UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Drone positioning via feature detection on aerial images taken at flight time

Diploma Thesis

Galanis Achilleas-Alexandros-Vasileios

Supervisor: Spyros Lalis

Month 2024

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Drone positioning via feature detection on aerial images taken at flight time

Diploma Thesis

Galanis Achilleas-Alexandros-Vasileios

Supervisor: Spyros Lalis

Month 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Υπολογισμός της θέσης ενός drone μέσω εντοπισμού χαρακτηριστικών στοιχείων σε αεροφωτογραφίες που λαμβάνονται κατά την διάρκεια της πτήσης

Διπλωματική Εργασία

Γαλάνης Αχιλλέας-Αλέξανδρος-Βασίλειος

Επιβλέπων/πουσα: Σπύρος Λάλης

Μήνας 2024

Approved by the Examination Committee:

Supervisor **Spyros Lalis**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Christos D. Antonopoulos**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Nikolaos Bellas**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First of all I would like to extend my heartfelt thanks to my supervisor in this endeavor, Prof. Spyros Lalis for his expert guidance, invaluable advice, and the motivation he provided, which were essential for the completion of this thesis.

I would also like to thank Manos Koutsoubelias and Maria-Rafaella Gkeka for their dedication and time providing me with resourceful ideas and advice that were essential for the progression of my work.

Finally, now that my academic journey comes to an end, I want to express my gratitude to my friends, family, and loved ones for their unwavering support and steadfast faith in me.

No matter how hard or impossible it is, never lose sight of your goal.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Galanis Achilleas-Alexandros-Vasileios

Diploma Thesis

Drone positioning via feature detection on aerial images taken at flight time

Galanis Achilleas-Alexandros-Vasileios

Abstract

Over the past few years, the use of Unmanned Aerial Vehicles, generally referred to as drones, has soared in various fields like aerial surveillance, environmental monitoring, and precision agriculture. In these applications, GPS serves as the foundation for ensuring accurate location tracking. However, the reliability of GPS is not infallible, raising the critical question of how to determine a drone's position in the event of a GPS failure. Addressing this issue, this work introduces a robust solution that utilizes feature detection on downward images captured by the drone's camera to efficiently calculate the drone's position. The proposed algorithm detects and tags image features with their geographical coordinates, enabling accurate position computation. The integration of visual data guarantees continuous monitoring, thus improving the reliability of the drone function. The effectiveness of this method is validated through extensive simulations and experimental setups, demonstrating its robustness and accuracy in diverse operational scenarios.

Διπλωματική Εργασία

Υπολογισμός της θέσης ενός drone μέσω εντοπισμού χαρακτηριστικών στοιχείων σε αεροφωτογραφίες που λαμβάνονται κατά την διάρκεια της πτήσης

Γαλάνης Αχιλλέας-Αλέξανδρος-Βασίλειος

Περίληψη

Τα τελευταία χρόνια, η χρήση Μη Επανδρωμένων Αεροχημάτων, γενικά γνωστών ως drones, έχει εκτοξευθεί σε διάφορους τομείς όπως η εναέρια επιτήρηση, η παρακολούθηση του περιβάλλοντος και η ακριβής γεωργία. Σε αυτές τις εφαρμογές, το GPS χρησιμεύει ως η βάση για την εξασφάλιση ακριβούς εντοπισμού θέσης. Ωστόσο, η αξιοπιστία του GPS δεν είναι αλάθητη, θέτοντας το κρίσιμο ερώτημα πώς να προσδιοριστεί η θέση ενός drone σε περίπτωση αποτυχίας του GPS. Αντιμετωπίζοντας αυτό το ζήτημα, η παρούσα εργασία εισάγει μια αξιόπιστη λύση που χρησιμοποιεί την ανίχνευση χαρακτηριστικών σε εικόνες που λαμβάνονται από την κάμερα του drone για να υπολογίσει αποτελεσματικά τη θέση του. Ο προτεινόμενος αλγόριθμος ανιχνεύει και επισημαίνει χαρακτηριστικά εικόνας με τις γεωγραφικές τους συντεταγμένες, επιτρέποντας ακριβή υπολογισμό της θέσης. Η ενσωμάτωση οπτικών δεδομένων εγγυάται συνεχή παρακολούθηση, βελτιώνοντας έτσι την αξιοπιστία της λειτουργίας του drone. Η αποτελεσματικότητα αυτής της μεθόδου επικυρώνεται μέσω εκτενών προσομοιώσεων και πειραματικών διατάξεων, επιδεικνύοντας την ανθεκτικότητα και την ακρίβειά της σε διάφορα λειτουργικά σενάρια.

Table of contents

Acknowledgements **ix**

Abstract **xii**

Περίληψη **xiii**

Table of contents **xv**

List of figures **xvii**

List of tables **xix**

1 Introduction **1**

1.1 Motivation 1

1.2 Focus 2

1.3 Contributions 2

1.4 Outline 3

2 Related Work **5**

2.1 Visual Odometry 5

2.1.1 Types 5

2.1.2 Challenges 6

2.1.3 Approaches 7

2.2 Our Approach 8

2.3 Approaches for feature detection 9

2.4 Feature detection method chosen 10

3 Approach **13**

3.1 Overview 13

3.2 Implementation details 16

3.2.1 Camera parameters 16

3.2.2 Feature detection algorithm 16

3.2.3 Data structure and logic for coordinate assignment 18

3.2.4 Data structure and logic for drone positioning 21

4 Experimental Setup **25**

4.1 ArduPilot SITL 25

4.2 Gazebo simulation environment 26

4.3 Experimental parameters 29

4.3.1 Drone mission 29

4.3.2 Objects 30

5 Evaluation **33**

5.1 Accuracy of coordinate assignment algorithm 33

5.2 Accuracy of drone positioning algorithm 34

5.3 Results with respect to accuracy 34

5.3.1 Accuracy of coordinate assignment to detected features 34

5.3.2 Accuracy of drone positioning 37

5.4 Results with respect to performance 39

5.5 General observations 41

6 Conclusion **43**

Bibliography **45**

List of figures

2.1 Flowchart of visual odometry system algorithm. 8

3.1 Overview of the approach. 13
3.2 Image capture. 14
3.3 Feature detection. 15
3.4 Coordinate assignment. 15
3.5 Drone positioning. 16
3.6 Data structure analysis in coordinate assignment. 19
3.7 Data structure analysis in drone positioning. 22

4.1 Gazebo environment setup. 26
4.2 Gazebo-SITL control loop. 27
4.3 Drone camera created in Gazebo environment. 28
4.4 Camera view inside Gazebo environment. 28
4.5 Image captured from the Gazebo camera. 29

List of tables

5.1 1 cube mission coordinate assignment results for 10 meters. 35
5.2 1 cube mission coordinate assignment results for 15 meters. 35
5.3 Multiple cube mission coordinate assignment results with different color and 0.5x0.5x0.5 cube size. 36
5.4 Multiple cube mission coordinate assignment results with different color and 3x3x3 cube size. 36
5.5 Multiple cube mission coordinate assignment results with same color and 0.5x0.5x0.5 cube size. 37
5.6 Multiple cube mission coordinate assignment results with same color and 3x3x3 cube size. 37
5.7 1 cube mission drone positioning results for 10 meters. 38
5.8 1 cube mission drone positioning results for 15 meters. 38
5.9 Multiple cube mission drone positioning results with different color and cube size 0.5x0.5x0.5. 38
5.10 Multiple cube mission drone positioning results with different color and cube size 3x3x3x. 39
5.11 Multiple cube mission drone positioning results with same color and cube size 0.5x0.5x0.5. 39
5.12 Multiple cube mission drone positioning results with same color and cube size 3x3x3x. 39
5.13 System specifications 40
5.14 Comparison of average times (in seconds) for one-cube and multiple-cube missions on PC and Rpi. 41

Chapter 1

Introduction

1.1 Motivation

Drones, also known as Unmanned Aerial Vehicles (UAVs), are increasingly important for various applications such as aerial surveillance [1], environmental monitoring [2], and precision agriculture [3]. These apps require accurate and dependable navigation features, typically supplied by Global Positioning Systems (GPS) [4]. Nevertheless, conventional GPS systems, although widely used and reliable, can be affected by different disturbances. Physical barriers like buildings, thick foliage, or bad weather can cause disruptions that hinder or block GPS signals. Furthermore, GPS signals can be deliberately disturbed or deceived, leading to noticeable errors in navigation. If a drone loses its GPS signal, it often does not have another way to navigate, resulting in emergency landings in possibly unsafe or unplanned areas, like rough terrains, bodies of water, or even populated spaces. This not only interrupts the drone's task but also presents considerable dangers, such as potential damage or loss of the drone, or even the risk of injuring people in the landing area. This problem is important in situations where constant operation is needed or when returning safely to the initial point is crucial, like in search and rescue missions, disaster relief, and agricultural surveillance. Not having a dependable backup navigation system can lead to failure of the mission, loss of important data, and significant financial setbacks.

1.2 Focus

Therefore, there is a pressing need for an alternative navigation system that can provide reliable positional information, enabling drones to continue their mission or, at the very least, guide them safely back to their starting point. Addressing this need can significantly enhance the reliability and safety of drone operations, expanding their utility and effectiveness across various applications.

This thesis focuses on this problem and studies a mechanism that infers the position of the drones based on information extracted from aerial images taken during flight using the drone's onboard camera. More specifically, as long as the drone has a good GPS signal, a feature detection method is used to identify the key features of each image/frame taken as the drone flies above a certain region. Each detected feature is kept in a list and is assigned a GPS coordinate based on the known position of the drone (via GPS). Then, if the GPS signal is lost, the position of the drone can be inferred following the reverse approach, by calculating it based on the GPS coordinates of the features that are identified in each image/frame.

1.3 Contributions

The main contributions of our work are briefly as follows:

1. We explore the usage of feature detection on aerial images to infer the position of a drone, so that this information can be used as a backup navigation system in case the GPS signal is lost during the mission.

2. We describe a concrete implementation of this approach.

3. For our experiments, we setup an elaborate software-in-the-loop configuration, where the ArduPilot autopilot framework [5] is coupled with a flight dynamics model in the Gazebo simulation environment [6] which also provides the video stream of the virtual drone's onboard camera during flight, which is processed by an external program running the feature detection method and performing the respective coordinate assignment.

4. We evaluate the proposed approach through a series of experiments for different scenarios regarding the shape, size, color and number of objects in the mission terrain. In

our proof-of-concept investigation, we use a simple, artificial terrain with cube-shaped objects. Our results show that the proposed approach works robustly for the tested scenarios, and that the calculated drone position based on the identified features is indeed close to its real position.

1.4 Outline

The rest of the thesis is structured as follows. Chapter 2 gives an overview of related work. Chapter 3 provides an overview of the approach and discusses the implementation in more detail. Chapter 4 describes the experimental setup used to test the approach, while Chapter 5 presents and discusses the results of the evaluation experiments. Finally, Chapter 6 concludes the thesis and points to directions for future work.

Chapter 2

Related Work

2.1 Visual Odometry

Precise positioning of UAVs is a fundamental challenge and one of the top priorities for robots in general. To achieve self-guided movement, monitor movement, and obstacle regognition and avoidance, a UAV must continually update its position. Visual odometry is a reliable method used for this application [7], [8]. It enables it to accurately determine its location using just a continuous series of pictures taken by a camera mounted on the vehicle. It is considered a cost-effective and relatively accurate alternative to traditional odometry techniques like GPS, INS, and wheel odometry [9].

2.1.1 Types

Visual odometry can be classified according to the type of camera sensor utilized to estimate the robot trajectory [10]. There are monocular, stereo, omnidirectional and RGB-D Visual odometry techniques.

1. **Monocular Visual Odometry:** This system utilizes a single camera to perform visual odometry. It is low-budget and easy-to-deploy technique that is mainly used in commercial devices such as smartphones and laptops. However, these systems predominantly suffer from scale ambiguity, meaning the distance to objects cannot be inferred directly but only the relative motion of the vehicle. In addition, they might have some trouble finding the depth measurement between the different terrain conditions.

2. **Stereo Visual Odometry:** When two cameras are used together to capture depth in-

formation, it increases the quality and accuracy of distance estimation and motion. They can be well-suited for dense visual environments, allowing improved tracking. However, they should be precisely calibrated and synchronized in order to be used effectively for tasks like visual odometry. Moreover, they can be expensive and complex in deployment.

3. **Omnidirectional visual odometry:** It uses wide-angle cameras to provide a wide field of view (even up to 360°), allowing more scene information to be available, which leads to extended feature tracking from the camera's view and hence produce accurate 3D models of the environment.

4. **RGB-D visual odometry:** It utilizes color and dense depth image capturing by cameras that combine visual and depth information in real-time. They operate very well in indoor environments but are limited to short ranges.

2.1.2 Challenges

1. **Computational Cost:** Real-time visual odometry needs heavy computational resource for image processing, feature extraction, and motion estimation. This may obstruct systems with limited processing power, such as UAVs [11], [12].

2. **Lighting and Environmental Conditions:** Changes in light, like shadows, glare, or even changes in brightness, often affect the proper identification and tracking of features. Problems are further amplified when working with repetitive textures or under poor lighting conditions [11], [12].

3. **Dynamic Environments:** Moving objects and dynamic scenes can disrupt feature tracking and motion estimation. Robust algorithms should be developed for such cases, so that there is a clear differentiation between an object in motion and the background's static features to have consistent performance in a dynamic environment [11], [12].

4. **Featureless Environments:** Some environments have no distinct visual characteristics—for example, plain walls, open fields, or uniform textures like sand—making detecting and tracking features problematic. Visual odometry systems build up their estimation of motion through the use of distinctive features, and in their absence, a deterioration of the system's performance can occur.

5. **Sensor Noise and Distortion:** It is commonly known that many camera sensors have inbuilt imperfections. These can, in turn, reduce the quality of the obtained images and, therefore, the accuracy of the visual odometry. Practical techniques to calibrate and rectify this feature, must be developed to implement an effective visual odometry technique

2.1.3 Approaches

Estimating the location of a UAV, which in our carried out by visual odometry, can be done in three ways: using a feature-based method, an appearance-based method, or a combination of the two.

Feature-based approach

This visual odometry approach involves identifying certain image features like corners, lines, and curves in a pair of consecutive image frames, after which they are matched through feature tracking to estimate motion [13]. This can be done by using algorithms like SIFT (Scale-Invariant Feature Transform) or SURF (Speeded-up Robust Features), which search for points of interest in the images and extract descriptors that can be compared across different frames [14]. Motion is estimated based on the computed displacement vectors of the matched features. Kalman filters are often added to refine the accuracy through the fusion of visual odometry data with outputs from the IMU (Inertial Measurement Unit), thus improving the estimates.

Appearance-Based Approach

The appearance-based approach for VO is based on differences in pixel appearance and brightness intensity between frames rather than focusing on specific feature points [15]. This approach often uses optical flow techniques, which estimate camera motion by considering the variations in pixel brightness patterns over the frames [16]. Dense optical flow algorithms, like the Horn-Schunck method [17], compute displacement for all pixels, while sparse optical flow methods, such as the Lucas-Kanade technique [18], focus on selected pixels to reduce noise and improve robustness. Furthermore, in this approach, template matching has become very popular [19], where one searches for a patch from the current image in the

next frame by sliding the template over the search area and calculating similarity measures such as sum of squared differences (SSD) or normalized cross-correlation (NCC).

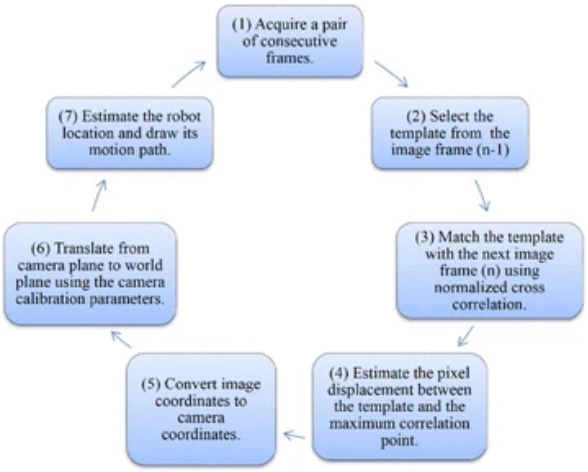

Figure 2.1: Flowchart of visual odometry system algorithm.

Hybrid of Feature-and Appearance-Based Approaches

The hybrid approach uses the strengths of both feature-based and appearance-based methods. By tracking points of interest in consecutive image frames and simultaneously utilizing pixel intensity information, this approach enhances robustness in dynamic environments. This combined strategy will be particularly advantageous in those environments where any one method may fail. Feature-based methods are not very effective in low-texture environments like sandy soils or concrete. In contrast, appearance-based methods do not perform well in occluded images and sudden lighting change situations. That makes the hybrid approach much more of a versatile solution for the problem: dealing with different textures and lighting conditions effectively, and thus becoming suitable for complex applications like autonomous navigation in diverse terrains.

2.2 Our Approach

Our approach is based on monocular feature-based VO method but is entirely different from the methodologies primarily used in this area. It utilizes advanced feature-matching methods to achieve great accuracy and high computational efficiency. Essentially, our method does not rely on visual odometry to calculate a drone's trajectory. Instead, it uses feature-based techniques from visual odometry to accurately determine the drone's location by de-

tecting and matching features. Then, by gathering enough data from ascertaining the position we can use it for any drone mission we want, like navigating to a position without the use of GPS or simply return to its home location.

2.3 Approaches for feature detection

Feature detection is one of the key aspects of the feature-based approach of visual odometry. It identifies distinctive and stable visual elements, or features, in an image that can be tracked or matched reliably in a different frame. Several feature detection algorithms have been devised over the years for a very robust and reliable characteristic identification. Some of the most common feature detection algorithms are the Scale-Invariant Feature Transform (SIFT) [20], Speeded-Up Robust Features (SURF) [21], Features from Accelerated Segment Test (FAST) [22], Binary Robust Invariant Scalable Keypoints (BRISK) [23], Oriented FAST and Rotated BRIEF (ORB) [24].

1. **Scale-Invariant Feature Transform (SIFT):** The SIFT algorithm was developed by David Lowe in 1999 for detecting and describing local features in images. SIFT is invariant to image scale and rotation, being robust to changes in illumination and minor affine distortions. It works through a set of stages: firstly, it detects local extrema of scale space using a Difference of Gaussians (DoG) function to identify potential key points. Next, it accurately localizes these key points and assigns an orientation based on local image gradients. Finally, a descriptor at each key point is calculated by examining gradient orientations in the local neighborhood of that point. Despite being a robust algorithm, SIFT has high computational cost and cannot be easily used for real-time applications.

2. **Speeded-Up Robust Features (SURF):** SURF is a robust image descriptor, approximating and enhancing SIFT. Devised by Herbert Bay and his team in 2006, SURF has been engineered to speed up calculations while upholding reliability and precision. Integral images make SURF faster at calculating box-style convolution filters, thus reducing processing time considerably. The method identifies important points by utilizing a measure based on a Hessian matrix and calculates descriptors by adding up Haar wavelet responses within a specific area surrounding the point of interest. The

speed advantages of SURF make it particularly valuable for live applications, although it remains more computationally expensive than newer algorithms.

3. **Features from Accelerated Segment Test (FAST):** This high-speed corner detection technique was developed by Edward Rosten and Tom Drummond in 2006. FAST operates by looking at a group of pixels surrounding a particular pixel. It determines whether there is a sizeable continuous cluster of pixels that are either brighter or darker than the central pixel. Its process is tremendously fast and thus very useful for live applications, though FAST does not produce descriptors. FAST is mainly combined with other algorithms for describing the features detected, such as BRIEF.

4. **Binary Robust Invariant Scalable Keypoints (BRISK):** The BRISK algorithm, developed by Stefan Leutenegger and his team in 2011, is a method for detecting and describing features that is efficient and robust. BRISK identifies critical points through a scale-space pyramid and generates binary descriptors by analyzing local intensity gradients. The algorithm's fast performance and efficient use of resources make it ideal for mobile and embedded systems.

5. **Oriented FAST and Rotated BRIEF (ORB):** ORB is derived from the FAST keypoint detector and the BRIEF descriptor. Ethan Rublee, along with his group, developed this design in 2011; the notion was to create a design that was efficient with low computational cost for real-time applications. The FAST detector detects keypoints by scanning a circle of pixels surrounding a specific pixel. The BRIEF descriptor characterizes these points with binary strings by comparing intensities in a smoothed image patch. ORB builds on this by adding orientation information to the descriptors to achieve rotation invariance and, in the process, provides good performance and efficiency when used in many different applications.

2.4 Feature detection method chosen

We decided to use the ORB or Oriented FAST, and Rotated BRIEF detector. This detector is considered one of the best feature detection algorithms due to its combined good performance, computational efficiency, and robustness. As stated earlier, ORB is a fusion of the very efficient FAST key point detector and the BRIEF descriptor, making it suitable for

real-time application. Including the orientation information of the BRIEF descriptors it helps the detector gain rotation invariance, while also making the descriptors have a binary format allows for efficient memory usage and quick matching. Furthermore, the features of ORB make it very effective when working in environments with variations in lighting and active scenes.

While algorithms such as SIFT and SURF are pretty accurate and reliable, they take much computational power and do not work well to real-time scenarios. While ORB excels in numerous situations, it can be substituted by algorithms such as BRISK in particular cases that demand varying compromises between speed, accuracy, and robustness. BRISK offers many advantages, such as scalable keypoints, making it a potential alternative depending on the specific needs of the application.

Chapter 3

Approach

In this chapter, we provide a high-level description of our approach, which consists of two basic phases: The first is the detection of features and assignment of geographical coordinates to them. The second is the positioning of the drone based on detected features and the assigned coordinates. We also provide some implementation details.

3.1 Overview

An visual overview of our approach is given in Figure 3.1. We briefly explain the individual steps in the following.

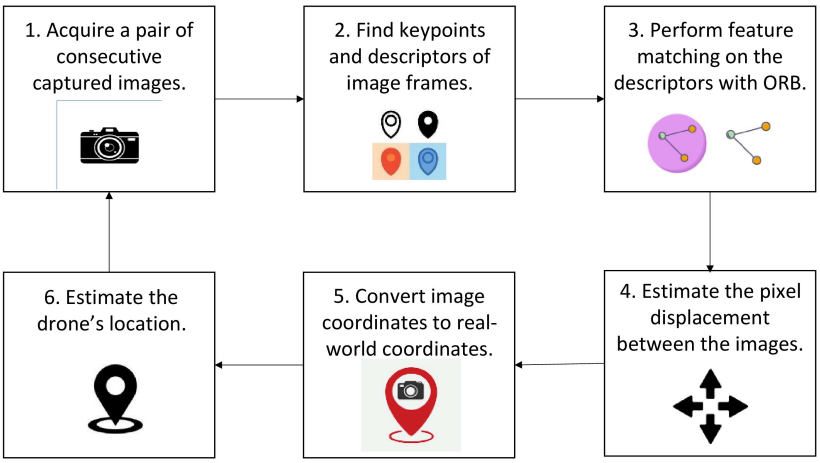

Figure 3.1: Overview of the approach.

Image capture. The first step includes image capturing. Once the drone has taken off and a steady slow speed, has been given, it begins capturing downward images and recording GPS data. Images are captured at a regular frame rate, and the drone's geographical coordinates, heading compared to true North and altitude of flight are recorded for each time frame. This is depicted in Figure 3.2.

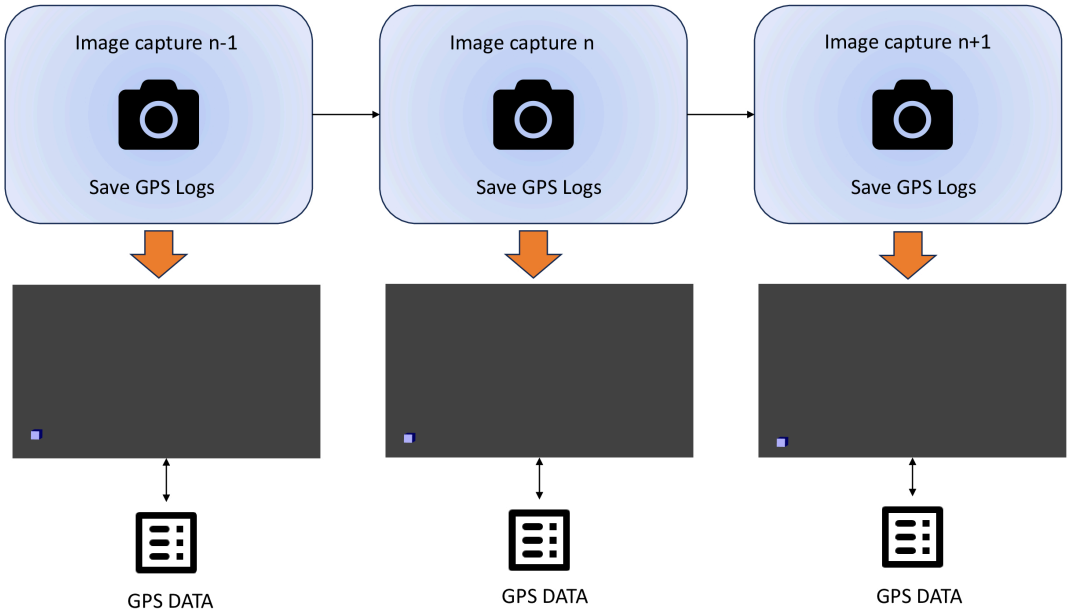

Figure 3.2: Image capture.

Afterwards, these images are used for feature detection and the drone's geographical coordinates, heading and altitude are recorded for the coordinate assignment.

Feature detection. After the image capture is finished (or during), feature detection is carried out using the ORB detector. At first, two images are chosen from the beginning of the mission, and their keypoints are computed with the FAST detector. Then, the BRIEF descriptor assigns a distinct descriptor to every keypoint. Lastly, feature matching is conducted to identify matching keypoints. This is illustrated in Figure 3.3.

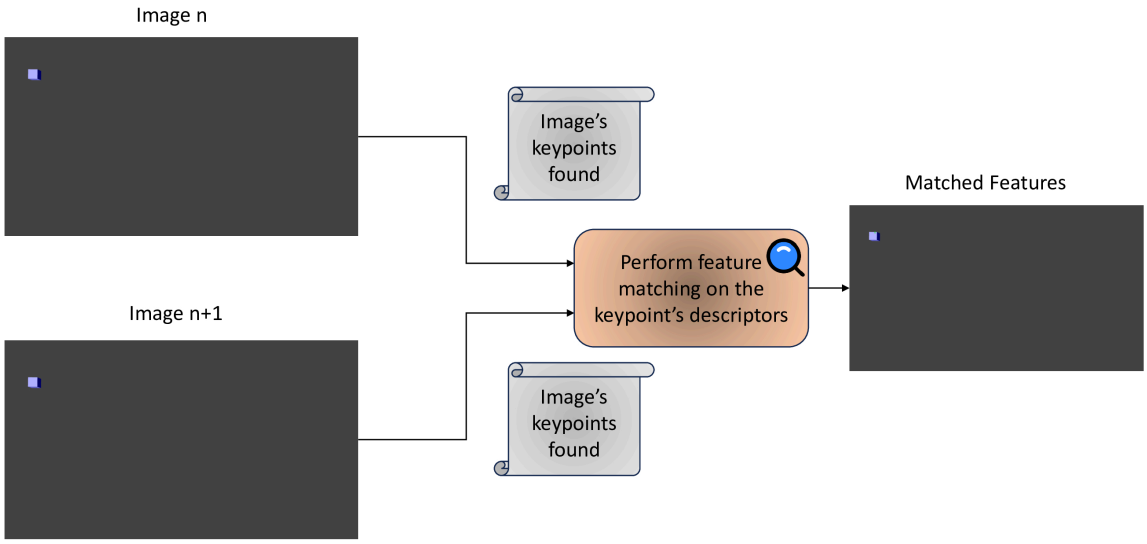

Figure 3.3: Feature detection.

Coordinate assignment. The last stage includes assigning coordinates to the detected features. The process starts with retrieving GPS information from the current image frames, including the drone's geographical coordinates, altitude, and heading. In the next step, the pixel distance of the matched keypoints from the center of the captured image, which represents the drone's actual location, is calculated. Then, by utilizing the drone's latitude, longitude, altitude heading information, the pixel measurement is converted to real-world distance, enabling the identification of the geographical coordinates of the corresponding keypoints. The described functionality is shown in Figure 3.4.

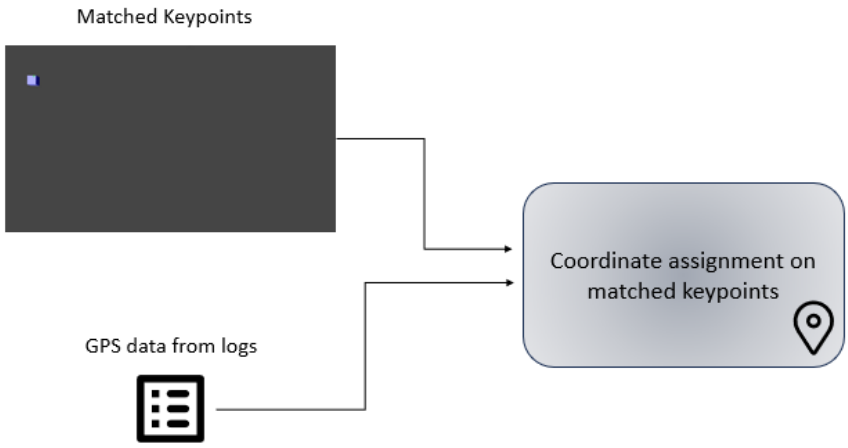

Figure 3.4: Coordinate assignment.

Drone positioning. Drone positioning essentially follows the reverse process of coordinate assignment. At first, we compute the pixel offset of the matched keypoints from the image pair relevant to the center of the image. This pixel offset is then converted into real-world distance and by using the geographic coordinates of keypoints that we calculated in the previous algorithm, we determine the latitude and longitude of the center of the image, which represents the drone's location. This is depicted in Figure 3.5.

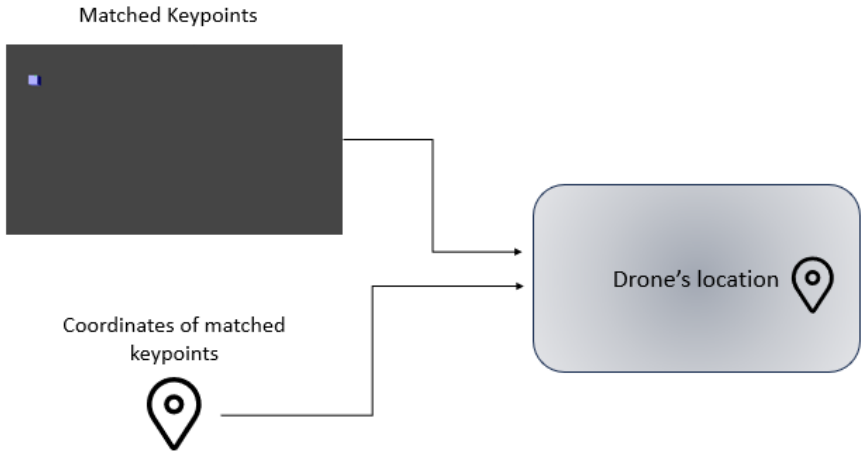

Figure 3.5: Drone positioning.

3.2 Implementation details

3.2.1 Camera parameters

In our camera configuration the camera position is under the drone and has an orientation of 90 degrees, meaning it faces downwards to the ground. It has a horizontal field of view of 90 degrees (1.57 radians) and vertical field of view of approximately 58.7 degrees (1.02 radians). The image output has a resolution of 1280x720 pixels in the B8G8R8 format (colored images). The camera is always active and updates at 30 frames per second.

3.2.2 Feature detection algorithm

As we discussed before we use the ORB feature detector to perform the feature detection on the captured images. The feature detector produces two results:

1. **Keypoints:** In ORB, keypoints are specific distinct points in an image that can be consistently recognized in various images. These points usually consist of corners or

edges, representing regions in the image with high variance. The FAST algorithm is utilized by ORB for detecting these keypoints.

2. **Descriptors:** Descriptors are strings of binary code that represent the visual characteristics of the region including color or rotation that surround a specific point of interest. ORB utilizes the BRIEF descriptor, which encapsulates the the feature's patterns within a small area surrounding each keypoint.

To achieve lower memory usage and faster performance we detect only one feature with the ORB detector. This occurs in memory usage of 48 bytes per keypoint and 32 bytes for each descriptor for a total of 80 bytes. The descriptor, within its 32 bytes, encapsulates all the information for each keypoint, including color, rotation, and intensity. The pseudocode can be depicted in Listing 3.1.

```
# Initiate ORB detector
orb = cv.ORB_create(nfeatures=1)

# Find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)
```

Listing 3.1: Keypoint detection pseudocode.

Afterward, we perform feature matching, where we compare the descriptors of the keypoints. We use a Brute-Force matcher for this work. Essentially, it takes the descriptor of one feature in first set and is attempting to match it with all other features in second set using some distance calculation. At the end, the closest one is returned.

The matching process involves a bit-by-bit comparison of the descriptors and if there are above a set percentage we deduct that the descriptors match and continue with the next one. We use the Hamming distance as a measurement of similarity and a cross check between the descriptors to yield better results. The Hamming distance is a measure of similarity between two strings of equal length. It counts the number of positions at which the corresponding symbols (characters or bits) are different. Cross check is a technique where:

1. The i-th feature in set A matches best with the j-th feature in set B
2. And the j-th feature in set B also matches best with the i-th feature in set A.

This means that both features must be each other's best match. This is described in Listing 3.2.

```
# Create BFMatcher object with cross check enabled for better results
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1, des2)

# Sort them in the order of their distance.
matches = sorted(matches, key=lambda x: x.distance)
```

Listing 3.2: Feature matching pseudocode.

3.2.3 Data structure and logic for coordinate assignment

In this section, we examine the data structure used in the coordinate assignment, how each entry is utilized during the process as well as the keypoint management algorithm during the assignment. The data structure is depicted in Listing 3.3 and the corresponding image illustrating the logic behind the data structure is shown in Figure 3.6.

```
class FeaturesData:
 def __init__(self, descriptor, appearances, lat, long, heading, image
     , keypoint, index):
      self.descriptor = descriptor
      self.appearances = appearances
      self.avg_lat = lat
      self.avg_long = long
      self.heading = heading
      self.coordinates = [(lat, long)]
      self.image = image
      self.keypoint = keypoint
      self.index = index

 def update(self, new_lat, new_long, keypoint):
      self.keypoint = keypoint
```

```
        self.avg_lat = (self.avg_lat * self.appearances + new_lat) / (
            self.appearances + 1)
        self.avg_long = (self.avg_long * self.appearances + new_long) / (
            self.appearances + 1)
        self.appearances += 1
        self.coordinates.append((new_lat, new_long))
```

Listing 3.3: Coordinate assignment data structure.

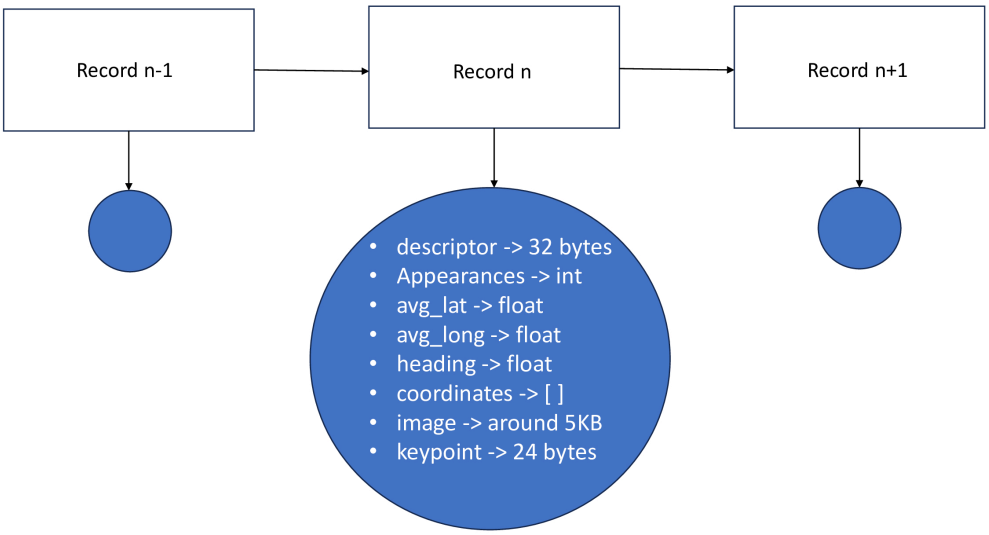

Figure 3.6: Data structure analysis in coordinate assignment.

1. **descriptor:** The descriptor string functions as a distinct identifier for every matched feature that was found. It functions both as an identifier and as a reference point for finding any other matching descriptors. As mentioned earlier, every descriptor takes up a space of 32 bytes.

2. **appearances:** The number of appearances for each descriptor is tracked. During the coordinate assignment process, if a previously detected descriptor is found again, we increment its counter and update its coordinates (the update process will be analyzed in the next entries). The number of appearances is an integer and it occupies 4 bytes in the memory.

3. **avg_lat:** The average latitude of the keypoint. This is calculated during the update process if a previously detected feature is found again and it is used mainly for evaluation

and as a measure of accuracy. It occupies 8 bytes of memory space as a float.

4. **avg_long:** The average latitude of the keypoint. This is calculated during the update process if a previously detected feature is found again and it is used mainly for evaluation and as a measure of accuracy. It takes 8 bytes of memory as it is clssified as a float.

5. **heading:** The initial heading of the drone when it detected the matched feature for the first time. This information is primarily used as a reference point to accurately locate the feature in ambiguous images and also for the evaluation process. It is a float so it takes up 8 bytes of memory space.

6. **coordinates:** The pair of coordinates (latitude and longitude) for each instance when the specific descriptor is detected and matched is recorded here. This is used mainly for evaluation purposes and each pair occupies 16 bytes.

7. **image:** The image of the first instance of the descriptor. This entry as well as the heading, are used for evaluation and as a measure of accuracy. Each image is around 5KB.

8. **keypoint:** The keypoint of the matched feature is also recorded. This is mainly used during the drone positioning process to know the keypoint's position in the image. As mentioned before 48 bytes in the memory.

9. **update script:** The update script is triggered when a previously detected feature is found again and performs three functions. First, it updates the keypoint so that the correct one can be used during the drone positioning process. Second, it recalculates the average latitude (avg_lat) and average longitude (avg_long) of the feature. Third, it updates the coordinates list with the newly found latitude and longitude and increments the appearance count by one.

A representation of the pseudocode used for managing the keypoints during the coordinate assignment phase is depicted in the following algorithmic block:

Algorithm 1 Keypoint management during coordinate assignment.

Find the **existing descriptors** that are already recorded in the data structure
Calculate the **angle** of the drone movement relevant to true north
updated feature ← **None**
for each match in the two images **do**
    find the **keypoint** of each match
    Calculate **pixel offsets** relevant to the center of the image and true north
    Convert to **geographical offsets** based on the flight altitude and the camera parameters

    keypoint latitude, keypoint longitude ← Calculate keypoint's coordinates
    if existing descriptors list is not empty **then**
        Find **descriptor matches** with current keypoint's descriptor using matcher
        if matches with descriptor from the list **then**
            Update **existing feature** from the list with the coordinates of current feature
            updated feature ← existing feature
            continue
        end if
    end if
    Create and append **new feature**
    updated feature ← new feature
end for
return updated feature

3.2.4 Data structure and logic for drone positioning

In this section, we analyze the data structure used in the drone positioning process, how each entry is utilized as well as the keypoint management algorithm used. The data structure is depicted in Listing 3.4 and the corresponding logic behind the data structure is shown in Figure 3.7.

```
class DroneData:
 def __init__(self, image, drone_lat, drone_long):
     self.image = image
```

```
        self.drone_lat = drone_lat
        self.drone_long = drone_long
```

Listing 3.4: Keypoint detection pseudocode.

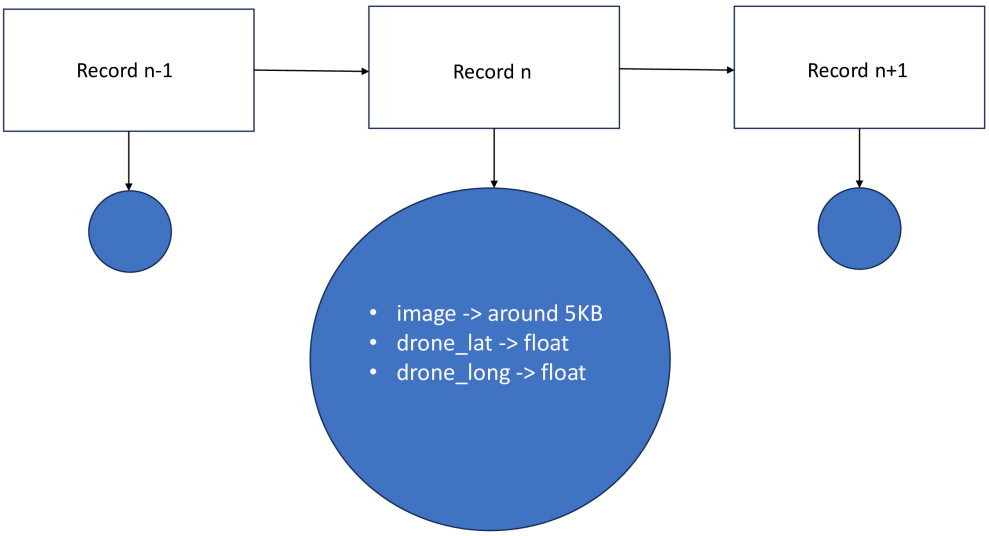

Figure 3.7: Data structure analysis in drone positioning.

1. **image:** This is the second image of the pair used during the feature detection and matching process (the first image can also be used; the choice is random). This image is also the one recorded during the coordinate assignment. It occupies approximately 5KB of memory.

2. **drone_lat:** The drone's latitude, obtained through the drone's positioning algorithm for the specific image pair used in the feature detection, matching process. It is stored as a float and occupies 8 bytes of memory.

3. **drone_long:** The drone's longitude, as calculated by the drone positioning algorithm for the specific image pair used in the feature detection, matching process. It occupies 8 bytes of memory as it is classified as a float.

The next algorithmic block depicts thoroughly the drone positioning algorithm:

Algorithm 2 Calculate Drone Position from Feature.

Require: matching feature from image pair

1: Calculate the **angle** of the drone movement relevant to true north
2: kp ← feature.keypoint
3: Calculate **pixel offsets** to the center of the image and true north
4: Convert to **geographical offsets** based on the flight altitude and the camera parameters
5: (lat, long) ← feature.coordinates
6: drone latitude, drone longitude ← Calculate drone's coordinates using **(lat, long)** and **geographical offsets**
7: **return** drone latitude, drone longitude

Chapter 4

Experimental Setup

In order to start testing our algorithm, we needed a dynamic robotics environment that could quickly execute testing scripts with minimal delay, along with a detailed software-in-the-loop setup configuration for our drone's autopilot. In order to achieve this goal, we opted for the Gazebo simulation environment and the ArduPilot system.

The next sections describe how these tools were used in combination with each other to provide a convenient setup for our experiments, and the different configurations used to evaluate our approach.

4.1 ArduPilot SITL

ArduPilot SITL (Software In The Loop) is a powerful simulation framework that allows the operation of ArduPilot's autopilot software on a computer without needing any actual hardware. It compiles the autopilot code with a standard C++ compiler to build a native executable, thus simulating effectively the autopilot's functionalities. SITL is a significant tool for testing and development; it enables developers to replicate many vehicle categories, including planes, copters, and ground rovers.

The SITL environment can be hooked up with several other simulators in which the accuracy of the simulation increases through more advanced physics and graphics. This builds the way for advanced testing of control algorithms and vehicle behaviors under wide-ranging and complex scenarios. SITL also supports emulating many different sensors and peripherals to further test sensor fusion algorithms and peripheral interactions.

We simulated the autopilot control system of the drone with the help of the SITL of

Ardupilot to give our drone the ability to fly and navigate.

4.2 Gazebo simulation environment

Gazebo is an open-source, highly advanced simulation environment created for testing and developing robotics systems under realistic scenarios. It brings to the table interactive simulation with many powerful physics and advanced rendering engines like OGRE for life-like 3D visualizations, and a complete set of sensor models such as laser range finders, cameras, and IMUs. Gazebo enables the generation and modification of robot designs and settings for the simulation. It has many simulation components backed up by the Simulation Description Format (SDF).

The environment permits both programmatic and graphical interfaces so that the user is enabled to have complete control and scrutiny of the simulations. The user can even write custom plugins that help broaden functionality and simulate complex behaviors within the virtual environment. Gazebo integrates easily with the Robot Operating System (ROS), and it makes it trivial to test control algorithms and robotic behaviors in a simulated environment before deploying them in the reael world. Such an integration then allows more complex robotics tasks, such as autonomous vehicle maneuvering and coordination of many robots.

We used Gazebo to set our experimental environment with the drone model spawned inside as seen in Figure 4.1

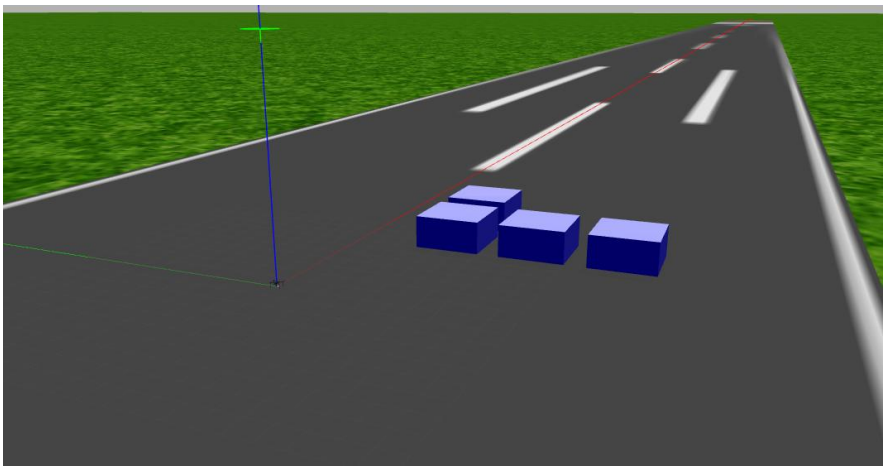

Figure 4.1: Gazebo environment setup.

The drone model we chose for executing and testing our algorithms is the IRIS quadcopter [25]. The IRIS quadcopter is an all-in-one, ready-to-fly UAV developed by 3D Robotics.

It integrates advanced autonomous abilities from the all-in-one in-built powerful Pixhawk autopilot system. It has various unique features like multiple controls (RC, computer, phone, and tablet), in-built telemetry to monitor the mission in real time, robust construction with Zytel Nylon arms and feet, various custom commands and functionalities. It provides GPS waypoint navigation for professional applications, and its compatibility with GoPro cameras for aerial imaging is unparalleled. The IRIS quadcopter is crafted to ensure easy use and flexibility to make it a perfect candidate across a wide array of aerial imaging applications.

Gazebo supports the physics model of various drones, including the IRIS quadcopter, enabling detailed and realistic simulations of drone flight dynamics. The IRIS quadcopter is created in Gazebo, including accurate properties of physics and aerodynamic characteristics of the actual drone. Overall, this comprehensive modeling of the drone will ensure that the behavior of the IRIS quadcopter is quite realistic compared to real-world performance.

Thus, we implemented the Iris quadrotor drone model in Gazebo and also connected it with the drone autopilot system. Figure 4.2 accurately depicts the coupling between Gazebo and ArduPilot SITL.

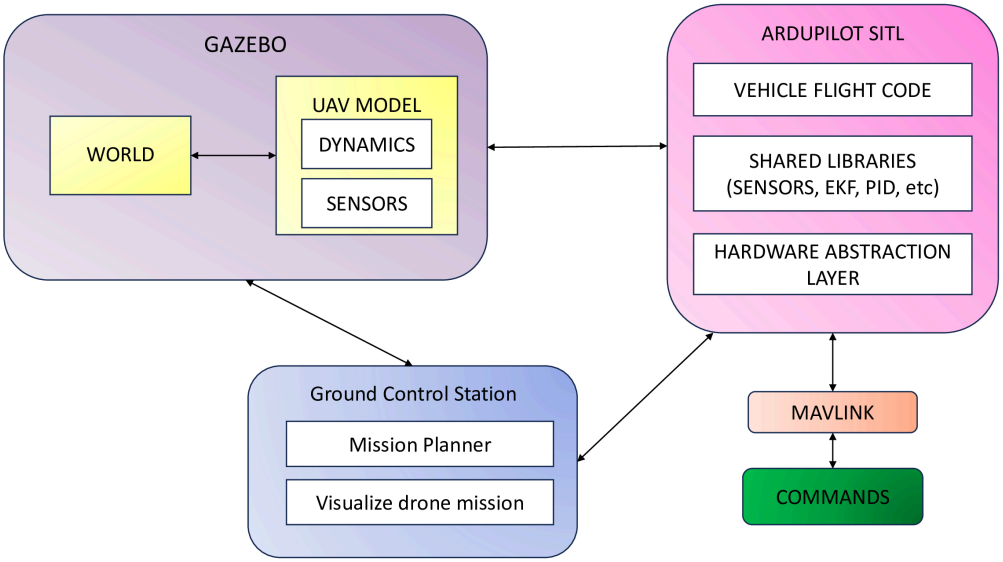

Figure 4.2: Gazebo-SITL control loop.

As mentioned before, the user can produce certain plugins to broaden the experience inside Gazebo. Thus, he can easily provide virtual cameras and virtual camera streams plug-ins, constituting an effective aide in the simulation and testing of visual sensing systems for

robotics. Simulated devices within Gazebo, like virtual cameras, simulate a physical camera by producing image data from a simulated scene. They can be configured to have varying properties such as desired resolution, field of view, and frame rate, which allows them to closely model the capabilities of physical cameras utilized in robotic applications.

The integration of Gazebo with the Robot Operating System (ROS) also enhances the functionalities of virtual cameras because one can easily communicate between the simulation environment and ROS software.

Figures 4.3, 4.4, 4.5 illustrate the camera mounted on the drone as well as an image taken inside the Gazebo environment.

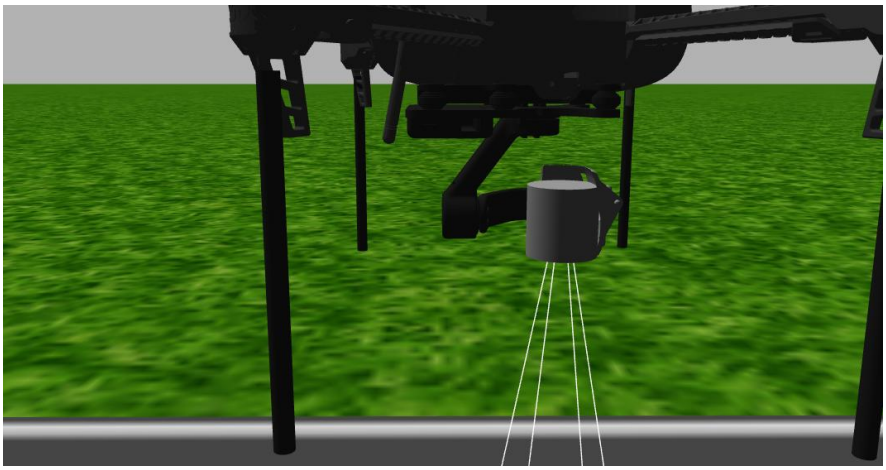

Figure 4.3: Drone camera created in Gazebo environment.

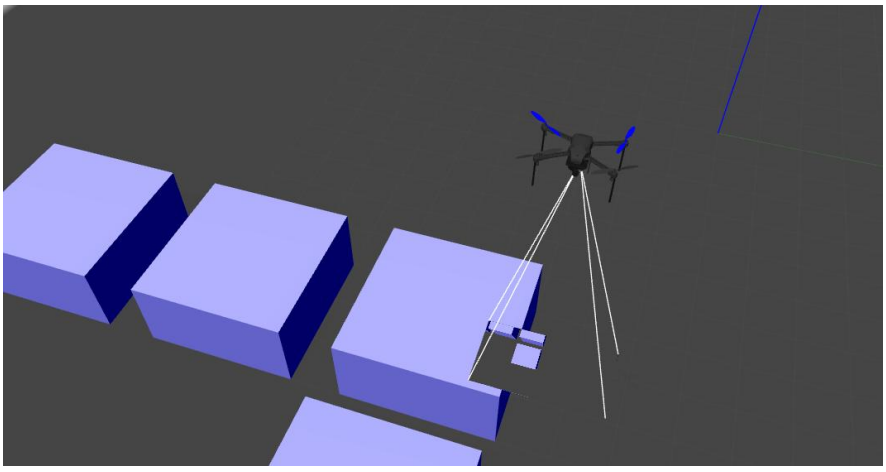

Figure 4.4: Camera view inside Gazebo environment.

Figure 4.5: Image captured from the Gazebo camera.

4.3 Experimental parameters

In this section, we offer a detailed explanation of the parameters used in our evaluation testing trials.

4.3.1 Drone mission

First, we present a description of the drone mission selected for execution. The goal of this mission was to collect enough data to successfully carry out the feature detection and matching algorithm. Basically, the drone performs a square scan inside a predetermined area, imitating a lawnmower's pattern. Such a pattern will scan the area effectively, providing the maximum number of found features. We selected a square area of 20 meter by 20 meter with a spacing of 2 meters between horizontal movements. We also set the drone's speed to a low level to minimize tilt on the z-axis, ensuring more accurate data collection. Also, we present the type of missions we run to evaluate our algorithm's performance. There are two types of missions we executed:

1. Missions with only one object under different sizes and altitudes.

2. Missions in certain altitude but with different number of objects, colors and sizes.

These mission types can easily showcase both the benefits in performance and potential issues with our algorithm.

4.3.2 Objects

As stated earlier, the ORB detector is corner-based and is orientation invariant thus it yields very accurate results. We set up our Gazebo environment with cubes. The reasons that we chose to use cubes are:

1. **Ease of setup:** Cubes are straightforward to configure and position in the simulation environment.

2. **Precise corner location** We are able to accurately determine the geographical coordinates of the corners because we know their exact positions.

Furthermore, it is essential to experiment with our algorithm using cubes of varying sizes as well as different or identical colors. Evaluating the algorithm's performance under various levels of complexity and clutter is crucial to ensure its ability to handle real-world scenarios with multiple objects. Last but not least, it is important to test our algorithm at various flying altitudes to ensure its effective performance under different flying conditions.

Thus the experimental parameters that occurred are:

1. **Size of cubes:** We found that four different cube sizes were needed to fully address all testing scenarios. Those are 0.5x0.5x0.5, 1x1x1, 2x2x2, 3x3x3.

2. **Number of cubes:** We tested our algorithm for one cube as well as two, three and four simultaneously.

3. **Color of cubes:** The color of the cubes varied depending on the setup. We selected both identical colors and a range of different colors, including blue, green, red, black, and white.

4. **Flying altitude:** We concluded that a flying altitude of ten and fifteen meters would suffice our testing needs.

Thus, as previously mentioned, we created two execution scenarios for the drone to follow:

1. **One cube scenario:** In this scenario, we chose to focus on a single cube with varying sizes (0.5x0.5x0.5, 1x1x1, 2x2x2, 3x3x3) and flying altitudes of ten or fifteen meters. This serves as a solid starting point to effectively observe the performance of our algorithm. We executed 5 runs for each different parameter combination.

2. **Multiple cubes scenario:** Transitioning towards a more real-world scenario, we decreased the range of cube sizes, kept a flying height of fifteen meters, and modified the number and color of cubes. In particular, we experimented with arrangements containing two, three, and four cubes of the same or different colors, in dimensions of 0.5x0.5x0.5 and 3x3x3. This test is intended to further evaluate how well the algorithm performs in more intricate and realistic scenarios. We launched 3 runs for each different parameter combination.

Chapter 5

Evaluation

In this chapter, we test the correctness of our algorithm, showcasing and discussing the results of the experiments we conducted.

5.1 Accuracy of coordinate assignment algorithm

In this section, we will discuss how the accuracy of the coordinate assignment process is calculated. Essentially, we had two alternatives to ascertain the accuracy:

1. Calculate the accuracy after processing a pair of images.

2. Complete the drone mission, then calculate the accuracy based on the average geographical coordinates of the matched features.

We went the second method because it provides a more precise pinpointing of the geographic coordinates of the features. Using the average as a measurement, when a feature is spotted 20 times, the algorithm's average latitude and longitude estimations will be more accurate to the real geographical coordinates than with just one detection. Basically, the accuracy is calculated by comparing the latitude and longitude of the geographical coordinates of the algorithm with the expected geographical coordinates that are pre-determined during the drone mission execution. For the Coordinate assignment only statistically relevant results are taken into account (>10 detection) and thus used for the drone positioning algorithm.

5.2 Accuracy of drone positioning algorithm

The drone positioning algorithm essentially reverses the process of the coordinate assignment algorithm. Therefore, the geographical coordinates calculated by the algorithm after processing each image pair will be the same as those logged by the drone during the mission. Consequently, it is more meaningful to calculate the average drone position and thus the accuracy after the drone mission ends and only based on the statistically relevant results given from the coordinate assignment algorithm. The accuracy is calculated like in the coordinate assignment algorithm: we compare the average drone position with the expected drone position and record the results.

5.3 Results with respect to accuracy

In this section, the results of our algorithm will be showcased with respect to accuracy. We examine the data gathered from different test situations, and explore the implications of our findings. This extensive evaluation will emphasize the advantages of our algorithm and offer understanding into the factors causing any detected inaccuracies. Besides reporting the deviations in latitude and longitude, we also calculate the displacement of the estimated vs actual position by using the Pythagorean theorem.

We start by reporting the accuracy of coordinate assignment to detected features. Then, we investigate the accuracy of drone position estimation based on the coordinates of the featured detected.

5.3.1 Accuracy of coordinate assignment to detected features

Missions with 1 cube

The accuracy of coordinate assignment for 1 cube missions for 10 and 15 meter meter flying altitude and different cube sizes are shown in Table 5.1 and Table 5.2 respectively.

cube size	0.5x0.5x0.5	1x1x1	2x2x2	3x3x3
lat diff	0.110867m	0.183870m	0.348381m	0.170365m
lon diff	0.339840m	0.500890m	0.734040m	0.748697m
pos diff	0.357467m	0.533571m	0.812517m	0.767836m

Table 5.1: 1 cube mission coordinate assignment results for 10 meters.

cube size	0.5x0.5x0.5	1x1x1	2x2x2	3x3x3
lat diff	0.207781m	0.274891m	0.400693m	0.393544m
lon diff	0.473451m	0.604239m	0.803220m	1.046857m
pos diff	0.517039m	0.663830m	0.897618m	1.118386m

Table 5.2: 1 cube mission coordinate assignment results for 15 meters.

Based on the accuracy results, we can deduce the following observations:

1. **Smaller cube sizes yield better results:** This fact is accurate because the exact distance between the drone and the object is not known in advance. Only the flying altitude is passed as an argument in the coordinate assignment algorithm. Therefore, the result will be more precise with the objects closer to the ground.

2. **Smaller flying altitudes result in better accuracy:** As a result of the drone's close proximity to the ground, a more significant amount of image detail is captured, which produces more accurate results.

3. **Longitude difference is more significant than latitude difference:** This is true because drone scanning occurs in north and south movement headings, and we use the geodesic formula for our longitude calculations. The square area is small and does not present curvature.

Missions with several cubes

Next, tackling more real-world scenarios, we present the accuracy results of coordinate assignment for multiple cubes placed in the simulation environment, featuring different colors, cube size of 0.5x0.5x0.5 and 3x3x3 and a flying altitude of 15 meters. The results are depicted in Table 5.3 and Table 5.4 respectively.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.207781m	0.197219m	0.191361m	0.214300m
lon diff	0.473451m	0.402408m	0.403002m	0.365007m
pos diff	0.517039m	0.448138m	0.446128m	0.423266m

Table 5.3: Multiple cube mission coordinate assignment results with different color and 0.5x0.5x0.5 cube size.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.393544m	0.575790m	0.507088m	0.561334m
lon diff	1.046857m	0.929633m	0.958152m	0.870638m
pos diff	1.118386m	1.093504m	1.084063m	1.035909m

Table 5.4: Multiple cube mission coordinate assignment results with different color and 3x3x3 cube size.

By analyzing the results we can observe:

1. **Multiple cubes do not affect the algorithm's performance:** This was the expected result because the detected features are more easily recognized due to their different colors.

2. **Smaller cube sizes yield better results:** As mentioned before, this fact holds true because the precise distance between the drone and the object is not known in advance; only the flying altitude is predetermined and passed as an argument in the coordinate assignment algorithm.

Lastly, in the same context as before we present the accuracy results of coordinate assignment for multiple cubes placed in the simulation environment, with same colors, cube size of 0.5x0.5x0.5 and 3x3x3 and a flying altitude of 15 meters. The results are shown in Table 5.5 and Table 5.6 respectively.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.207781m	0.258936m	0.199025m	0.284591m
lon diff	0.473451m	0.527467m	0.421902m	0.415564m
pos diff	0.517039m	0.587596m	0.466489m	0.503672m

Table 5.5: Multiple cube mission coordinate assignment results with same color and 0.5x0.5x0.5 cube size.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.393544m	0.428905m	0.616601m	0.592975m
lon diff	1.046857m	0.996016m	1.154091m	0.921184m
pos diff	1.118386m	1.084439m	1.308481m	1.095536m

Table 5.6: Multiple cube mission coordinate assignment results with same color and 3x3x3 cube size.

From the results that occur we can deduce:

1. **Multiple cubes do not affect the algorithm's performance:** Even with same colored cubes, the accuracy remains consistent throughout multiple cubes. This means that the algorithm can efficiently adapt into real-world scenarios.

2. **Smaller cube sizes yield better results:** As mentioned before, this fact also holds true in this scenario

5.3.2 Accuracy of drone positioning

Missions with 1 cube

Here, we will present and discuss the accuracy results of drone positioning for 1 cube missions for 10 and 15 meter meter flying altitude and varying cube sizes. As mentioned before, we use the statistically relevant results from the coordinate assignment algorithm to calculate the drone positioning and accuracy. These are shown in Table 5.7 and Table 5.8 respectively.

cube size	0.5x0.5x0.5	1x1x1	2x2x2	3x3x3
lat diff	0.110240m	0.136899m	0.241489m	0.185644m
lon diff	0.060810m	0.057921m	0.054396m	0.033543m
pos diff	0.125899m	0.148648m	0.247540m	0.188650m

Table 5.7: 1 cube mission drone positioning results for 10 meters.

cube size	0.5x0.5x0.5	1x1x1	2x2x2	3x3x3
lat diff	0.086664m	0.096970m	0.128114m	0.140953m
lon diff	0.060579m	0.049418m	0.054053m	0.031751m
pos diff	0.105738m	0.108836m	0.139050m	0.144485m

Table 5.8: 1 cube mission drone positioning results for 15 meters.

By observing the results, we see that they are quite accurate. The different cube sizes do not significantly affect the drone positioning algorithm, as expected, and the differences observed are minimal.

Missions with several cubes

As before, we present the accuracy results of drone positioning for multiple cubes placed in the simulation environment featuring different as well as identical colors, varying cube sizes and a flying altitude of 15 meters. You can observe the results in Tables 5.9, 5.10, 5.11, 5.12.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.086664m	0.101034m	0.099811m	0.127053m
lon diff	0.060579m	0.066487m	0.106468m	0.079839m
pos diff	0.105738m	0.120947m	0.145937m	0.150056m

Table 5.9: Multiple cube mission drone positioning results with different color and cube size 0.5x0.5x0.5.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.140953m	0.157050m	0.159656m	0.111708m
lon diff	0.031751m	0.022533m	0.038859m	0.056811m
pos diff	0.144485m	0.158658m	0.164317m	0.125324m

Table 5.10: Multiple cube mission drone positioning results with different color and cube size 3x3x3x.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.086664m	0.082464m	0.108981m	0.092618m
lon diff	0.060579m	0.057965m	0.078820m	0.071306m
pos diff	0.105738m	0.100799m	0.134497m	0.116887m

Table 5.11: Multiple cube mission drone positioning results with same color and cube size 0.5x0.5x0.5.

nof cubes	1 cubes	2 cubes	3 cubes	4 cubes
lat diff	0.140953m	0.107819m	0.067329m	0.164512m
lon diff	0.031751m	0.047731m	0.020839m	0.072762m
pos diff	0.144485m	0.117912m	0.070480m	0.179884m

Table 5.12: Multiple cube mission drone positioning results with same color and cube size 3x3x3x.

Similar to the single cube missions, the results here are also pretty accurate. Different cube numbers, sizes or colors do not seem to affect the results and the algorithm seem that can be integrated to real-world scenarios.

5.4 Results with respect to performance

Next, we will present the results with respect to performance. The tests that were conducted were aimed at measuring the times to complete our algorithm over the course of a single drone mission. This is important to know not only on a high-end PC but also on a

board that can directly fit on the drone itself so that the algorithm can be effectively integrated and utilized as part of the drone's onboard software. In particular, we have to evaluate the average time per iteration and also the maximum iteration time. By doing that, we should be able to estimate if the algorithm can apply to a Raspberry Pi board. Thus, these evaluations enable us to appreciate how well the algorithm works on the different hardware platforms, giving a more comprehensive insight into its performance.

The specifications of both the PC and also the Raspberry Pi 4 board are depicted in Table 5.13

Specs	**Personal Computer**	**Raspberry Pi 4**
CPU	AMD Ryzen 7 5800X 8-Core @3.80 GHz	ARM Cortex-A72 6-Core
RAM	16 GB	Needs 8GB

Table 5.13: System specifications

Note that the PC is powerful enough that whether feature detection, feature matching, the coordinate assignment and the drone positioning algorithm are performed on a pair of images or not (meaning that it can't find suitable descriptors), the iteration time remains practically the same. On the other hand, the Raspberry Pi 4 board experiences a much longer iteration time if appropriate descriptors are found, as opposed to when descriptors are not found. To focus on the critical case, for the RPi we only report iteration times where one or more features are detected.

The minimum, maximum and average computing times for a single iteration (coordinate assignment to features / drone positioning based on feature coordinates) in missions with a single and several cubes on each system are given in Table 5.14. In addition, the total time for a whole drone mission to be processed by our algorithm is reported as well.

Times (sec)	One Cube PC	Multiple Cubes PC	One Cube Rpi	Multiple Cubes Rpi
Minimum	0.01054	0.01106	0.45097	0.45020
Maximum	0.12344	0.14868	0.92482	0.99993
Average	0.01844	0.02027	0.50912	0.51289
Total	18.94449	20.64238	305.21710	487.77970

Table 5.14: Comparison of average times (in seconds) for one-cube and multiple-cube missions on PC and Rpi.

From the results, we can observe that our algorithm can easily be executed in the high-end PC. Also, as the average iteration times are not exceeding one second and are around half a second on the board, it means it can be actually executed on the board and be integrated into real world scenarios for further testing. The total time of the Raspberry pi is greater than the PC and we can't know if it will cause problems in the physical world. Lastly, we notice that the multiple cube missions can actually take more time than the single cube ones. This fact holds true because there are more features to process and that can cause the greater delay.

5.5 General observations

In this section, we will discuss about general observations regarding the accuracy results, mainly concentrating on the causes of potential inaccuracies:

1. **System Inaccuracies:** Errors occurring from the drone's sensors like the GPS or IMU (i.e rounding errors), can introduce inaccuracies in the positioning data.

2. **Drone Tilt in the Z Axis:** Tilt during movement can distort image capture during the drone mission, contributing to reduced accuracy. That is why we opted for lower speeds, to reduce the tilt.

3. **Image Quality:** Poor image quality can affect feature detection and matching, leading to errors in coordinate assignment and drone positioning processes.

4. **Algorithm inaccuracies:** Inaccuracies in the algorithm, such as accounting for the Earth's curvature when working in a small, relatively flat patch of land, can lead to inaccuracies in position estimation.

Chapter 6

Conclusion

In this work, we have investigated the idea of using feature detection on aerial images taken during flight time to provide navigation information in case the GPS signal is lost, In a first phase, detected features are assigned coordinates based on the drone's GPS position. In a second phase, the GPS coordinates of the features are used to calculate the current position of the drone. Using a suitable software-in-the-loop setup, we have performed experiments for several scenarios where we vary the size and number of objects in the mission terrain, showing that the proposed approach can achieve robust results in all cases.

Our work can be used as a basis to pursue further directions. On the one hand, it is important to test the approach using more realistic terrains that resemble the real world. Note, however, that this can be highly application-specific, depending on whether the drone operates above urban areas, land or sea. On the other hand, our implementation can be properly integrated into the autopilot system, in the form of an auxiliary GPS that is considered only when the primary GPS encounters reception issues. This way, the drone could continue its mission or return to land in a transparent way, via the usual high-level goto and return-to-land commands.

Bibliography

[1] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek. Autonomous uav surveillance in complex urban environments. In *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 82–85. IEEE, 2009.

[2] Muhammad Imran Majid, Yunfei Chen, Osama Mahfooz, and Wajahat Ahmed. Uav-based smart environmental monitoring. In *Employing Recent Technologies for Improved Digital Governance*, pages 317–335. IGI Global, 2020.

[3] Praveen Kumar Reddy Maddikunta, Saqib Hakak, Mamoun Alazab, Sweta Bhattacharya, Thippa Reddy Gadekallu, Wazir Zada Khan, and Quoc-Viet Pham. Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges. *IEEE Sensors Journal*, 21(16):17608–17619, 2021.

[4] Khairul Nizam Tahar and Sarah Kamarudin. Uav onboard gps in positioning determination. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B1:1037–1042, 06 2016.

[5] ArduPilot. `https://ardupilot.org/`.

[6] Gazebo Simulation Environment. `https://gazebosim.org/`.

[7] Mohammad Omar Abouelela Aqel, Mohammad Hamiruce Marhaban, M.I. Saripan, and H.I. Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1897, 2016.

[8] Davide Scaramuzza and Friedrich Fraundorfer. Tutorial: visual odometry. *IEEE Robot Autom Mag*, 18(4):80–92, 2011.

[9] Noureldin Aboelmagd, Tashfeen Bakary Karmat, and Jacques Georgy. *Fundamentals of inertial navigation, satellite-based positioning and their integration*. Springer, Berlin, 2013.

[10] David Valiente García, Luis Fernández Rojo, Alvaro Gil Aparicio, et al. Visual odometry through appearance-and feature-based method with omnidirectional images. *J Robot*, pages 1–13, 2012.

[11] Navid Nourani-Vatani and Paulo Borges. Correlation-based visual odometry for ground vehicles. *J. Field Robotics*, 28:742–768, 09 2011.

[12] Rafael Gonzalez, Francisco Rodriguez, Jose Luis Guzman, et al. Control of off-road mobile robots using visual odometry and slip compensation. *Advanced Robotics*, 27(11):893–906, 2013.

[13] Andrea Cumani. Feature localization refinement for improved visual odometry accuracy. *International Journal of Circuits, Systems and Signal Processing*, 5(2):151–158, 2011.

[14] Houssem Benseddik, Oualid Djekoune, and Mahmoud Belhocine. Sift and surf performance evaluation for mobile robot-monocular visual odometry. *Journal of Image and Graphics*, 2:70–76, 01 2014.

[15] Yang Yu, Cédric Pradalier, and Guanghua Zong. Appearance-based monocular visual odometry for ground vehicles. In *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 862–867, 2011.

[16] J. Campbell, R. Sukthankar, and I. Nourbakhsh. Techniques for evaluating optical flow for visual odometry in extreme terrain. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 4, pages 3704–3711 vol.4, 2004.

[17] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.

[18] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.

[19] Min-Seok Choi and Whoi-Yul Kim. A novel two stage template matching method for rotation and illumination invariance. *Pattern Recognition*, 35:119–129, 01 2002.

[20] Deepanshu Tyagi. Introduction to sift(scale invariant feature transform). https://medium.com/@deepanshut041/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40.

[21] Deepanshu Tyagi. Introduction to surf (speeded-up robust features). https://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e.

[22] Deepanshu Tyagi. Introduction to fast (features from accelerated segment test). https://medium.com/@deepanshut041/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65.

[23] Hasan Alsulaiman. Feature-matching using brisk. https://medium.com/analytics-vidhya/feature-matching-using-brisk-277c47539e8.

[24] Deepanshu Tyagi. Introduction to orb (oriented fast and rotated brief). https://medium.com/@deepanshut041/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf.

[25] ArduCopter. https://www.arducopter.co.uk/iris-quadcopter-uav.html.