the industrial sector. Autonomous drones can perform tasks without the need for constant human intervention, making them ideal for operations in environments that are hazardous or difficult for humans to access. For instance, in the case of infrastructure inspection, drones can easily access high or confined spaces, reducing the risk to human inspectors and increasing the efficiency of the inspection process.

A key aspect of autonomous operation in drones is their ability to accurately determine their position and navigate through their environment. This is where Visual Odometry (VO) comes into play. VO is a technique used for estimating the 3D pose of a robot by analyzing the camera images. In the context of drones, VO allows them to localize themselves in their environment using only the images captured by an onboard camera.

The implementation of visual odometry algorithms in drones offers several advantages. Firstly, it allows for more precise navigation, especially in environments where GPS signals may be weak or non-existent. This is particularly useful in indoor environments or in areas with dense infrastructure. Secondly, visual odometry can provide real-time feedback, allowing for more responsive control of the drone. This can be crucial in situations where the drone needs to react quickly to changes in its environment.

Moreover, the combination of autonomous operation and visual odometry in drones opens up new possibilities for their application. For instance, in the context of Industry 4.0, drones equipped with VO can work in conjunction with other autonomous systems, such as ground vehicles, to perform complex tasks. This cooperative system can lead to even greater efficiencies in industrial operations.

In conclusion, the use of UAVs, particularly those equipped with visual odometry algorithms, holds great promise for the future of industrial operations. Their ability to operate autonomously and navigate accurately in various environments can lead to significant improvements in operational efficiency and safety. As research in this area continues, we can expect to see even more innovative applications of these technologies in the industrial sector.

1.2 The FIXIT Project

This thesis is centered around the autonomous capabilities of the drone involved in the FIXIT project. Specifically, the emphasis is on the implementation of Visual Odometry (VO) algorithms to enable the drone to navigate autonomously in indoor environments. The FIXIT project is an initiative by CIM 4.0, the Competence Industry of Manufacturing 4.0, based in Turin.

The drone, an Unmanned Aerial Vehicle (UAV), is designed to operate in various environments, including but not limited to industrial settings, both indoors and outdoors. The drone's mission involves not only data collection and processing but also providing services in various structures, thereby enhancing operational efficiency and safety.

Upon completion of its mission, the drone's primary task is to land on an Autonomous Mobile Robot (AMR). The AMR is equipped with a Lidar sensor and uses a Simultaneous Localization and Mapping (SLAM) algorithm to estimate its position, enabling it to navigate autonomously within the environment.

The drone's ability to operate autonomously, collect, and process data is vital for the success of the FIXIT project. The data collected by the drone can offer valuable insights into various processes, facilitating more informed decision-making and optimization of operations.

The drone's autonomous capabilities are significantly enhanced by the use of Visual Odometry (VO) techniques. VO allows the drone to estimate its position and orientation by analyzing the camera images. This is particularly beneficial in indoor environments or areas with dense infrastructure where GPS signals may be weak or non-existent.

In conclusion, the FIXIT project signifies a considerable advancement in the application of UAVs in various sectors. The combination of autonomous operation, data collection, and processing capabilities of the drone, along with the autonomous navigation capabilities of the AMR, can lead to significant improvements in operational efficiency and safety.

Figure 1.1: The FIXIT Project

1.3 Objective of the Thesis

The primary objective of this thesis is to enable autonomous indoor flight for Unmanned Aerial Vehicles (UAVs) using Visual Odometry (VO) techniques, in the absence of GPS. The focus is on creating a balance between the accuracy of the drone's movements and the speed of its operations, ensuring smooth navigation while avoiding sluggish performance.

The cornerstone of this project is the implementation of a visual odometry algorithm. This algorithm is tasked with estimating the drone's position and orientation using the onboard camera and other sensors. The challenge lies not only in the successful implementation of the algorithm but also in its integration with the drone's existing hardware setup.

The aim is to leverage the data from various onboard sensors to enhance the accuracy and speed of the drone's pose estimation. This involves a careful calibration of the algorithm parameters, ensuring that the drone can navigate smoothly and swiftly in various indoor conditions.

The experimental tests conducted as part of this thesis aim to validate the effectiveness of the proposed solutions in real-world scenarios. The ultimate goal is to enhance the operational efficiency and safety of UAVs in challenging indoor environments, thereby contributing to the broader field of autonomous systems and Industry 4.0.

1.4 Thesis Outline

- **Chapter 1: Introduction** - This chapter provides an overview of the thesis, including the motivation, objectives, and contributions of the research. It also presents the organization of the thesis.

- **Chapter 2: Literature Review** - This chapter delves into the existing literature on visual odometry algorithms and data fusion techniques. It provides a comprehensive review of different types and methodologies of visual odometry algorithms, camera selection for VO, optimal conditions for VO algorithms, and data fusion techniques like Extended Kalman Filter and Dead Reckoning.

- **Chapter 3: Building the Drone and Software Setup** - This chapter details the construction of the drone and its components, including the drone frame, Cube Orange, Jetson Nano, Intel RealSense d435i, and Lidar. It also explains the software installation and configuration process, reading data from IMU, Lidar, and Camera on ROS Topics, and the use of the RTAB-Map ROS Algorithm for state estimation.

- **Chapter 4: RTAB-Map Parameters and Tuning** - This chapter focuses on understanding the RTAB-Map parameters and how to tune them for speed and accuracy.

- **Chapter 5: Data Fusion and Filtering Techniques** - This chapter discusses the need for data fusion and presents a Python script for data fusion. It also discusses the use of Lidar for height measurement, the implementation of Robot Localization, and the tuning of parameters of the EKF.

- **Chapter 6: Results and Discussions** - This chapter presents the experimental setup, testing methodology, and data analysis. It discusses the performance of Dead Reckoning and EKF, challenges in featureless environments, and handling high-speed turns. It also discusses the response of algorithms to data loss from RTAB-Map.

- **Chapter 7: Future Work and Conclusion** - This chapter discusses potential future work, including testing more computationally expensive forms of data fusion algorithms, implementing the YOLO algorithm for object recognition, and the use of the drone in high magnetic field environments. It also concludes the thesis by summarizing the findings and contributions of the research.

Chapter 2

Literature Review

2.1 Visual Odometry Algorithms

2.1.1 Types and Methodologies of Visual Odometry Algorithms: An In-depth Review

Visual Odometry (VO) is a critical technology in numerous applications, such as autonomous vehicles, robotics, and augmented reality, where the precise and efficient estimation of the camera's trajectory is paramount [3, 4, 5, 7, 8, 9]. This comprehensive review delves into the types and methodologies of VO algorithms, drawing insights from recent papers in the field.

Visual odometry (VO) algorithms have become an integral part of many autonomous systems, including drones, self-driving cars, and robotic systems. These algorithms estimate the motion of a camera in real-time by analyzing a sequence of images, providing crucial information for navigation and control. In this in-depth review, we will explore the types and methodologies of VO algorithms, focusing on feature-based methods, dense visual odometry, and direct visual odometry. We will also delve into the use of RANSAC (RANdom SAmple Consensus) and SURF algorithms in feature-based VO, and discuss the impact of various types of noise on visual data [2].

Feature-based VO algorithms, also known as indirect methods, hinge on the extraction and matching of features between different images. These features, such as corners or edges, are invariant to image scale, rotation, and translation, making them robust to changes in viewpoint. The matched features are then used to estimate the camera's motion. However, feature-based methods are computationally expensive due to the need for feature extraction and matching, and they may struggle in texture-poor environments.

In the context of feature-based VO, the SURF (Speeded Up Robust Features) algorithm is often used to identify and describe distinctive points or features in an image [1]. The SURF algorithm works by detecting and extracting a set of interest points in the image, and then describing the appearance of the points using a feature descriptor. This descriptor is a numerical representation of the appearance of the interest points, which can be used to compare the points to one another. This method is less computationally expensive than dense visual odometry algorithms, as it only requires the analysis of a subset of keypoints or features rather than all pixels in the images. However, it may be less accurate and less robust than dense visual odometry algorithms, as it relies on the assumption that the keypoints remain visible and distinguishable over time. The SURF algorithm is particularly effective in situations where the scene contains repeating patterns, which can often lead to incorrect feature matches [1].

RANSAC (RANdom SAmple Consensus) is another crucial component in feature-based VO. It is an iterative method used to estimate the parameters of a mathematical model from a set of data that may contain outliers or errors [1]. RANSAC is widely used in computer vision and image processing applications, particularly in the context of model fitting and data fitting tasks. The robustness of RANSAC lies in its ability to estimate the parameters of a model even in the presence of significant amounts of noise or outliers. RANSAC is used in conjunction with SURF in the process of feature matching to eliminate incorrect matches and improve the accuracy of the resulting model [1].

Dense visual odometry, on the other hand, estimates the motion by analyzing the dense correspondence between pixels in the images. This can be more accurate than feature-based approaches, but it is also more computationally expensive. Dense visual odometry algorithms are typically more accurate than feature-based approaches, as they analyze the dense correspondence between all pixels in the images rather than just a subset of keypoints or features. This can result in more robust and accurate estimates of the motion of the drone.

Direct visual odometry (DVO) is a type of algorithm that estimates the motion of a camera by directly aligning the images in the image sequence and minimizing the photometric error between them. Unlike feature-based visual odometry algorithms, which extract a set of keypoints or features from the images and track them across successive images, DVO algorithms directly align the intensity values of the pixels in the images. This allows DVO algorithms to make use of the dense information available in the images and produce more accurate motion estimates [4].

The method presented by Angladon et al. [4] presents a VO algorithm for an RGB-D camera, combining both feature-based and direct methods. The algorithm uses a feature-based method for initial pose estimation, followed by a direct method for pose refinement. This hybrid approach leverages the strengths of both methods, achieving robust and accurate pose estimation. The authors highlight the importance of the initial pose estimation in determining the success of the subsequent pose refinement, underscoring the interdependence of the two methods.

Li et al. [5] propose a fully direct VO algorithm for stereo cameras. The algorithm uses a novel cost function that incorporates both photometric and geometric consistency, improving robustness to illumination changes and occlusions. The authors demonstrate the effectiveness of direct methods in challenging environments, such as low-texture and low-light scenarios. They also highlight the importance of geometric consistency in maintaining the accuracy of the estimated motion, particularly in the presence of occlusions.

The paper by Liu et al. [3] provides a comprehensive survey of VO algorithms, covering both feature-based and direct methods. The paper highlights the trade-offs between these methods and discusses recent trends in VO, such as the integration of machine learning techniques. The authors emphasize the importance of considering the specific requirements of the application when choosing a VO algorithm.

The integration of global positional information with visual and inertial measurements in VO algorithms has been underscored in recent studies [7, 8, 9]. Zhang and Scaramuzza [7] present a novel approach to VO that leverages both global positional information and visual and inertial measurements. This methodology, designed for long-term autonomous navigation, uses a tightly-coupled nonlinear-optimization-based estimator to fuse these different types of data. The authors argue that this approach allows for the exploitation of the correlations amongst all the measurements, leading to more accurate and globally consistent estimates.

Arroyo et al. [8] propose a VO algorithm that combines visual, inertial, and global position measurements in a common optimization problem. This tightly-coupled approach considers all measurement correlations, which is crucial for high precision estimates. The authors also highlight the importance of the IMU preintegration algorithm for efficiently deriving the global positional factors. This allows multiple global factors per keyframe in the sliding window with negligible extra computational cost.

Finally, the solution proposed by Zhang et al. [9] presents a tightly-coupled approach for fusing global with visual and inertial measurements in an optimization-based algorithm. The authors use the IMU preintegration method to efficiently derive the global positional error terms. This work demonstrates the potential of tightly-coupled approaches for achieving high-rate locally and globally consistent pose estimates in long-range navigation.

The choice of VO algorithm depends on the specific requirements of the application, including the computational resources available, the accuracy required, and the characteristics of the environment. Feature-based methods, such as those using SURF and RANSAC, are less computationally expensive and can be effective in structured environments, but may be less accurate and sensitive to noise. Dense and direct visual odometry methods can provide higher accuracy and handle larger motions, but are more computationally expensive and may require additional sensors.

In conclusion, VO algorithms have evolved significantly, with different methodologies offering unique advantages and challenges. Hybrid approaches that combine feature-based and direct methods, as well as the integration of machine learning techniques, are promising directions for future research [3, 4, 5, 7, 8, 9].

2.1.2 Camera Selection for VO

Visual Odometry (VO) relies heavily on the type of camera used to capture visual data. The choice of camera can significantly impact the performance of VO algorithms. The three primary types of cameras used in VO are monocular, stereo, and depth cameras, each with its own advantages and disadvantages.

Monocular Cameras

Monocular cameras capture images from a single viewpoint. They are the simplest and most cost-effective option for VO. However, they present a significant challenge in estimating depth, as they lack the stereo disparity that stereo cameras provide. This results in a scale ambiguity problem, where the actual distance of an object from the camera cannot be determined based only on the image. Despite this, monocular VO can still provide valuable information about the relative motion of the camera.

Stereo Cameras

Stereo cameras, consisting of two aligned cameras mimicking human binocular vision, capture two images of the same scene from slightly different viewpoints. This setup allows for the calculation of depth information based on the disparity between the two images. Stereo cameras can provide accurate depth information in a wide variety of lighting conditions, as they rely on geometric principles rather than light intensity. However, they require a textured environment to calculate disparity effectively. In texture-less or highly repetitive patterns, stereo cameras might struggle to find correspondences.

Stereo cameras work on the principle of binocular vision, which is similar to how human eyes perceive depth. They consist of two or more lenses with separate image sensors for each lens. This allows them to capture the same scene from slightly different angles, creating a disparity between the two images. This disparity can be used to calculate depth information.

Advantages of Stereo Cameras in Visual Odometry (VO):

- Stereo cameras can provide accurate depth information in a wide variety of lighting conditions, as they rely on geometric principles rather than light intensity or time of flight.
- They do not require any additional illumination (like IR projectors), making them suitable for outdoor use.
- Stereo cameras can be more cost-effective than depth cameras.

Disadvantages:

- The accuracy of depth estimation depends on the baseline (distance between the two cameras). A larger baseline can provide more accurate depth information, but it also increases the size of the camera setup.
- Stereo cameras require a good texture in the environment to calculate disparity. In texture-less or highly repetitive patterns, they might struggle to find correspondences.

A study by [10] compared the performance of monocular and stereo Fast-SLAM implementations. The results showed that the stereo vision implementation performed significantly better than the monocular one, providing more accurate and robust estimates. This underscores the advantages of stereo cameras in VO applications.

Depth Cameras

Depth cameras, such as the Intel RealSense D435i, use active infrared (IR) stereo technology to calculate depth. They emit a pattern of IR light into the scene, which is then captured by two IR sensors. The displacement of the pattern on the object is used to calculate depth, providing a dense depth map. Depth cameras can work well in texture-less environments, as they do not rely on finding correspondences between images. However, they can be sensitive to lighting conditions and may not work well in outdoor environments or in direct sunlight. They can also have difficulty with reflective or absorbent surfaces, which can distort the IR pattern.

Advantages of Depth Cameras in VO:

- Depth cameras can provide dense depth maps, as they calculate depth for each pixel in the image.
- They can work well in texture-less environments, as they do not rely on finding correspondences between images.
- The Intel RealSense D435i also includes an IMU (Inertial Measurement Unit), which can provide additional data for VO, such as acceleration and angular velocity.

Disadvantages:

- Depth cameras can be sensitive to lighting conditions. They may not work well in outdoor environments or in direct sunlight, as the IR light from the projector can be overwhelmed.
- They can have difficulty with reflective or absorbent surfaces, as these can distort the IR pattern.
- Depth cameras can be more expensive than stereo cameras.

In the context of Visual Odometry, the choice between a monocular, stereo, or depth camera depends on several factors, including the lighting conditions, the texture of the environment, the need for dense depth information, and the budget. Each type of camera offers unique advantages, and the choice should be guided by the specific requirements of the VO application.

2.1.3 Optimal Conditions for VO Algorithms

Visual Odometry (VO) algorithms are crucial for various applications, including robotics and autonomous vehicles. However, their performance can be significantly affected by different types of noise and environmental conditions. This section discusses the optimal conditions for VO algorithms and the types of noise that can impact their performance.

Types of Noise Affecting Visual Data

1. **Image Sensor Noise:** Image sensors in cameras are susceptible to several forms of noise, such as thermal noise, read noise, and shot noise. These can cause variations in the pixel values in the images, affecting the accuracy of the motion estimates produced by VO algorithms.

    - **Thermal Noise:** Also known as Johnson–Nyquist noise, thermal noise is caused by the random movement of charge carriers (like electrons) within a conductor due to thermal agitation. This can cause variations in the pixel values of the images.
    - **Read Noise:** This type of noise is introduced during the readout process of an image sensor. Factors such as electronic noise in the readout circuit, charge leakage, and cross-talk between pixels cause read noise.
    - **Shot Noise:** Shot noise is caused by the random fluctuations in the number of charge carriers (like electrons) detected by the image sensor. This noise is typically more pronounced at low light levels.

2. **Changes in Lighting Conditions:** Variations in the scene's lighting conditions can also affect the visual data used by the algorithm.

3. **Reflections and Occlusions:** Reflective surfaces or objects that occlude parts of the scene can cause variations in the visual data.

4. **Motion Blur:** If the camera or the objects in the scene are moving too quickly, the images may be blurry.

5. **Distortion:** Distortions in the images, such as lens distortion or perspective distortion, can affect the accuracy of VO algorithms.

   VO algorithms perform best under stable lighting conditions, with minimal occlusion and distortion. The scene should ideally contain a variety of textures and features that can be easily tracked across multiple frames. Rapid movements or rotations can cause motion blur, which can lead to errors in feature tracking and pose estimation. Therefore, slower camera movements are generally more favorable for VO.

Computational Cost of VO Algorithms

The computational cost of VO algorithms can be quite high, especially for methods that use dense image data or perform global optimization. The computational cost is primarily determined by the following factors:

- **Image Resolution:** Higher resolution images contain more pixels and thus more potential features to track, increasing the computational cost.
- **Number of Frames:** Processing more frames per second (FPS) increases the amount of data that the algorithm needs to handle, thereby increasing the computational cost.
- **Algorithm Complexity:** More complex algorithms, such as those that perform global optimization or use sophisticated models for motion and structure, can have higher computational costs.
- **Hardware Capabilities:** The performance of the hardware (e.g., CPU, GPU, memory) on which the VO algorithm runs can significantly impact the computational cost. Faster and more powerful hardware can handle higher computational loads, allowing for more complex algorithms or higher resolution images to be used.

Challenges and Limitations of VO Algorithms

Despite their usefulness, VO algorithms face several challenges and limitations. For instance, they are sensitive to image noise, lighting conditions, and rapid movements, as discussed above. They also suffer from drift, where small errors in the estimated motion accumulate over time, leading to larger errors in the estimated trajectory. Furthermore, VO algorithms typically assume a rigid world, which may not hold true in dynamic environments with moving objects.

VO in GNSS-Denied Environments

In environments where Global Navigation Satellite System (GNSS) signals are unavailable or unreliable (e.g., indoors, underwater, or in urban canyons), VO provides a valuable means of estimating the camera's motion. However, the challenges and limitations mentioned above can be more pronounced in such environments. For instance, indoor environments may have more occlusions and dynamic objects, while underwater environments can have low-light conditions and significant image distortion.[11]

Moving features, like a person walking close to the camera, can affect the accuracy of VO algorithms for estimating the position. In such cases, the YOLOv3 algorithm can be adopted [12]. YOLOv3 (You Only Look Once version 3) is a real-time object detection algorithm based on convolutional neural networks (CNNs). It is designed to be fast and accurate, making it suitable for real-time object detection applications such as drone navigation and video surveillance. YOLOv3 works by dividing the input image into a grid of cells and predicting the class and location of objects within each cell. The CNN, trained on a large dataset of annotated images, is used to extract features from the input images and classify and localize the objects in the image.

2.2 Data Fusion Techniques

As shown in the papers [7, 8, 9], fusing visual odometry with Inertial Measurement Unit (IMU) data can be highly beneficial for robust visual-inertial estimation.

Data fusion is a critical aspect of attitude estimation, which involves the combination of data from multiple sensors to compute the estimation of the state variable and the environment. Various techniques can be employed for this purpose, including Kalman Filters, Complementary Filters, and Particle Filters. These filters are valuable approaches for data fusion and outlier elimination Pre-processing operations are essential to improve the quality of sensor measurements. For instance, the raw data from inertial sensors can be pre-processed using an adaptive variable bandwidth filtering via sinusoidal data estimation. This technique guarantees less computational demand, making the filter suitable for real-time applications

Sensor calibration is another crucial operation to enhance the accuracy of sensor measurements. The calibration process is critical for both magnetometers and accelerometers. One approach is to use after-production calibrated sensors, although this may increase the cost

In conclusion, data fusion techniques play a pivotal role in enhancing the performance of attitude estimation systems. By effectively combining data from various sensors and implementing necessary pre-processing operations, it is possible to achieve more accurate and reliable estimations. Pre-processing operations are essential to improve the quality of sensor measurements. Even after a thorough calibration, it is crucial to remove the initial bias from the data coming from IMU and LiDAR. This is typically achieved by computing the mean of the initial values from the sensor data, which is then subtracted from the respective data sets. Furthermore, to smooth out the fluctuations in the sensor readings and reduce the noise, a moving average filter is applied. This filter works by taking the average of a certain number of points from the data set to produce each point in the resulting output. These pre-processing techniques help to enhance the quality of the sensor data, thereby improving the overall performance of the data fusion system.

2.2.1 Extended Kalman Filter

The Extended Kalman Filter (EKF) is a variant of the Kalman Filter, which is a recursive algorithm used for estimating the evolving state of a process in a way that minimizes the mean squared error. The EKF is particularly useful when the system is nonlinear, as it linearizes the system dynamics around the current estimate at each time step.

The EKF consists of two main steps: prediction and update. The prediction step uses the system dynamics to predict the state at the next time step, while the update step uses the new measurement to correct the predicted state.

The prediction step is given by:

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \tag{2.1}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2.2}$$

where $\hat{x}_{k|k-1}$ is the predicted state, f is the system dynamics function, u_k is the control input, $P_{k|k-1}$ is the predicted covariance, F_k is the Jacobian of f with respect to the state, and Q_k is the process noise covariance.

The update step is given by:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \tag{2.3}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - h(\hat{x}_{k|k-1})) \tag{2.4}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{2.5}$$

where K_k is the Kalman gain, H_k is the Jacobian of the measurement function h with respect to the state, R_k is the measurement noise covariance, z_k is the measurement, and I is the identity matrix.

The robot_localization ROS package is a highly versatile tool that offers a suite of state estimation nodes, including those based on the Extended Kalman Filter (EKF). This package is designed to provide a comprehensive and flexible solution for robotic state estimation by allowing for the fusion of an arbitrary number of sensors, each of which may be providing data at different rates.

2.2.2 Dead Reckoning

Dead reckoning is a method of estimating the current position of a vehicle, such as a UAV (Unmanned Aerial Vehicle), by using a previously known position and advancing that position based upon known or estimated speeds over elapsed time, and course direction.

Working Principle

The fundamental principle behind dead reckoning is the use of motion information to estimate the change in position. The process begins with an initial position and then advances that position based on known or estimated speeds and the direction of travel over elapsed time. This is mathematically represented as:

$$newposition = oldposition + velocity \times time \tag{2.6}$$

where velocity is the speed of the vehicle in the direction of travel.

Inertial Navigation Systems and Dead Reckoning

Inertial Navigation Systems (INS) often use dead reckoning to estimate the position of a vehicle. INS uses the principle of dead reckoning, integrating information from motion sensors (accelerometers) and rotation sensors (gyroscopes) to calculate changes in position over time.

The accelerometers measure the accelerations of the vehicle, which, when integrated over time, give the velocity. The velocity, when integrated over time, gives the position. The gyroscopes measure the angular rates, which, when integrated over time, give the orientation of the vehicle. The paper "Dead Reckoning of a Mobile Robot in 2-Dimensional Special Euclidean Group" by Da Bin Jeong and Nak Yong Ko [13] presents a unique perspective on the use of dead reckoning for mobile robots. The authors argue that dead reckoning is indispensable for estimating the pose (location and attitude) of a mobile robot. If there is no information on localization except for the velocity of the mobile robot, dead reckoning is the only way of pose estimation.

The paper proposes the use of Lie theory for dead reckoning and formulation of motion model. The method represents the pose of a robot as an element in a Lie group called the special Euclidean group, SE(2), and the increment of the pose as an element of se(2) that is the Lie algebra corresponding to SE(2).

The authors argue that proper formulation of dead reckoning is essential to keep the pose estimation convergent and consistent. Even though dead reckoning inevitably incurs error, the authors believe that their proposed method using Lie theory provides exact prediction of robot location and attitude regardless of the sampling period.

The paper concludes that the proposed method can be extended to 3-dimensional space dead reckoning just by extending the SE(2) to SE(3). The method and the proposed formulation can be used for predicting the pose at the prediction step of KF based method. Also, it can be utilized for application of KF in Lie group where linearization of the motion model is required. The Lie group theory facilitates linearization of model and derivation of covariance matrix in Lie algebra. It improves the convergence and robustness of the KF application.

Dead Reckoning in UAVs: A Case Study

In the context of UAVs, dead reckoning is particularly useful when GPS signals are unavailable or unreliable. The UAV uses the data from its onboard INS to estimate its current position relative to a known starting point. This involves measuring the acceleration and angular velocity of the UAV, integrating these values to calculate velocity and change in orientation, and then integrating again to estimate the change in position.

However, dead reckoning is subject to cumulative errors. Small errors in the measurement of acceleration or angular velocity can lead to increasingly large errors in the estimated position over time. To mitigate this, UAVs often use sensor fusion techniques, combining the INS data with GPS data (when available) and other sensor data to improve the accuracy of the estimated position.

The paper "Dead Reckoning of a Fixed-Wing UAV with Inertial Navigation Aided by Optical Flow" by Lorenzo Fusini, Tor A. Johansen, and Thor I. Fossen [14] discusses the use of dead reckoning for UAV navigation in detail. The paper presents experimental results for dead reckoning of a fixed-wing UAV using a nonlinear observer (NLO) and a more recent tool called the eXogenous Kalman Filter (XKF). The sensors used for this purpose include an IMU (accelerometers, inclinometers, and rate gyros), a camera, and an altimeter. The observed states are position, velocity, and attitude.

A machine vision system provides the body-fixed velocity of the UAV. This calculated velocity, although affected by a bias, is necessary for estimating the attitude and for bounding the rate of divergence of the position during dead reckoning.

Gyro, accelerometer, and optical flow (OF) velocity biases are estimated, but only as long as GNSS is available. When dead reckoning begins, these biases are frozen at their last calculated value.

The experimental results show that the position error grows at a bounded rate with the proposed estimators. This is important as it helps to understand the accuracy of the dead reckoning approach.

The paper tests an NLO and an XKF for dead reckoning on experimental data. A machine vision system calculates the body-fixed linear velocity of the UAV using optical flow. This velocity is used both as a vector for the injection term of the NLO, and as a correction term, combined with the estimated attitude, for the estimated NED velocity. The results show that the inclusion of compensation for the additional biases and for the rotation of Earth help reduce the position error of the NLO, and that the XKF can reduce the error even further by providing a better estimate of the velocity that is less dependent than the NLO on the optical flow velocity bias.

Chapter 3

Building the Drone and Software Setup

3.1 Drone Frame and Components

The design and assembly of a drone involve the careful selection and integration of various components, each playing a crucial role in the drone's overall performance and stability. The choice of these components is particularly important when implementing and testing algorithms such as visual odometry, where data from various sensors must be accurately fused to provide reliable results.

The equilibrium and control of an unmanned aerial vehicle (UAV) hinge on the appropriate integration of a multitude of sensors. The application of certain sensors, including the accelerometer, magnetometer, and gyroscope, has been theoretically explored in previous sections of the literature review. To facilitate a more practical comprehension, this section offers an in-depth examination of the primary components typically incorporated in drones, emphasizing those currently accessible in the market.

3.1.1 Drone Frame

The drone frame serves as the backbone of the UAV, providing the necessary structure to house the various components and systems. It plays a crucial role in the overall performance and stability of the drone. The choice of the drone frame configuration significantly influences the drone's flight capabilities, payload capacity, and resilience to failures.

One of the popular configurations is the Octocopter, specifically the OCTO QUAD X8 configuration. This configuration features eight motors and propellers arranged in a coaxial setup, with two motors on each arm of the quad frame. The benefits of this configuration are multifold:

- **Enhanced Lift and Payload Capacity:** The OCTO QUAD X8 configuration, with its eight motors, can generate a significant amount of lift. This makes it capable of carrying heavier payloads, which is beneficial for applications that require additional equipment, such as cameras or sensors.
- **Improved Stability and Control:** The Octocopter configuration offers better stability and control during flight, especially in windy conditions. The additional propellers provide more control points, allowing for more precise maneuvers.
- **Redundancy and Reliability:** The OCTO QUAD X8 configuration provides redundancy in the event of a motor failure. If one motor fails, the drone can still maintain control and land safely, making it a reliable choice for critical applications.
- **Versatility:** The Octocopter configuration is versatile and can be used in a wide range of applications, from aerial photography to surveying and mapping, due to its stability and payload capacity.

In conclusion, the OCTO QUAD X8 configuration offers a balance of power, stability, and reliability, making it a suitable choice for a wide range of UAV applications.

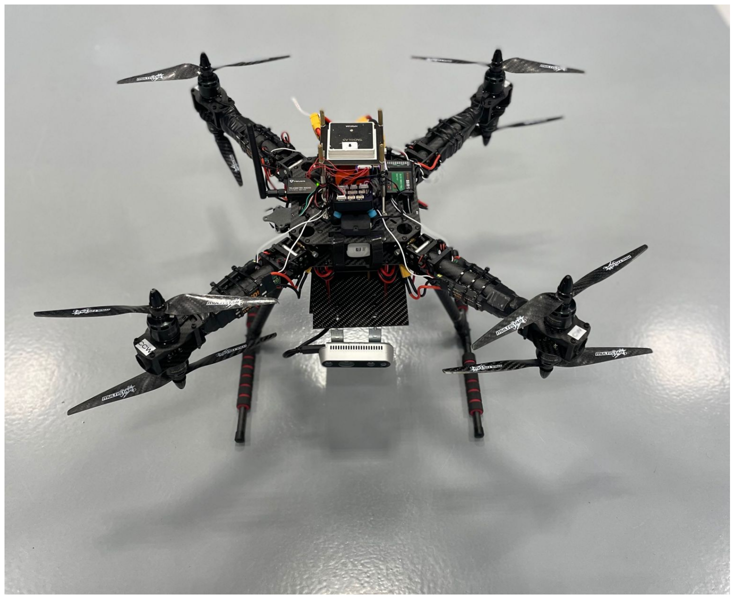

Figure 3.1: Photo of the drone

3.1.2 Cube Orange

The Cube Orange, also known as CubePilot, is a highly versatile and powerful autopilot system designed for unmanned vehicles. It is built on the robust and open-source ArduPilot software, which provides a solid foundation for a wide range of applications. The Cube Orange offers several benefits that make it an excellent choice for UAV systems:

- **Advanced Sensors:** The Cube Orange includes a suite of advanced sensors, including a triple redundant IMU system and a barometer. These sensors provide accurate and reliable data for navigation and control.

- **Versatility:** The Cube Orange supports a wide range of vehicle types, from multirotors and fixed-wing aircraft to rovers and boats. This versatility makes it a suitable choice for various UAV applications.

- **Expandability:** The Cube Orange features numerous I/O ports and interfaces, allowing for the integration of additional sensors, actuators, and communication devices. This expandability makes it adaptable to specific application requirements.

- **Safety Features:** The Cube Orange incorporates several safety features, such as a built-in vibration isolation system and a dedicated failsafe co-processor. These features enhance the reliability and safety of the UAV system.

- **Open-Source Software:** The Cube Orange is built on the ArduPilot software, which is open-source and has a large and active community. This not only ensures continuous development and improvement of the software but also provides users with extensive resources and support.

In summary, the Cube Orange offers a powerful, versatile, and reliable solution for UAV autopilot systems. Its advanced features and capabilities make it a suitable choice for a wide range of UAV applications, from hobbyist projects to professional and industrial applications.

Figure 3.2: Cube Orange

3.1.3 Jetson Nano

The Jetson Nano Developer Kit is a powerful, compact AI computer from NVIDIA that provides the compute performance to run modern AI workloads and is highly suitable for visual odometry algorithms. It offers several benefits that make it an excellent choice as a companion computer for UAV systems:

- **High Performance:** The Jetson Nano is equipped with a quad-core ARM Cortex-A57 MPCore processor and a 128-core NVIDIA Maxwell GPU. This combination provides substantial computational power to handle demanding tasks, including the processing and analysis of visual data for odometry algorithms.
- **AI Capabilities:** The Jetson Nano is designed to support AI workloads. It can run multiple neural networks in parallel to process data and make decisions in real-time. This capability is particularly useful for implementing advanced navigation and control algorithms based on machine learning.
- **Energy Efficiency:** Despite its high performance, the Jetson Nano is energy-efficient, making it suitable for battery-powered UAV applications. It offers several power modes, allowing users to choose the right balance between performance and power consumption.
- **Compact Size:** The Jetson Nano Developer Kit is compact and lightweight, which is a critical factor for UAV applications. Its small footprint makes it easy to integrate into a drone frame without significantly affecting the vehicle's weight or balance.
- **Software Support:** The Jetson Nano supports a wide range of software libraries and frameworks, including CUDA, cuDNN, and TensorRT. These tools make it easier to develop and optimize AI algorithms. Additionally, it supports the Robot Operating System (ROS), which is widely used in the robotics community.
- **Community and Resources:** NVIDIA provides extensive documentation and resources for the Jetson Nano, and there is a large and active community of developers who use and support this platform. This ensures that users can find help and resources to overcome development challenges.

The Jetson Nano Developer Kit offers a powerful and efficient solution for running visual odometry and other AI algorithms on UAVs. Its high performance, AI capabilities, and software support make it a suitable choice for advanced UAV applications.

Figure 3.3: Nvidia Jetson nano

3.1.4 Intel RealSense d435i

The Intel RealSense d435i is a depth camera that combines a stereo vision system with an inertial measurement unit (IMU). This combination allows the camera to capture high-resolution 3D depth maps while also tracking its own motion, making it an excellent choice for visual odometry applications.

The camera's stereo vision system consists of two infrared cameras and an infrared projector. This setup allows the camera to perceive depth by triangulating the distance to objects in its field of view, similar to how human eyes work. The system can capture depth data at a resolution of up to 1280x720 pixels, with a maximum range of approximately 10 meters.

The integrated IMU consists of a 3-axis accelerometer and a 3-axis gyroscope. These sensors allow the camera to track its own motion in 3D space, providing valuable data for odometry and SLAM algorithms. The IMU data can be used to correct for motion blur in the depth images, improving the accuracy of the depth data.

The RealSense d435i also supports hardware synchronization, allowing it to be used in multi-camera setups. This feature can be useful for applications that require a wider field of view or more detailed depth data.

The camera is supported by the Intel RealSense SDK, which provides tools and libraries for capturing and processing depth data. The SDK also includes algorithms for aligning the depth and color data, filtering the depth data, and performing other processing tasks.

Its combination of high-resolution depth sensing and motion tracking capabilities make it well-suited for implementing visual odometry algorithms, as discussed in Section 2.1.2. The camera's capabilities, combined with the support provided by the RealSense SDK, make it a versatile and capable component for any UAV system.

Figure 3.4: Intel RealSense d435i

3.1.5 Lidar

Lidar, an acronym for Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure distances. The TFmini Lidar is a single-point micro ranging module designed for use in UAVs and other systems where accurate distance measurement is required.

The TFmini Lidar operates by emitting a short pulse of infrared light and then measuring the time it takes for the light to return after bouncing off an object. This time, known as the time-of-flight, is then used to calculate the distance to the object.

In the context of a UAV, the TFmini Lidar is typically mounted on the underside of the drone and pointed towards the ground. This allows the Lidar to measure the height of the drone above the ground, providing valuable data for maintaining a stable altitude and for landing procedures.

The TFmini Lidar is capable of measuring distances from 30 cm to 12 m with a resolution of 1 cm, making it a highly accurate tool for altitude measurement. It operates at a frequency of 40 Hz, providing real-time distance data for the UAV's control system.

Figure 3.5: TFmini Lidar

3.1.6 Optional sensors and Hardware

Ultrasonic sensors present an affordable alternative to Lidars for providing drones with environmental awareness, albeit with a significantly restricted operational range. The HC SR-04 chip, for instance, operates by emitting a high-frequency sound wave (40 kHz) through one of its piezoelectric transducers. It then detects the echo or the returning pulse and converts it into a corresponding voltage fluctuation.

However, the performance of ultrasonic sensors is influenced by several factors. Light conditions, for example, can significantly impact the sensor's effectiveness, making it a critical parameter in both indoor and outdoor environments. Additionally, the reflectivity of the material off which the sound wave bounces can also affect the sensor's readings.

Other factors that can potentially distort the measurements include ambient noise, temperature, and humidity. Given these limitations, ultrasonic sensors are not typically used as the primary device for obstacle avoidance or altitude measurement. However, they could potentially be implemented to prevent lateral collisions with unobserved objects.

Figure 3.6: HC SR-04 sensor

Beyond the earlier mentioned sensors, a sound alert system and a protective toggle are also utilized. While these components aren't indispensable for the drone and its self-governing capabilities, they significantly enhance the user's interaction with the device and comprehension of its varying flight patterns.

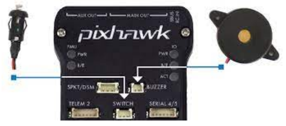

Figure 3.7: Buzzer and safety switch

3.2 Software Installation and Configuration

The software installation and configuration process for the drone involved several intricate steps, each of which was pivotal to ensure the successful operation of the drone and the implementation of the visual odometry algorithm. The process commenced with the hardware setup, which entailed connecting the Cube Orange to the Jetson Nano via USB and the Lidar to the GPS2 port on the Cube Orange. The Jetson Nano was then connected to a display using an HDMI cable, and a mouse, keyboard, and ethernet cable were also connected to facilitate the software setup process.

The software setup process was initiated with the installation of Ubuntu 20.04 on the Jetson Nano. This operating system was selected due to its compatibility with ROS Noetic, the framework utilized for the drone's software. The installation of ROS Noetic was executed through the terminal.

To enable communication between the Cube Orange and the Jetson Nano, MAVROS and MAVLink were installed. MAVROS is a ROS package that provides MAVLink protocol capabilities, which is essential for drone communication.

The Cube Orange was then configured following the steps outlined in the previous chapter to establish a MAVROS connection. This involved learning to use ROS and Mavros, as well as Python to write custom scripts that subscribe and publish to and from different ROS topics.

The telemetry setup involved connecting the Holybro SiK Telemetry Radio V3 to the PC and configuring the COM port and baud rate in Mission Planner. Mission Planner was used as the ground station control, which is a comprehensive open-source software for programming and controlling autonomous vehicles. It provides a user-friendly interface for setting up and tuning the vehicle, planning and tracking missions, and monitoring the vehicle status and sensor data in real-time. The SERIAL0 BAUD and SERIAL0 PROTOCOL were set to match the baud rate and MAVLink communication protocol, respectively.

To ensure that data from the Cube Orange was transmitted to the companion computer at a high frequency, the SR0 ADBS to SR0 RAW SENSE parameters in Mission Planner were set to 50Hz.

The connection between the Jetson Nano and the Cube Orange autopilot was established, user permissions were configured, and MAVROS was launched. The connection was verified, and the user account on the Jetson Nano was given access to serial devices by adding the user to the dialout group.

The Intel RealSense Camera was set up with ROS Noetic on the Jetson Nano. This involved updating the system and installing dependencies, installing the librealsense2 SDK, and setting up the ROS Wrapper for Intel RealSense Devices.

The RTAB-Map Package in ROS Noetic was set up, followed by the setup of the Robot Localization Package. These packages were crucial for implementing the visual odometry algorithm and ensuring the drone's stable flight and accurate data collection.

Figure 3.8: Holybro SiK Telemetry Radio V3

3.3 Reading Data from IMU, Lidar, and Camera on ROS Topics

In the context of our drone setup, the Cube Orange, Lidar, and the Intel RealSense Camera are all significant sources of data, each publishing information on different ROS topics at varying frequencies.

The Inertial Measurement Unit (IMU) on the Cube Orange publishes data at a frequency of 50Hz on the ROS topic `/mavros/imu/data`. The IMU data is of the message type `sensor_msgs/Imu`, which includes orientation represented in quaternions, angular velocity in rad/s, and linear acceleration in x, y, and z directions. This data is crucial for understanding the drone's motion and orientation in space.

The Lidar sensor, which is used for measuring the distance to the ground, publishes data at a frequency of 20Hz on the ROS topic `/mavros/distance_sensor/rangefinder_pub`. The Lidar data is of the message type `sensor_msgs/Range`, which includes the range measurement.

The Intel RealSense Camera, a depth camera, publishes data on several ROS topics to which the RTAB-Map package subscribes. These topics include
`/camera/color/image_raw`, `/camera/depth/image_rect_raw`, and `/camera/color/camera_info`, among others. The camera data is of the message type `sensor_msgs/Image` and `sensor_msgs/CameraInfo` for the image data and camera metadata respectively.

The RTAB-Map package, which is used for generating the drone's map and estimating its position, publishes odometry data at a frequency of approximately 3Hz on the ROS topic `/rtabmap/odom`. The odometry data is of the message type `nav_msgs/Odometry`, which includes the drone's pose and orientation represented in quaternions.

In order to activate each of these components and start the data flow, custom launch files are used. These launch files are executed in the ROS environment and they initiate the respective nodes for the Cube Orange, the Intel RealSense Camera, and the RTAB-Map package.

To launch the RealSense camera node, the following command is used in a terminal:

roslaunch realsense2_camera rs_camera.launch align_depth:=true color_width:=848 color_height:=480 depth_width:=848 depth_height:=480 color_fps:=60 depth_fps:=60

This command starts the RealSense camera node and sets the parameters for depth alignment, color and depth image dimensions, and the frame rate, which is set to 60Hz.

To launch the RTAB-Map node, navigate to the workspace root in a new terminal and use the following command:

```
roslaunch my_rtabmap_launch rtabmap_d435i.launch
```

Finally, to launch the IMU data publisher node for the Cube Orange, the following command is used in another terminal:

```
roslaunch cube_orange cube_orange_imu.launch
```

3.4 RTAB-Map ROS Algorithm for State Estimation

RTAB-Map, an acronym for Real-Time Appearance-Based Mapping, is a sophisticated graph-based SLAM (Simultaneous Localization and Mapping) methodology. This appearance-based SLAM paradigm leverages data harvested from vision sensors to localize the robot and map its surroundings. The algorithm employs a process known as loop closures to ascertain whether a location has been previously encountered by the drone. As the UAV moves into unexplored areas, the map grows, and the number of images that each new image must be compared to escalates. This results in an increase in the time taken for loop closures, with complexity growing linearly. However, RTAB-Map has been optimized for large-scale and long-term SLAM, employing a variety of strategies to ensure real-time loop closure detection. Loop closure detection is performed quickly enough to produce results before the next set of camera images is acquired.

The Dual Structure of RTAB-Map: Front End and Back End

As shown in the article by Shiva Chandrachary [15] front end of RTAB-Map is primarily concerned with the sensor data used to obtain the constraints that are utilized for feature optimization approaches. In RTAB-Map, landmark constraints are not employed. The focus is solely on odometry constraints and loop closure constraints. The odometry constraints can be derived from a variety of sources such as wheel encoders, IMU, LiDAR, or visual odometry. Visual odometry is accomplished using 2D features such as Speeded Up Robust Features (SURF).

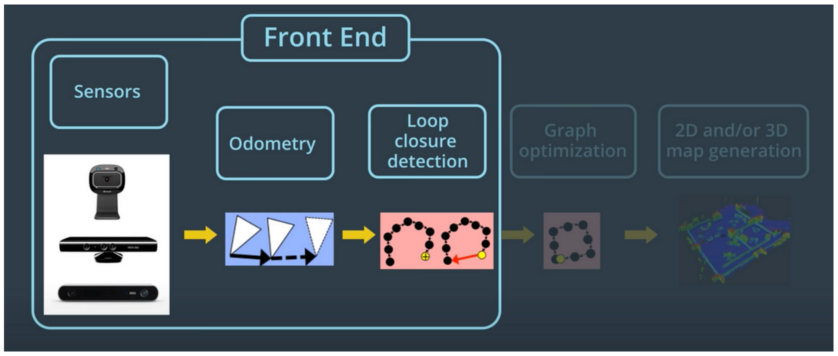

Figure 3.9: Front End of RTAB-Map

The back end of RTAB-Map encompasses graph optimization and the assembly of an occupancy grid from the data of the graph.

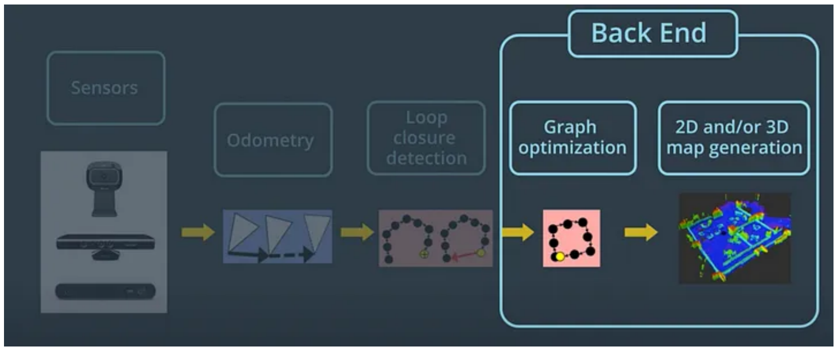

Figure 3.10: Back End of RTAB-Map

RTAB-Map is a robust and efficient RGB-D (Red Green Blue - Depth) Graph-Based Simultaneous Localization and Mapping (SLAM) approach. This algorithm is underpinned by an incremental appearance-based loop closure detector, which employs a bag-of-words approach to determine if a new image captured by the robot matches any previous images. If a loop closure hypothesis is validated, a new constraint is added to the map's graph, and a graph optimizer minimizes the errors in the map.

The RTAB-Map algorithm can be bifurcated into two main components:

1. **Mapping:** The algorithm fabricates a 3D map of the environment using the RGB-D data. It employs a graph-based approach, where each node in the graph represents a place that the robot has visited, and each edge represents the spatial relation between places.

2. **Localization:** The algorithm estimates the robot's pose within the map. It employs a particle filter to track the robot's pose, and it uses the loop closure detector to correct the robot's drift over time.

One of the salient advantages of RTAB-Map is its capability to handle large-scale environments. It employs a memory management approach to maintain the number of locations and the map size constant over time. This makes it suitable for long-term operation in large environments.

RTAB-Map is an excellent choice for autonomous vehicles due to its robustness and efficiency. It can handle dynamic environments and recover from localization failures. Moreover, it has been integrated with the Robot Operating System (ROS), making it easy to use with a variety of robots and sensors.

In RTAB-Mapping, loop closure detection employs a bag-of-words approach. Features, or specific characteristics of an image, are extracted using a method called Speeded Up Robust Features (SURF). Each feature has a unique descriptor, and these descriptors are compared to a vocabulary for faster processing. This vocabulary is a collection of similar features or synonyms. When a feature descriptor matches one in the vocabulary, it's referred to as a visual word. An image becomes a bag-of-words when all its features are quantized. Each word is linked to the images it's associated with, making image retrieval more efficient.

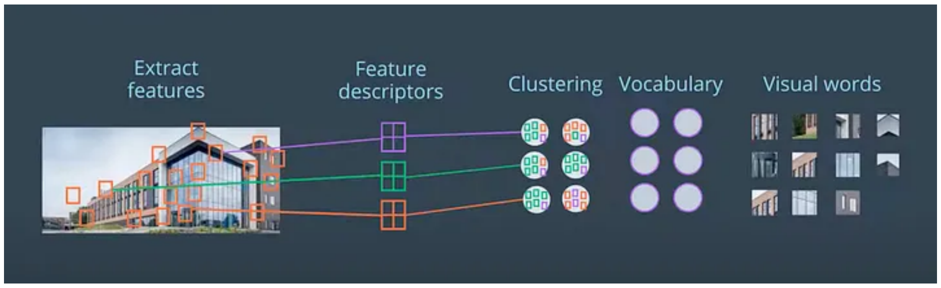

Figure 3.11: Bag-of-Words approach

To compare a new image with previous ones, a matching score is given to all images containing the same words. If a word is seen in an image, the score of this image increases. If an image shares many visual words with the new image, it scores higher. A Bayesian filter evaluates these scores, and if the hypothesis that an image has been seen before reaches a certain threshold, a loop closure is detected.

RTAB-Map also employs a memory management technique to limit the number of locations considered during loop closure detection. The most recent and frequently observed locations are kept in the robot's Working Memory (WM), while others are transferred into Long-Term Memory (LTM). When a new image is acquired, a new node is created in the Short Term Memory (STM), and nodes are weighted based on how long the robot spent in the location. When STM reaches its capacity, the oldest node is moved to WM for loop closure detection. Loop closure happens in the WM, and when the time required to process new data reaches a certain limit, some nodes are transferred from WM to LTM, keeping the WM size nearly constant. If loop closure is detected, neighbors in LTM of an old node can be transferred back to the WM.

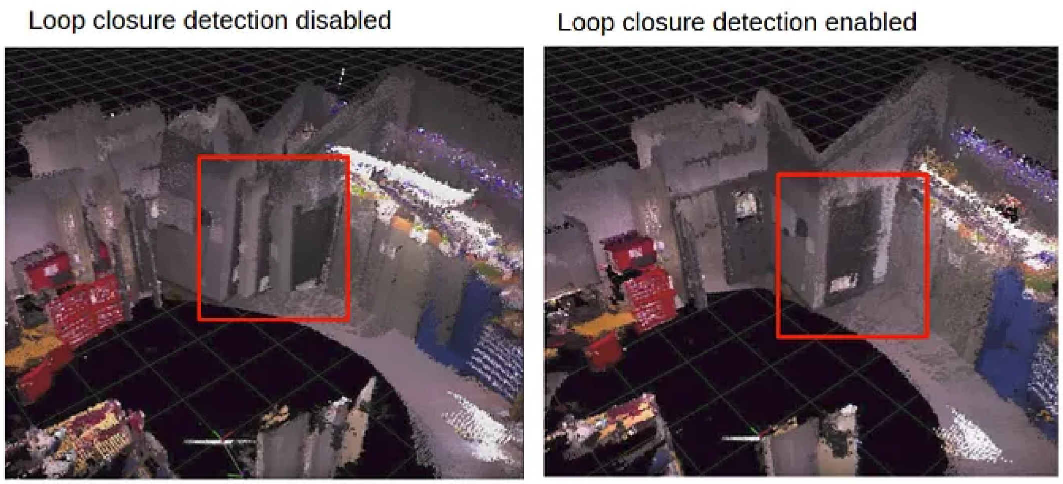
Loop closure detection disabled
Loop closure detection enabled

Figure 3.12: Loop closure disabled/enabled

Chapter 4

RTAB-Map Parameters and Tuning

In Chapter 4, we delve into the crucial aspects of RTAB-Map parameters and tuning, building upon the foundation laid out in section 3.4. As we have previously explored, RTAB-Map is an essential component of the ROS (Robot Operating System) framework, enabling powerful visual odometry capabilities for a variety of applications.

To harness the full potential of RTAB-Map and achieve optimal performance, it becomes essential to fine-tune its parameters. This chapter focuses on the various factors that significantly impact the accurate functioning of RTAB-Map in ROS. By understanding and appropriately adjusting these parameters, you can enhance the results and outcomes of your projects.

Furthermore, the performance of RTAB-Map can be further improved by adjusting the camera settings. By fine-tuning the camera specifications, you can unlock even better visual odometry for UAVs or other relevant applications.

Throughout this chapter, we will explore how RTAB-Map parameters function, highlighting their significance in generating robust visual odometry. Additionally, we will discuss how these parameters have been modified and optimized specifically for our project. Furthermore, we will delve into the adjustments made to the camera specifications to achieve superior visual odometry for UAVs.

By the end of this chapter, you will have a comprehensive understanding of RTAB-Map parameters, their impact on performance, and how they can be tailored to suit your specific project requirements.

4.1 Understanding RTAB-Map Parameters

The RTAB-Map algorithm operates based on a set of parameters, each of which influences the behavior of the algorithm. Understanding these parameters is crucial for effectively tuning the algorithm to suit specific applications. The parameters are defined in the launch file and can be modified to adjust the performance of the algorithm. Here, we will discuss some of these parameters in detail, following the guide available at http://wiki.ros.org/rtabmap$_{r}os/Tutorials/Advanced$

- **fcu url**: This parameter defines the port at which the flight control unit (FCU) is connected. The default value is set to /dev/ttyACM0:57600, but this may need to be changed according to your setup.
- **tgt system and tgt component**: These parameters define the MAVLink target system and component ids respectively.
- **pluginlists yaml and config yaml**: These parameters define the paths to the yaml files for the MAVROS plugins list and configuration.
- **Odom/Strategy**: Determines the odometry approach to use. Two strategies are available:
  - Frame-to-Map (0): The current frame is matched directly with the global map. This method provides more accurate and globally consistent pose estimation but is computationally expensive.
  - Frame-to-Frame (1): The current frame is matched only with the previous frame. This method is computationally efficient but more prone to drift and inconsistency.
- **Vis/MaxFeatures**: This parameter defines the maximum number of features (keypoints) to extract from each image. Reducing this number can speed up processing time.
- **Rtabmap/DetectionRate**: This parameter controls the frequency of map updates. Reducing this value can lighten the computational load.
- **Vis/MinInliers**: This parameter defines the minimum number of inlier matches required for successful registration. Reducing this number can potentially increase the speed of the algorithm.

- **Vis/InlierDistance**: This parameter defines the inlier distance threshold used in the RANSAC algorithm for pose estimation.
- **Odom/ImageDecimation**: This parameter controls the decimation of the image, effectively reducing its size and speeding up processing.
- **Odom/MaxDepth**: This parameter defines the maximum depth value considered by the algorithm.
- **camera topic, depth topic, and camera info topic**: These arguments set the respective topics for the color image, depth image, and camera info.
- **approx sync**: This argument determines whether the synchronization between RGB and depth images is approximately done or not.

In the context of RTAB-Map, inliers are the set of matches that are consistent with the estimated transformation. When two images are matched, RTAB-Map extracts features from both images and finds correspondences. These correspondences are then used to estimate the transformation between the images. However, not all correspondences are correct due to noise, occlusion, and other factors. The RANSAC algorithm is used to robustly estimate the transformation by identifying inliers, which are correspondences that are consistent with the estimated transformation, and outliers, which are not. The Vis/MinInliers parameter controls the minimum number of inliers required for a successful registration, and the Vis/InlierDistance parameter defines the threshold for determining whether a correspondence is an inlier or an outlier.

4.2 Tuning Parameters for Speed and Accuracy

Tuning the parameters of the RTAB-Map algorithm involves making tradeoffs between speed and accuracy. The goal is to find a balance that best suits the specific application. Here, we will discuss how to adjust some key parameters to optimize the speed of the algorithm while maintaining an acceptable level of accuracy.

In the custom launch file, several parameter values have been modified to optimize the algorithm's speed. These changes include:

- **rtabmap/Odom/Strategy**: Initially set to 1, signifying Frame-to-Frame odometry.

- **rtabmap/Vis/MaxFeatures**: Adjusted from the default value of 500 to 200. This reduces the number of keypoints extracted from each image, thereby speeding up the processing time.

- **rtabmap/Rtabmap/DetectionRate**: Enhanced from the default 2 to 5. This reduces the frequency of map updates, which can lighten the computational load.

- **rtabmap/Vis/MinInliers**: This parameter is now set to 5. This requires fewer inlier matches for successful registration, which can potentially increase the speed of the algorithm.

- **rtabmap/Vis/InlierDistance**: This parameter, set to 0.1, defines the inlier distance threshold used in the RANSAC algorithm for pose estimation.

- **rtabmap/Odom/ImageDecimation**: Increased from 2 to 4 to reduce the image size and speed up processing.

- **rtabmap/Odom/MaxDepth**: Kept at the default of 4.0, maintaining the maximum depth value considered by the algorithm.

The `Vis/CorType` parameter, though not explicitly set in the custom launch file, plays a crucial role in the RTAB-Map algorithm. It determines the method used to find correspondences between keypoints in the images. The options are:

- **Features Matching (0)**: In this method, keypoints are extracted and described using feature descriptors. The descriptors are then matched between consecutive frames to find correspondences. This method tends to be more robust as it relies on distinctive descriptors to find matches.

- **Optical Flow (1)**: This approach estimates the motion of individual pixels or keypoints between consecutive frames by analyzing the intensity pattern of the images. This method can provide more matches as it tracks keypoints continuously, even when their appearance changes.

However, it may be less robust than Features Matching, as it relies on the assumption that pixel intensities remain constant between consecutive frames.

In addition to the RTAB-Map parameters, the settings for the Intel RealSense camera were also adjusted in the launch file to optimize the performance of the mapping algorithm. These changes include:

- **align_depth**: This parameter is set to true to ensure that the depth frame is aligned with the color frame. This alignment is crucial for accurate depth perception and feature matching in the RTAB-Map algorithm.

- **color_width, color_height, depth_width, depth_height**: These parameters control the resolution of the color and depth images. They have been reduced from their default values to speed up image processing. Lower resolution images require less computational resources to process, which can significantly improve the speed of the mapping algorithm.

- **color_fps, depth_fps**: These parameters control the frame rate of the color and depth images. They have been set to 60 frames per second (fps) to provide a smooth and accurate representation of the environment. A higher frame rate means that the changes between consecutive frames are smaller, which can improve the accuracy of feature matching in the RTAB-Map algorithm. However, it's important to note that a higher frame rate also requires more computational resources and can potentially slow down the algorithm if the system's processing power is limited.

These adjustments to the camera settings, combined with the tuning of the RTAB-Map parameters, aim to provide a balance between speed and accuracy that is suitable for the specific requirements of the application.

Chapter 5

Data Fusion and Filtering Techniques

5.1 Need for Data Fusion

The primary emphasis is that despite the modifications made to the RTAB-Map parameters in Chapter 4, the topic: `/rtabmap/odom` - the output of RTAB-Map that publishes the position and orientation in quaternion form as an odometry type - only broadcasts at approximately 3Hz. For the drone's movements to be smooth and continuous, it is desirable to have an estimate of the position and orientation at a frequency close to 30Hz.

Furthermore, the script delineated in this section actualizes the dead reckoning algorithm, as expounded in Section 2.2.1.

It's of paramount importance to underscore that the data from the Inertial Measurement Unit (IMU) is procured at a swift frequency of 50Hz, while the Lidar data is disseminated at a rate of 20Hz. These relatively high frequencies, juxtaposed with the output rate from RTAB-Map, have been empirically proven to establish an ideal setting for the efficacious implementation of a dead reckoning algorithm.

This high-frequency data transmission ensures a more fluid and continuous estimation of the drone's position and orientation, thereby augmenting the overall efficacy of the navigation system.

In addition, given the proven accuracy of the RTAB-Map algorithm in estimating the drone's position, the data from the IMU is integrated and utilized only in the intervals between the RTAB-Map messages. These RTAB-

Map messages continue to serve as the primary reference for the drone's position estimation.

5.2 Python Script for Data Fusion

For this reason, a custom script was adopted. Here, we present a simplified pseudocode that outlines the most important functions and methods used in the script:

```
Import necessary libraries

Define a global variable for the last known position

Define a function to convert quaternion to euler

Define a function to convert euler to quaternion

Define a class DroneState with the following methods:

    - Initialization: Initialize state variables, deques for
      storing IMU and LIDAR data, and a publisher for the drone stat

    - publish_state: Publish the current state of the drone

    - update: Update the drone state based on the type of
      data received (odometry, IMU, or LIDAR)

    - record_state: Record the current state of the drone

    - update_and_save_path: Update the drone state
      and save the current path

    - moving_average: Compute the moving average of a given vector

    - print_and_plot_state: Print the current state and plot the
      drone's trajectory

If this script is the main module:
```

```
- Initialize a ROS node

- Create an instance of the DroneState class

- Define callback functions for odometry,
  IMU, and LIDAR data

- Subscribe to the appropriate ROS topics for odometry,
  IMU, and LIDAR data

- Set a ROS timer to record the drone state at a regular interval

- Start the ROS event loop

- If interrupted, shut down the ROS node and print and plot
  the final state of the drone
```

This pseudocode provides a high-level overview of the script's structure and main operations. For a detailed understanding and to see all the functions and methods, please refer to the full Python script provided in Appendix 1.

This script is designed to fuse data from multiple sources, namely odometry, IMU, and LIDAR, to estimate the drone's state, which includes its position and orientation. The state is updated at a frequency of 30Hz, which is ten times faster than the publishing rate of the RTAB-Map odometry topic. This higher update rate results in smoother and more continuous drone movements.

The script first initializes a `DroneState` object, which maintains the drone's current state and a history of its path. The state is represented as a seven-element array, with the first three elements representing the drone's position in the x, y, and z directions, and the last four elements representing the drone's orientation in quaternion form.

The script then defines callback functions for the odometry, IMU, and LIDAR topics. These functions call the `update` method of the `DroneState` object, passing in the message received from the topic and a string indicating the type of the data.

The `update` method updates the drone's state based on the received data.

If the data is from the odometry topic, the method updates the drone's position in the x and y directions. If the data is from the IMU topic, the method updates the drone's orientation and, if accelerometer readings are available, the drone's position in the x and y directions. If the data is from the LIDAR topic, the method updates the drone's position in the z direction.

The script incorporates a bias removal mechanism. When the script initially receives data from the IMU topic, it stores the first 10 readings in a vector. The mean of these readings is then computed and considered as the bias value. This bias value is subsequently subtracted from all future IMU readings. This process of bias removal aids in correcting any constant offset in the IMU readings, thereby enhancing the accuracy of the state estimation.

In addition to the bias removal, the script also utilizes a moving average filter to smooth the drone's trajectory. The filter operates by averaging the most recent readings from the accelerometer and LIDAR. This process of averaging aids in mitigating the impact of noise in the readings, resulting in a smoother trajectory.

Finally, the script includes a mechanism for recording the drone's path. Every 1/30th of a second, the script adds the drone's current state to the path history. This recorded path can then be plotted to visualize the drone's trajectory.

In conclusion, this script provides a robust and efficient method for fusing data from multiple sources to estimate the drone's state. By employing techniques such as bias removal and moving average filtering, the script ensures that the estimated state is both accurate and smooth, enabling precise and controlled drone movements.

5.2.1 Use of Lidar for Height Measurement

Several experiments were conducted to measure the drone's vertical position (pos_z) using different methodologies. These included:

1. Double integration of the linear acceleration in the z-direction from the IMU.

2. Utilizing only Lidar data.

3. A fusion of both IMU and Lidar data.

The outcomes of these experiments revealed some noteworthy insights. The height measurements derived solely from the accelerometer data (the first method) were found to be the most susceptible to noise and drift. This method, while straightforward, did not yield the most reliable or stable results.

Interestingly, there was no significant discrepancy between the measurements obtained using only Lidar data and those obtained from the fusion of both IMU and Lidar data. Given that the Lidar provides data at a sufficiently high rate and is highly precise for ranges below 10 meters, it is particularly suitable for indoor applications.

Therefore, considering these factors,it was decided to rely solely on Lidar data for estimating the drone's height. This decision was based on the Lidar's precision, reliability, and suitability for the specific application of indoor flight.

5.3 Implementation of Robot Localization

The implementation of the Robot Localization ROS package, while seemingly straightforward at first glance, presented a number of challenges that required additional work and innovative solutions.

One of the primary challenges was the need to publish the lidar data in a format that could be accepted by the Robot Localization package. The standard format for lidar data, as published on the topic `mavros/distance_sensor/rangefinder_pub`, is the Range message type. However, the Robot Localization package requires the lidar data to be published as a PoseWithCovarianceStamped message type. To address this issue, an additional Python script was developed to convert the lidar data from the Range format to the PoseWithCovarianceStamped format.

Another challenge was the inability to set initial biases in the Robot Localization package. In the dead reckoning script, biases were removed from the IMU data to improve the accuracy of the position and orientation estimates. However, the Robot Localization package does not provide a straightforward way to set these initial biases. To overcome this limitation, the same approach used in the dead reckoning script was applied, effectively removing the biases from the IMU data before it was input to the Robot Localization package.

The final configuration of the Robot Localization package included three main inputs: the RTAB-Map data for pos_x, pos_y, roll, pitch, and yaw from the topic `/rtabmap/odom`, the IMU data with biases removed for roll, pitch, yaw, angular velocity roll, pitch, yaw, and linear acceleration x, y, z published on the topic `/imu_localization`, and the lidar data for height (pos_z) in the PoseWithCovarianceStamped format from the topic `/lidar_localization`.

This configuration allowed for the successful implementation of the Robot Localization package and the effective fusion of data from multiple sensors, providing a comprehensive and accurate state estimation for the drone.

5.4 Tuning of Parameters of the EKF

The tuning of parameters in the Extended Kalman Filter (EKF) is a crucial step in achieving accurate state estimation. Two of the key parameters that require careful tuning are the process noise covariance matrix and the initial estimate covariance matrix.

The process noise covariance matrix is a representation of the noise that is added to the total error after each prediction step. This matrix can be difficult to tune and may vary for each application. The values in this matrix should ideally be smaller when the omnidirectional motion model matches the system well. However, if a given variable is slow to converge, one approach is to increase the diagonal value for that variable in the process noise covariance matrix. This will cause the filter's predicted error to be larger, which in turn will cause the filter to trust the incoming measurement more during the correction step.

The initial estimate covariance matrix, on the other hand, represents the initial value for the state estimate error covariance matrix. Setting a diagonal value (variance) to a large value in this matrix will result in rapid convergence for the initial measurements of the variable in question. However, care should be taken not to use large values for variables that will not be measured directly.

In the context of our implementation, these matrices were carefully tuned to ensure accurate state estimation. The tuning process involved adjusting the values in these matrices based on the performance of the EKF in estimating the state of the drone. By carefully tuning these parameters, we were able to achieve a balance between rapid convergence of the state estimates and the accuracy of these estimates.

During the testing phase, it was observed that the estimation of //the z-position was not satisfactory and there was noticeable noise in the x and y positions after small movements, causing them to oscillate. To address these issues, the values in the process noise covariance matrix and the initial estimate covariance matrix were adjusted.

In the process noise covariance matrix, the value corresponding to the z-position was increased. This caused the filter's predicted error to be larger in the z-direction, which in turn caused the filter to trust the incoming measurement more during the correction step. This adjustment helped improve the convergence of the filter in the z-direction.

In the initial estimate covariance matrix, a larger value was set for the z-position variance. This resulted in rapid convergence for the initial measurements of the z-position.

Despite the initial measurements for pos_x and pos_y being accurate, it was noticed that after small rapid movements, the noise in these positions was high, leading to oscillations. To address this, the corresponding values in the process noise covariance matrix were increased. This adjustment allowed the filter to trust the incoming measurements more during the correction step, thereby reducing the noise and oscillations in the pos_x and pos_y estimates.

These adjustments helped improve the performance of the EKF in estimating the state of the drone, leading to more accurate and reliable results.

Chapter 6

Results and Discussions

6.1 Experimental Setup

The experimental setup was meticulously designed to test the performance of the dead reckoning script and the Extended Kalman Filter (EKF) from the Robot Localization ROS package in an indoor environment. The drone was placed in a controlled indoor setting, with a clearly defined path for it to follow. The initial position of the drone was set at the origin of the coordinate system, (0,0,0), with the drone's motors turned off. This was done to ensure that the drone was stationary at the start of each test and that any movement was manually controlled.

The main focus of these tests was to evaluate the precision of the position estimates provided by both algorithms. For safety reasons and to ensure the accuracy of the path followed, the drone's motors were kept off during the tests. This also helped to eliminate any potential noise in the sensor data that could be introduced by the vibrations or other factors associated with the drone's motors.

The path chosen for the drone to follow was a rectangular shape with dimensions of 9.5 meters in the x-direction and 6 meters in the y-direction. The drone was manually moved along this path, starting from the initial position and moving ahead to a position of (4 meters on the x-axis, 0 on the y-axis, and 1.4 meters in height). This path was chosen to provide a variety of movements, including straight lines and turns, to thoroughly test the performance of the algorithms. Additionally, the edges of the rectangle presented different scenarios and light conditions, providing a comprehensive test environment to evaluate how RTAB-Map position estimate reacted to varying conditions.

The position (4,0,1.4) was not chosen randomly. It was selected to test the drone with a variety of movements, including straight lines and turns. Most importantly, upon completing the mission at this position, the drone's frontal camera could only see a grid and a glass window on the wall. This setup provided valuable insights into the physical elements that affected the state estimation of RTAB-Map, which will be discussed in detail in the next section.

It's important to note that every time the script is launched, the drone starts at the position (0,0,0). In this coordinate system, x represents the forward direction (in the direction of the drone's nose), y represents the left direction, and z represents the upward direction.

In the next section, the testing methodology used to evaluate the performance of the dead reckoning script and the EKF will be discussed in more detail.

6.2 Testing Methodology

The testing methodology was designed to evaluate the performance of the dead reckoning script and the Extended Kalman Filter (EKF) from the Robot Localization ROS package under a variety of conditions. The tests were conducted following a specific set of procedures, and the performance of the algorithms was evaluated based on a set of predefined metrics.

6.2.1 Test Procedures

The tests were initiated by launching the necessary files for the RealSense camera, RTAB-Map, CubeOrange, and the dead reckoning script or the Robot Localization package. The following launch commands were used:

- RealSense: `roslaunch realsense2_camera rs_camera.launch align_depth:=true color_width:=640 color_height:=480 depth_width:=640 depth_height:=480 color_fps:=60 depth_fps:=60`
- RTAB-Map: `roslaunch my_rtabmap_launch rtabmap_d435i.launch`
- CubeOrange: `roslaunch cube_orange cube_orange_imu.launch`

- Dead Reckoning: `python dead_reckoning.py`
- Robot Localization: `python topics_for_ekf.py` and `roslaunch my_robot_localization_config localization.launch`

Once the necessary files were launched, the drone was manually moved along the predetermined path. The tests included lifting the drone up and descending at various rates, moving along straight lines at different speeds, and turning at the edges of the path at different velocities. The drone was also exposed to different lighting and environmental conditions to test the performance of the algorithms under varying scenarios.

6.2.2 Performance Metrics

The primary performance metric was the accuracy of the position estimates provided by the algorithms. This was measured in terms of how much the estimated path deviated from the designated path, with the deviation measured in centimeters. Additionally, the ability of the algorithms to handle outliers and provide consistent position estimates was also evaluated.

6.2.3 Data Analysis

The data for the tests was collected from the topics published by the dead reckoning script and the Robot Localization package. The data was recorded and plotted at the end of each test to visually assess the performance of the algorithms and to facilitate a detailed analysis of the results. In the next section, the performance of the dead reckoning script and the EKF will be discussed in more detail based on the results of these tests.

6.3 Performance of Dead Reckoning and EKF

The performance of both the dead reckoning script and the Extended Kalman Filter (EKF) from the Robot Localization ROS package was evaluated based on the accuracy of their position estimates and their ability to handle varying conditions. The results of the tests are discussed in detail in this section.

6.3.1 Drift from the Ideal Path

One of the key performance metrics was the drift from the ideal path in the x and y directions. Both the EKF and the dead reckoning script were evaluated based on this metric. The drift was measured in terms of the deviation of the estimated path from the designated path.

- **Dead Reckoning:** The performance of the Dead Reckoning algorithm, as illustrated in the subsequent images, exhibits a linear drift along the vertical stretches of the x-axis. For instance, from the initial position to the first corner (top right), the estimated position deviates by 27cm to the right of the ideal path. Similarly, at the third corner (bottom left), the estimated position shifts 22cm to the left.

  During the straight stretches, the position estimation remains remarkably stable, with minimal oscillations. The horizontal drift peaks at a deviation of approximately ±14cm. This consistency in the horizontal stretches demonstrates the robustness of the Dead Reckoning algorithm in estimating a steady trajectory, despite the inherent challenges of manual drone navigation.

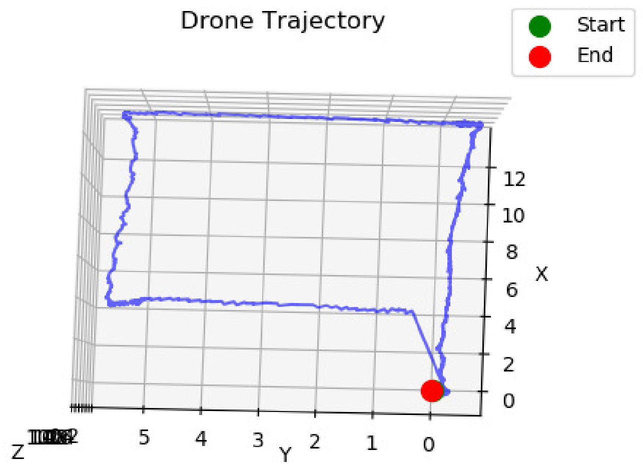

Figure 6.1: Top view of the test path (dead reckoning)

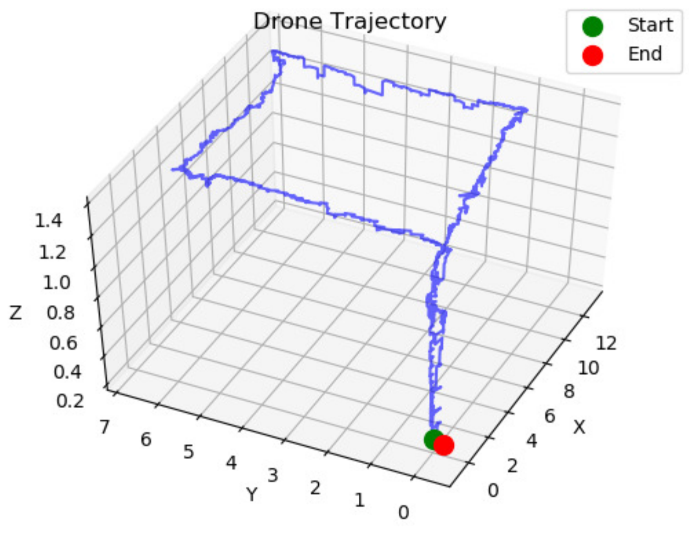

Figure 6.2: Plot of another view of the test path

Error in Returning to Initial Position with dead reckoning

It's crucial to highlight that this approach has consistently yielded impressive results across various tests, particularly in accurately estimating the target position. This is a key outcome that holds significant potential for future applications, such as precision landing, which is a component of the Fixit project. Across different tests, including the one depicted in the figure below, the maximum average deviation observed is approximately 10cm. This level of precision underscores the effectiveness of the implemented solution in maintaining close proximity to the desired path.

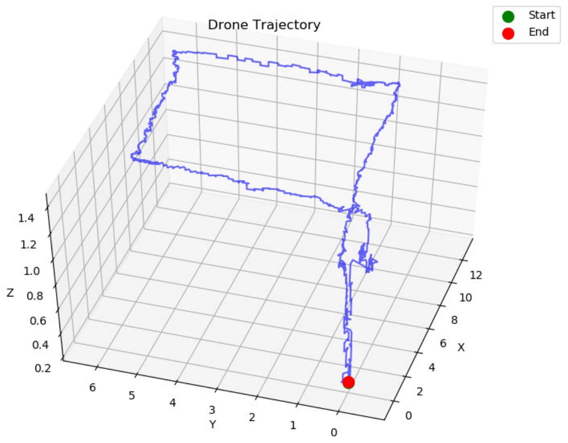

Figure 6.3: Plot of the path from an other test

- **EKF:** The performance of the Extended Kalman Filter (EKF) algorithm, as depicted in the subsequent images, demonstrates a slightly reduced linear drift along the vertical stretches of the x-axis compared to the Dead Reckoning algorithm. For instance, the horizontal deviation is only 20cm to the right of the ideal path at the top right corner and 16cm to the left at the bottom left corner.

  During the straight stretches, the position estimation remains remarkably stable, mirroring the performance of the Dead Reckoning algorithm. The primary differences are observed in the slightly reduced noise on the straight lines, which is around 10cm, except for the line going from the top left to the bottom left corner where the noise increases, reaching a deviation of 16cm.

However, it's important to note that in multiple tests, the EKF algorithm consistently presented outliers. For example, in the top left corner, the estimated trajectory deviates almost 40cm from the designated path. This occurrence of outliers is a significant factor to consider when evaluating the overall performance of the EKF algorithm.

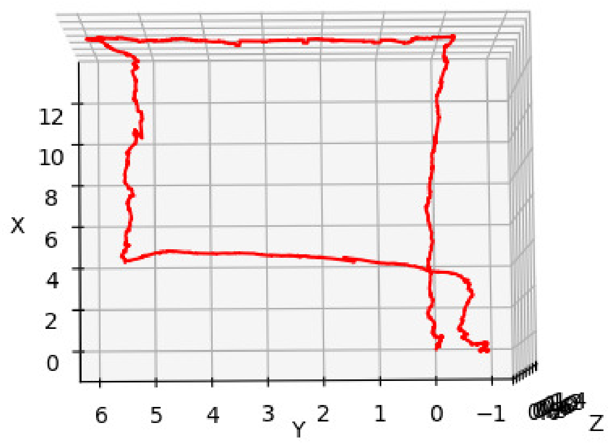

Figure 6.4: Top view of the test path (EKF) from the Robot Localization package

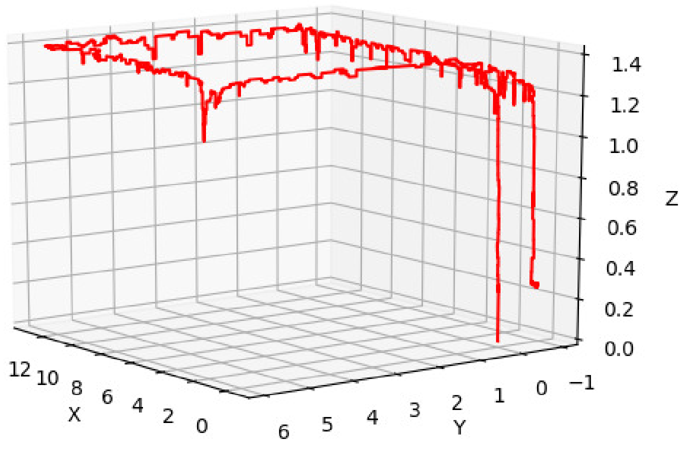

Figure 6.5: Plot of another view of the test path

Error in Returning to Initial Position with robot localization EKF

The implementation of the Extended Kalman Filter (EKF) algorithm with Robot Localization, despite its success in reducing noise during straight-line movements, reveals a significant shortcoming in the accuracy of position estimation when returning to the starting position. As illustrated in the subsequent figure, the deviation along the vertical axis (x-axis) is negligible. However, along the horizontal axis (y-axis), the estimated position deviates to the right by almost one meter.

This substantial deviation could have consequential implications in the context of the FIXIT project or other applications that demand precise landing. Therefore, while the EKF algorithm demonstrates strengths in certain aspects of position estimation, this significant discrepancy in the final position accuracy underscores the need for further refinement and optimization of the algorithm.

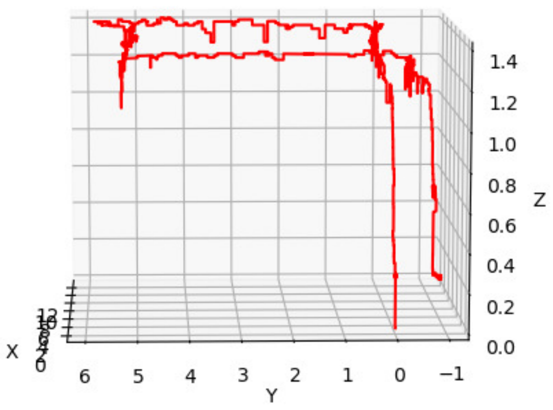

Figure 6.6: Plot of the path from an other prospective

6.3.2 Noise in Z Position Estimation

The estimation of the drone's position along the z-axis, or its height, is a critical aspect of the navigation algorithms. However, this estimation can be subject to noise, which can impact the accuracy of the drone's perceived position. In this subsection, we will examine the noise in the z position estimation for both the EKF and Dead Reckoning algorithms, and discuss the impact of tuning on the EKF's performance.

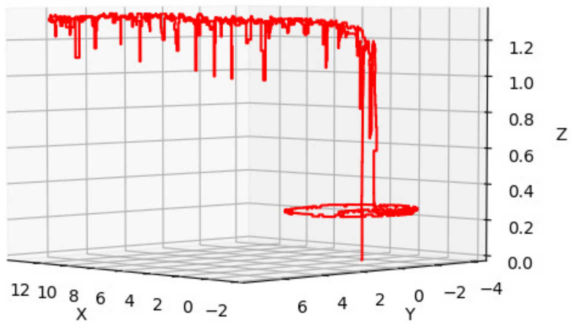

Figure 6.7: EKF Z Position Estimation Before Tuning

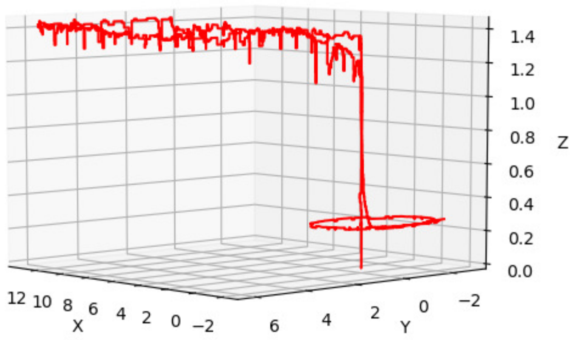

Figure 6.8: EKF Z Position Estimation After Tuning

As shown in Figure 6.7 and 6.8, the EKF's initial estimation of the z position exhibited significant noise. This noise could potentially lead to inaccuracies in the drone's perceived position and trajectory. However, after tuning the EKF parameters, as shown in Figure 6.8, the noise in the z position estimation was significantly reduced. This improvement demonstrates the importance of tuning in optimizing the performance of the EKF.

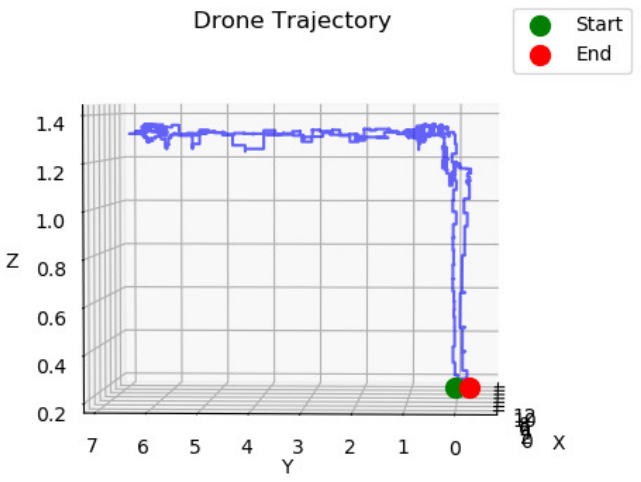

Figure 6.9: Dead Reckoning Z Position Estimation

In comparison, the Dead Reckoning algorithm, as shown in Figure 6.9, exhibited significantly less noise in the z position estimation from the outset. This suggests that the Dead Reckoning algorithm may provide a more accurate estimation of the drone's height, potentially making it a more suitable choice for applications that require precise vertical positioning.

6.4 Challenges and Exception Handling

During the testing phase, we encountered several challenges that tested the robustness and adaptability of both the Dead Reckoning and Extended Kalman Filter (EKF) algorithms. These challenges ranged from high-speed turns to signal loss due to passing through featureless zones. In this section, we discuss how these issues were handled and the measures implemented to ensure the safety of the drone and its surroundings.

6.4.1 Handling High-Speed Turns

One of the significant challenges was dealing with high-speed turns. Moving the drone in a straight line or up and down at different speeds did not significantly impact the quality of the pose estimation. However, executing sharp turns, such as 180 or 90 degrees, at high speed often led to the algorithm losing its references. This loss of reference points hindered the closure loop detection, which is fundamental for the algorithm's performance.

Despite these challenges, we found that even 90-degree turns could be executed at a sufficient rate under optimal conditions. However, high-speed turns often led to critical results, with the position estimation getting lost in 90% of the cases.

6.4.2 Challenges in Featureless Environments

Featureless environments posed a significant challenge for the algorithms. Instances where there were no objects within a 4-meter range(The depth range of the intelrealsense d435i) in the camera view often led to the loss of position estimation. Similarly, environments with monotonous color or texture, such as a white wall or a large blue curtain, also posed difficulties.

Lighting conditions also played a significant role in the performance of the algorithms. Soft or low lighting conditions were only manageable when the environment was rich in features.

In conclusion, the performance of both the Dead Reckoning and EKF algorithms was influenced by a variety of factors, including the drone's speed, the richness of features in the environment, and the lighting conditions. Exception handling measures were implemented to ensure the safety of the drone and its surroundings, and to improve the robustness of the algorithms in challenging conditions.

6.4.3 Feature Detection and Closed-Loop Recognition

The performance of the algorithms was also influenced by the environment's feature richness. The RTAB-Map algorithm, for instance, relies on detecting features in the environment to estimate the drone's position. We provide an example of how RTAB-Map detects features in the environment in the following figure.

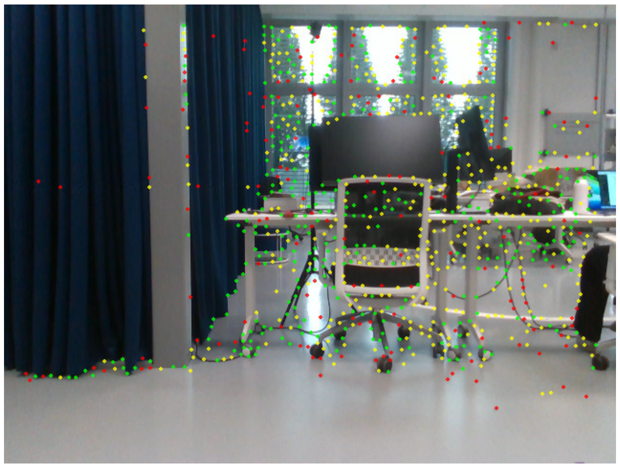

Figure 6.10: All detected Features

Similarly, the algorithm's ability to recognize loop closure is crucial for accurate position estimation. An example of which features RTAB-Map uses when recognizing loopclosure is provided in the subsequent figure.

As depicted in both images, RTAB-Map struggles to identify any discernible features on the uniform surfaces of the blue curtain and the grey floor. Consequently, the loop closure detector fails to locate any previous references in these areas, which hampers its ability to gather data when the drone navigates around these featureless zones.

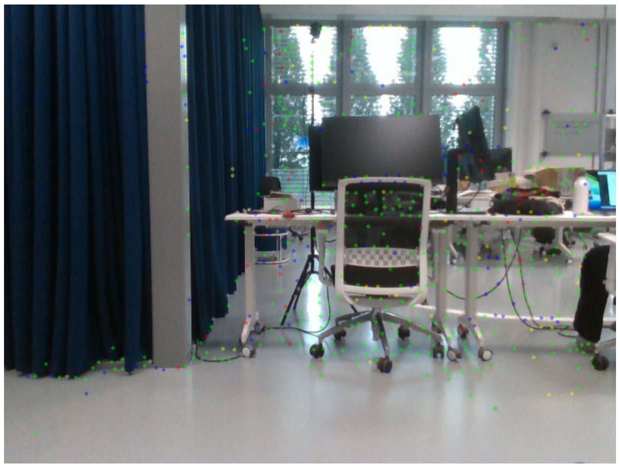

Figure 6.11: Loop Closure Detector

6.4.4 Challenges in Featureless Environments

Featureless environments posed a significant challenge for the algorithms. Instances where there were no objects within a 4-meter range (The depth range of the intelrealsense d435i) in the camera view often led to the loss of position estimation. Similarly, environments with monotonous color or texture, such as a white wall or a large blue curtain, also posed difficulties.

Lighting conditions also played a significant role in the performance of the algorithms. Soft or low lighting conditions were only manageable when the environment was rich in features. To illustrate the impact of lighting conditions on feature detection, we conducted tests under two different lighting conditions: light and dark. The following figures show the feature detection results in these two scenarios.

As shown in Figure 6.12, the feature detection algorithm performs well under light conditions, identifying a large number of features. However, the performance significantly decreases under dark conditions, as shown in the following figure.

As depicted in Figure 6.13, the algorithm struggles to identify features under poor lighting conditions. This reduction in the number of detected features directly impacts the quality of odometry, as fewer features mean fewer potential loop closure detectors. This can lead to inaccurate position estimation and potential navigation issues.

RTAB-Map evaluates the quality of odometry using a metric called "odom

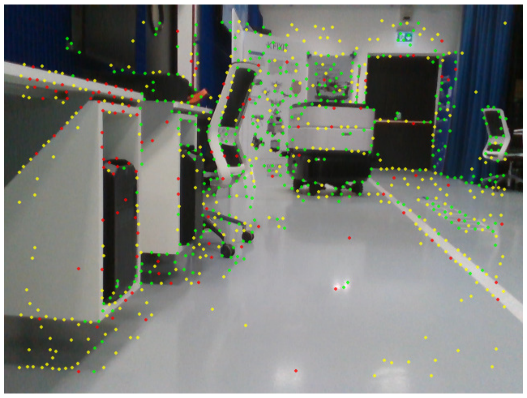

Figure 6.12: Feature Detection in Light Conditions

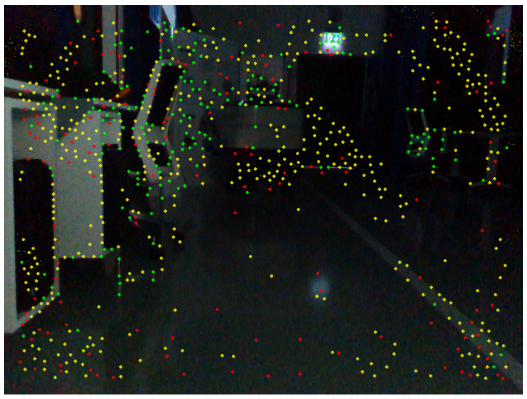

Figure 6.13: Feature Detection in Dark Conditions

quality". A higher odom quality value indicates a higher number of detected features and, therefore, a more reliable odometry. For instance, an odom quality of 500 means that the algorithm has detected 500 distinct features in the environment, which it can use for position estimation and loop closure detection. However, under poor lighting conditions, the odom quality can significantly decrease, leading to less reliable odometry.

6.4.5 Response of Algorithms to Data Loss from RTAB-Map

The reliability and consistency of the data stream from RTAB-Map is of paramount importance for both the EKF and Dead Reckoning algorithms. A steady and continuous reference from RTAB-Map significantly mitigates drift, enhancing the accuracy of position and orientation estimation.

However, there are instances when RTAB-Map may fail to provide data, and the response of both algorithms to such a situation is crucial. As depicted in the accompanying image, the EKF, when deprived of data from RTAB-Map, starts to estimate the position in a spiraling pattern, with the diameter of the spiral reaching up to 4 meters at its peak. This erratic behavior underscores the importance of a reliable data source for the EKF algorithm.

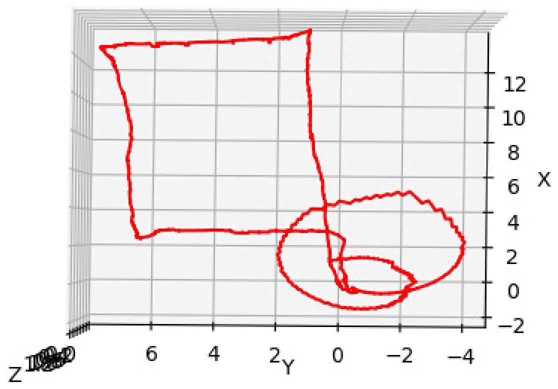

Figure 6.14: Spiral Trajectory of EKF Position Estimation during RTAB-Map Data Loss

On the other hand, the Dead Reckoning algorithm, when faced with a similar situation, maintains a relatively accurate reference for a longer duration, provided the drone remains stationary. However, in both cases, it is essential to detect such data loss promptly and initiate appropriate measures to safeguard the drone and its surroundings.

To handle such scenarios, I have implemented a script that continuously monitors data from the RTAB-Map topic. The script is designed to trigger an alert when it detects a failure in the data stream for a continuous duration of 3 seconds. This duration was chosen based on multiple tests that showed that the algorithm could recover and resume position and orientation estimation after a brief interruption of 1 or 2 seconds. Upon detecting a prolonged data failure, the script instructs the drone to land, thereby preventing any potential mishaps due to inaccurate position estimation.

Chapter 7

Future Work and Conclusion

7.1 Testing More Computationally Expensive Forms of Data Fusion Algorithms

The exploration of more computationally intensive data fusion algorithms presents a compelling avenue for future research. While these algorithms may increase computational costs and potentially slow down the system, they also hold the promise of delivering more accurate and smoother results and trajectories. This trade-off between computational efficiency and accuracy is a central challenge in the field of data fusion and is a key consideration in the development of advanced navigation systems.

The integration of global positional information with visual and inertial measurements in Visual Odometry (VO) algorithms has been underscored in recent studies [7, 8, 9]. These studies present a compelling case for the use of more computationally intensive, tightly-coupled data fusion algorithms that can leverage the correlations amongst all the measurements, leading to more accurate and globally consistent estimates.

One of the most commonly used data fusion algorithms in navigation systems is the Kalman filter. This algorithm uses a set of mathematical equations to predict the state of a system at a future time, based on the current state and the system dynamics. It then updates this prediction with new measurements, weighting the prediction and the measurements based on their estimated uncertainties. The Kalman filter is computationally efficient and provides optimal estimates under certain conditions, but it assumes that the system and measurement noises are Gaussian and that the system dynamics are linear, which may not always be the case.

Particle filters, on the other hand, are a more flexible and powerful alternative to Kalman filters. They can handle nonlinear system dynamics and non-Gaussian noises, and they can represent multi-modal probability distributions. However, particle filters are more computationally intensive than Kalman filters, and they require careful tuning of the number of particles and the resampling strategy.

Bayesian networks are another type of data fusion algorithm that can handle complex, nonlinear relationships between variables. They represent the joint probability distribution over a set of variables, and they can incorporate prior knowledge about the relationships between these variables. Bayesian networks can provide a more accurate and complete representation of the system than Kalman filters or particle filters, but they are also more computationally intensive and require more data to train.

The choice of data fusion algorithm depends on the specific requirements of the system and the characteristics of the data. For a drone navigation system, the accuracy and smoothness of the estimated trajectory are of paramount importance, but the computational cost and speed of the system are also critical considerations. More computationally intensive algorithms may provide more accurate and smooth estimates, but they may also slow down the system and increase the computational cost.

In the future, it would be interesting to explore the use of more computationally intensive data fusion algorithms in the context of drone navigation. This could involve implementing and testing different algorithms, comparing their performance in terms of accuracy, smoothness, computational cost, and speed, and optimizing their parameters for the specific characteristics of the drone and its sensors. It would also be interesting to investigate the use of machine learning techniques for data fusion, such as deep learning algorithms that can learn complex, nonlinear relationships from data.

In conclusion, the work presented in this thesis represents an important step towards the development of more accurate and efficient navigation systems for drones. However, there is still much work to be done. The exploration of more computationally intensive data fusion algorithms, as well as the integration of other types of sensors and the use of machine learning techniques, will be crucial for further improving the accuracy and reliability of drone navigation systems. [7, 8, 9].

7.2 Implementing YOLO Algorithm for Object Recognition

The advent of Industry 4.0 has ushered in a new era of automation and data exchange in manufacturing technologies. This revolution is characterized by the integration of cyber-physical systems, the Internet of Things, and cloud computing. Unmanned Aerial Vehicles (UAVs), or drones, play a pivotal role in this transformation, particularly in the realm of data collection.

Drones equipped with advanced object recognition capabilities can significantly augment the efficiency and effectiveness of industrial operations. They can be utilized for tasks such as inventory management, where they can swiftly and accurately count items in a warehouse, or for surveillance purposes, where they can monitor and report unusual activities, thereby enhancing security.

The integration of the You Only Look Once (YOLO) algorithm into drone systems can be a game-changer in this context. YOLO, an object detection system targeted for real-time processing, can identify objects in a scene and classify them into predefined categories. The latest iterations of this algorithm, such as YOLO v3 or YOLOv8 mini, have demonstrated impressive accuracy and speed, making them well-suited for implementation in drone systems.

The integration of advanced object recognition algorithms like YOLO into drone systems represents a promising avenue for future research and development. As the capabilities of these algorithms continue to improve, so too will the potential applications of drones in Industry 4.0 and beyond.

7.3 Use of Drone in High Magnetic Field Environments

Intense magnetic fields can disrupt the functionality of a drone's Inertial Measurement Unit (IMU), a critical component for ensuring stability and navigation. The IMU, which often incorporates a magnetometer, can be substantially influenced by the presence of potent magnetic fields, resulting in erroneous readings and potential flight instability.

Despite these hurdles, the potential advantages of deploying drones in such environments are considerable. Drones could be employed for a multitude of tasks, such as inspecting machinery in power plants, monitoring conditions in scientific research facilities, or even executing tasks in environments that pose hazards to humans.

To effectively navigate these high magnetic field environments, drones would likely necessitate specialized hardware and software. For example, drones could be equipped with shielded IMUs or alternative navigation systems that are less prone to magnetic interference. In addition, sophisticated algorithms could be devised to counteract the effects of magnetic fields on the drone's sensors, enhancing stability and navigation.

Moreover, machine learning techniques could be utilized to anticipate and mitigate the effects of magnetic fields on the drone's operation. By training a model on data gathered from flights in high magnetic field environments, it might be feasible to anticipate the impacts of these fields and adjust the drone's operation accordingly.

In addition to visual odometry, other methods could be explored for navigation in extreme conditions. Techniques such as LIDAR-based navigation, radar, or even acoustic sensors could provide alternative means of navigation when visual methods are not feasible. These alternative methods and techniques could be tested to see if they can offer a robust solution for navigation in high magnetic field environments, further expanding the potential applications of drone technology.

7.4 Conclusions

This thesis has explored the development and implementation of a localization strategy for Unmanned Aerial Vehicles (UAVs) operating in indoor environments where GPS signals may not be reliable. The focus has been on the use of visual odometry, leveraging a depth camera and other onboard sensors to enhance localization accuracy under various conditions.

The cornerstone of this project has been the RTAB-Map ROS algorithm, which has been used for state estimation. The parameters of this algorithm have been carefully tuned to optimize the speed of state estimation while maintaining an acceptable level of accuracy. To ensure smooth drone movement, a Python script was developed to fuse data from RTAB-Map, Lidar, and IMU using a filtering technique.

The drone, equipped with a Jetson Nano companion computer, a CubeOrange with Ardupilot, and a Lidar, has been tested in an indoor GPS-denied environment. The results have validated the effectiveness of the proposed solutions, demonstrating that the drone can operate autonomously and navigate smoothly in such environments.

This work has potential applications in security and inventory management in factories, among others, contributing to the ongoing efforts to enhance the adaptability and flexibility of UAVs in the context of Industry 4.0.

Future work includes testing more computationally expensive forms of data fusion algorithms and implementing a YOLO algorithm for object recognition. The ultimate goal is to assist human operators in specific missions by collecting and processing data, thereby enhancing operational efficiency and safety in challenging indoor environments.

In conclusion, this thesis has demonstrated that with careful calibration of algorithm parameters and effective data fusion techniques, it is possible to enable autonomous indoor flight for UAVs using visual odometry techniques, even in the absence of GPS. This contributes to the broader field of autonomous systems and Industry 4.0, opening up new possibilities for the use of UAVs in various industrial applications.

Bibliography

[1] Li, Z., & Zhou, Y. (2022). Feature Extraction and Matching Algorithm Based on SURF and RANSAC in Monocular Visual Odometry. In 2022 7th International Conference on Control, Automation and Robotics (ICCAR) (pp. 1-5). https://doi.org/10.1109/ICCAR54193.2022.9625170Link

[2] Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In Communications of the ACM (Vol. 24, No. 6, pp. 381-395). https://doi.org/10.1145/358669.358692Link

[3] Liu, H., Chen, X., Zhang, Y., & Su, X. (2019). Visual Odometry for UAVs: A Review. IEEE Access, 7, 102390-102410. https://ieeexplore.ieee.org/document/8754671Link

[4] Angladon, V., Gasparini, S., & Charvillat, V. (2018). Video Dataset of Driving Scenes with Synchronized Visible and Thermal Infrared Videos. In Proceedings of the 9th ACM Multimedia Systems Conference (pp. 423-428). https://dl.acm.org/doi/10.1145/3204949.3208124Link

[5] Li, R., Wang, S., Long, Z., & Gu, D. (2019). Undirect: A Novel Method for Stereo Visual Odometry Estimation by Using Direct Method to Solve the Scale Problem. arXiv preprint arXiv:1903.02046. https://arxiv.org/abs/1903.02046Link

[6] Wang, R., Schworer, M., & Cremers, D. (2020). Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras. International Journal of Computer Vision, 128, 1014-1027. https://link.springer.com/article/10.1007/s11263-019-01242-2Link

[7] Zhang, Z., & Scaramuzza, D. (2018). A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. In 2018 IEEE/RSJ In-

ternational Conference on Intelligent Robots and Systems (IROS) (pp. 7244-7251). IEEE. https://ieeexplore.ieee.org/document/8593949Link

[8] Arroyo, R., Gonzalez, D., & Ollero, A. (2014). Fusion of Visual and Inertial Data for Robust RGB-D SLAM. In 2014 IEEE International Conference on Robotics and Automation (ICRA) (pp. 5556-5563). IEEE. https://ieeexplore.ieee.org/document/6907746Link

[9] Zhang, Z., Scaramuzza, D., & Davison, A. (2020). Tightly-Coupled Fusion of Global Pose and Inertial Measurements for Robust Monocular Visual-Inertial Estimation. IEEE Transactions on Robotics, 36(4), 1003-1019. https://ieeexplore.ieee.org/document/9099040Link

[10] Anzalone, S. M., & Dunlap, D. D. (2016). A comparison of monocular and stereo visual FastSLAM implementations. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)* (pp. 1-6). IEEE. doi:10.1109/ICARCV.2016.7838702

[11] J. Szrek, P. Trybała, M. Góralczyk, A. Michalak, B. Ziętek, and R. Zimroz, "Accuracy Evaluation of Selected Mobile Inspection Robot Localization Techniques in a GNSS-Denied Environment," Sensors, vol. 21, no. 1, p. 141, 2021. https://doi.org/10.3390/s21010141

[12] Z. Menghan, L. Zitian and S. Yuncheng, "Optimization and Comparative Analysis of YOLOV3 Target Detection Method Based on Lightweight Network Structure," 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), 2020, pp.

[13] L. Fusini, T. A. Johansen and T. I. Fossen, "Dead reckoning of a fixed-wing UAV with inertial navigation aided by optical flow," 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 1250-1259, doi: 10.1109/ICUAS.2017.7991433.

[14] D. B. Jeong and N. Y. Ko, "Dead Reckoning of a Mobile Robot in 2-Dimensional Special Euclidean Group," 2022 22nd International Conference on Control, Automation and Systems (ICCAS), Jeju, Korea, Republic of, 2022, pp. 1069-1071, doi: 10.23919/ICCAS55662.2022.10003795.

[15] S. Chandrachary, Introduction to 3D SLAM with RTAB-Map, 2021. [Online]. Available: https://shivachandrachary.medium.com/introduction-to-3d-slam-with-rtab-map-8df39da2d293. [Accessed: July 2, 2023].

.1 Appendix

Python Code

```
#!/usr/bin/env python3



import rospy
from sensor_msgs.msg import Imu, Range
from nav_msgs.msg import Odometry
import math
from collections import deque
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from geometry_msgs.msg import
PoseWithCovarianceStamped, PoseStamped, Quaternion, Pose

last_known_position = None

# Function to convert quaternion to euler
def euler_from_quaternion(x, y, z, w):
    t0 = +2.0 * (w * x + y * z)
    t1 = +1.0 - 2.0 * (x * x + y * y)
    roll_x = math.atan2(t0, t1)

    t2 = +2.0 * (w * y - z * x)
    t2 = +1.0 if t2 > +1.0 else t2
    t2 = -1.0 if t2 < -1.0 else t2
    pitch_y = math.asin(t2)

    t3 = +2.0 * (w * z + x * y)
    t4 = +1.0 - 2.0 * (y * y + z * z)
    yaw_z = math.atan2(t3, t4)

    return roll_x, pitch_y, yaw_z  # in radians
```

```
def get_quaternion_from_euler(roll, pitch, yaw):
    qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2)
    - np.cos(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)

    qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)
    + np.sin(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)

    qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)
    - np.sin(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)

    qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2)
    + np.sin(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)

    return Quaternion(x=qx, y=qy, z=qz, w=qw)

class DroneState:
    def _init_(self):
        self.state = np.array([0, 0, 0, 0, 0, 0, 1], dtype=float)
        self.imu_acc_deque_x = deque(maxlen=5)
        self.imu_acc_deque_y = deque(maxlen=5)
        self.last_time = None
        self.dt = 0
        self.bias_x = None
        self.bias_y = None
        self.bias_z = None
        self.bias_roll = None
        self.bias_pitch = None
        self.bias_yaw = None
        self.bias_saved = False
        self.path = {"x": [], "y": [], "z": []}
        self.yaw_bias = None
        self.pub = rospy.Publisher
        ('drone_state', PoseStamped, queue_size=10)
        self.lidar_range_deque = deque(maxlen=5)


    def publish_state(self):
        msg = PoseStamped()
        msg.header.stamp = rospy.Time.now()
        msg.header.frame_id = "drone"

        msg.pose.position.x = self.state[0]
        msg.pose.position.y = self.state[1]
        msg.pose.position.z = self.state[2]

        msg.pose.orientation.x = self.state[3]
        msg.pose.orientation.y = self.state[4]
        msg.pose.orientation.z = self.state[5]
        msg.pose.orientation.w = self.state[6]

        self.pub.publish(msg)

    def update(self, msg, data_type):
        if self.last_time is None:
            self.last_time = msg.header.stamp

        self.dt = (msg.header.stamp - self.last_time).to_sec()

        if data_type == "odom":

            self.state[0] = msg.pose.pose.position.x
            self.state[1] = msg.pose.pose.position.y


        elif data_type == "imu":
            roll, pitch, yaw = euler_from_quaternion
            (msg.orientation.x,
            msg.orientation.y,
            msg.orientation.z,
            msg.orientation.w)

            # Save bias values if not saved yet
            if not self.bias_saved:
                self.bias_x = msg.linear_acceleration.x
                self.bias_y = msg.linear_acceleration.y
                self.bias_z = msg.linear_acceleration.z
            self.bias_roll = roll
            self.bias_pitch = pitch
            self.bias_yaw = yaw
            self.bias_saved = True



        # Subtract bias values from readings
        acc_x = msg.linear_acceleration.x - self.bias_x
        acc_y = msg.linear_acceleration.y - self.bias_y
        acc_z = msg.linear_acceleration.z - self.bias_z
        roll -= self.bias_roll
        pitch-=self.bias_pitch
        yaw-=self.bias_yaw

        # Add the adjusted acceleration data to the deques
        self.imu_acc_deque_x.append(acc_x)
        self.imu_acc_deque_y.append(acc_y)

        # Update roll, pitch, yaw regardless of
        whether accelerometer readings are available
        quaternion= get_quaternion_from_euler(roll, pitch, yaw)

        self.state[3] = quaternion.x
        self.state[4] = quaternion.y
        self.state[5] = quaternion.z
        self.state[6] = quaternion.w


        # Only update x and y positions if accelerometer reading
        are available
        if self.imu_acc_deque_x and self.imu_acc_deque_y:
            self.state[0] += (self.moving_average
            (self.imu_acc_deque_x) * self.dt**2)/2
            self.state[1] += (self.moving_average
            (self.imu_acc_deque_y) * self.dt**2)/2

    elif data_type == "lidar":
            # Add LIDAR reading to deque
            self.lidar_range_deque.append(msg.range)
            # Only update z position if LIDAR readings are available
            if self.lidar_range_deque:
                self.state[2] = self.moving_average
                (self.lidar_range_deque)

        self.last_time = msg.header.stamp

        self.path["x"].append(self.state[0])
        self.path["y"].append(self.state[1])
        self.path["z"].append(self.state[2])
        self.publish_state()

    def record_state(self, event):
        self.path["x"].append(self.state[0])
        self.path["y"].append(self.state[1])
        self.path["z"].append(self.state[2])

    def update_and_save_path(self, event):
        self.update("odom")
        self.update("imu")
        self.update("lidar")
        self.path["x"].append(self.state[0])
        self.path["y"].append(self.state[1])
        self.path["z"].append(self.state[2])

    def moving_average(self, vec, n=5):
        if len(vec) > n:
            vec.pop(0)
        return np.mean(vec)

    def print_and_plot_state(self):
        # Print state
        print(np.round(self.state, 5))

        # Plot trajectory
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.plot(self.path["x"], self.path["y"], self.path["z"])
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        plt.show()

if _name== "main_":
    rospy.init_node('state_estimator')

    drone = DroneState()

    def odom_callback(msg):
        drone.update(msg, "odom")

    def imu_callback(msg):
        drone.update(msg, "imu")

    def lidar_callback(msg):
        drone.update(msg, "lidar")

    rospy.Subscriber("/rtabmap/odom", Odometry, odom_callback)
    rospy.Subscriber("/mavros/imu/data", Imu, imu_callback)
    rospy.Subscriber("/mavros/distance_sensor/rangefinder_pub"
    , Range, lidar_callback)

    rospy.Timer(rospy.Duration(1.0/30.0), drone.record_state)

    try:
        rospy.spin()
    except KeyboardInterrupt:
        rospy.signal_shutdown('Interrupted')
    finally:
        drone.print_and_plot_state()
```