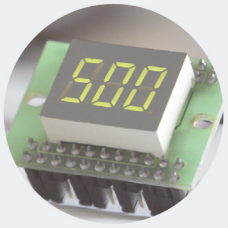
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






GNC autónomo de um drone para exploração planetária

TIAGO ALEXANDRE SOUSA DE SÁ PEREIRA
julho de 2025

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Autonomous drone GNC for planetary exploration

Tiago Alexandre Sousa de Sá Pereira

Master in Electrical and Computer Engineering
Specialization Area of Autonomous Systems

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

July, 2025

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Autonomous Systems.

Candidate: Tiago Alexandre Sousa de Sá Pereira, No. 1191075, 1191075@isep.ipp.pt

Scientific Guidance: Manuel Silva, mss@isep.ipp.pt

Scientific Co-Guidance: Marcelo Petry, marcelo.petry@inesctec.pt

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

July, 2025

Para Deus, os meus pais e namorada

Acknowledgements

Gostaria de expressar a minha mais sincera gratidão ao meu orientador, Professor Manuel Silva, e ao meu coorientador, Professor Marcelo Petry, pela oportunidade de desenvolver este trabalho.

Agradeço não só pelos conselhos científicos e acadêmicos, mas também pela confiança depositada e pelo incentivo contínuo ao longo de todas as fases do desenvolvimento deste trabalho.

Abstract

Planetary exploration poses unique challenges due to unknown terrain, communication delays, and the necessity for autonomous operation in harsh environments. This thesis focuses on the development and implementation of an autonomous Guidance, Navigation, and Control (GNC) system for a drone intended to support planetary surface missions. The primary objective is to achieve real-time, reliable, and efficient navigation and control in Global Navigation Satellite Systems (GNSS)-denied environments, such as Mars.

The proposed system integrates onboard sensor fusion for localization—leveraging monocular odometry—with path planning algorithms to enable efficient trajectory generation. A central design consideration is feasibility: the selection of sensors and components is based on current or near-future space-qualified technologies, with realistic performance characteristics accounted for throughout development.

System performance was validated in a simulated environment that incorporated representative planetary terrain and constraints. This work contributes to the advancement of autonomous aerial robotics for future space missions, enhancing the capabilities of planetary surface exploration and scientific data collection.

Keywords: Guidance, Navigation, and Control (GNC), Global Navigation Satellite Systems (GNSS)-denied environments, monocular odometry, path planning, feasibility.

Resumo

A exploração planetária apresenta desafios únicos devido ao terreno desconhecido, aos atrasos nas comunicações e à necessidade de operação autónoma em ambientes adversos. Esta dissertação foca-se no desenvolvimento e implementação de um sistema autónomo de Guidance, Navigation, and Control (GNC) para um drone concebido para apoiar missões de superfície planetária. O principal objetivo é permitir uma navegação e controlo em tempo real, fiáveis e eficientes, em ambientes sem Global Navigation Satellite Systems (GNSS), como é o caso de Marte.

O sistema proposto integra fusão sensorial a bordo para localização — recorrendo à odometria monocular — com algoritmos de planeamento de trajectórias para geração eficiente de percursos. Um dos principais focos do design do sistema é a viabilidade: a seleção de sensores e componentes baseia-se em tecnologias atuais ou de futuro próximo, qualificadas para aplicações espaciais, tendo sempre em consideração características de desempenho realistas ao longo do desenvolvimento.

O desempenho do sistema foi validado num ambiente simulado que incorpora terreno e restrições representativas de um cenário planetário. Este trabalho contribui para o avanço da robótica aérea autónoma em futuras missões espaciais, reforçando as capacidades de exploração da superfície planetária e de aquisição de dados científicos.

Palavras-Chave: Guidance, Navigation, and Control (GNC), ambientes sem Global Navigation Satellite Systems (GNSS), odometria monocular, planeamento de trajetórias, viabilidade.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

2.5D	2.5-Dimensional
2D	2-Dimensional
3D	3-Dimensional
APL	Applied Physics Laboratory
ATE	Absolute Trajectory Error
BRIEF	Binary Robust Independent Elementary Features
CEO	Chief Executive officer
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
DLR	Deutsches Zentrum für Luft- und Raumfahrt
DoF	Degrees of Freedom
DRAM	Dynamic Random Access Memory
DSU	Data Storage Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
EFC	Estimated Final Coordinates
EKF	Extended Kalman Filter
ESA	European Space Agency
ETD	Estimated Travel Distance
EZ	Exploration Zone
FAST	Features from Accelerated Segment Test

FC	Flight Control
FoV	Field of View
FPGA	Field Programmable Gate Array
FU	Freie Universität
GC	Goal Coordinates
GNC	Guidance, Navigation, and Control
GNSS	Global Navigation Satellite Systems
GPU	Graphics Processing Unit
HazCam	Hazard Avoidance Camera
HiRISE	High Resolution Imaging Science Experiment
HRSC	High Resolution Stereo Camera
IMU	Inertial Measurement Unit
JAXA	Japan Aerospace Exploration Agency
JPL	Jet Propulsion Laboratory
LC	Loop Closure
LiDAR	Light Detection And Ranging
LoS	Line of Sight
LRF	Laser Rangefinder
LVDS	Low-Voltage Differential Signaling
MCU	Micro Controller Unit
MER	Mars Exploration Rover
MIL-STD-1553	Military Standard 1553
MLR	Multilevel Relaxation
MSE	Mean Square Error
MU	Monocular Usage
NASA	National Aeronautics And Space Administration

NavCam	Navigation Camera
ORB	Oriented FAST and Rotated BRIEF
PanCam	Panoramic Camera
PMO	Planetary-Mass Object
PoC	Proof of Concept
PULP	Parallel Ultralow Power
PWM	Pulse-Width Modulation
R-CNN	Region-based Convolutional Neural Networks
RADAR	Radio Detection And Ranging
RAM	Random Access Memory
RANSAC	RANdom SAmple Consensus
RCE	Rover Compute Element
RE	Rotational Error
RFC	Real Final Coordinates
RMSE	Root Mean Square Error
ROI	Region of Interest
ROS	Robot Operating System
RRT	Rapidly Exploring Random Tree
RTD	Real Travel Distance
RTE	Return-To-Earth
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SoC	System on Chip
SpaceX	Space Exploration Technologies Corporation
SSD	Solid-State Drive
SURF	Speeded-Up Robust Features

SVD	Singular Value Decomposition
TMU	Texture Mapping Unit
ToF	Time of Flight
TRL	Technology Readiness Level
TRN	Terrain Relative Navigation
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
UKF	Unscented Kalman Filter
USA	United States of America
VCE	Vision Compute Element

Chapter 1

Introduction

Contrary to common misconceptions, space exploration has far-reaching implications beyond the aerospace sector, significantly influencing everyday life. The innovations driven by space research have led to breakthroughs across various fields, including the development of solar panels, water purification systems, cancer therapies, and lightweight materials, among others [1, 2]. For example, in the early 1990s, a team from National Aeronautics And Space Administration (NASA) Jet Propulsion Laboratory (JPL) led by Eric Fossum first developed Complementary Metal-Oxide-Semiconductor (CMOS) image sensor technology — originally intended to enhance spacecraft performance [3, 4]. This technology now forms the foundation of modern digital cameras, an integral part of today's consumer electronics. The impact of space technologies on the everyday world is often overlooked and taken for granted, yet it is deeply embedded in modern society. Space exploration inherently demands the pursuit of more efficient and advanced scientific, technical, and technological methods. This is largely due to the unique challenges posed by the harsh conditions of outer space and planetary environments, which require robust and highly reliable solutions [1]. Additionally, financial and technical constraints have driven NASA engineers to develop systems that are not only more capable but also lighter and more efficient [3]. There are many more direct and indirect benefits risen from space exploration, which cover a wide range of topics, as can be seen in Table 1.1.

One cannot discuss space exploration without addressing the search for extraterrestrial life and humanity's understanding of its place in the universe. On that note,

Table 1.1: Direct vs. indirect benefits of space exploration (adapted from [1])

Direct Benefits	**Indirect Benefits**
Scientific knowledge	Economic prosperity
Technical competence is improved	Safety & security
Innovation is transferred across fields	Environmental benefits

why should nations invest the technological and financial resources necessary to explore Mars? NASA and the European Space Agency (ESA) tackle this question by emphasizing that these missions extend beyond merely searching for alien life [5, 6]; it encompasses a far more profound purpose. It seeks to understand the roots of humankind's origins, as well as Earth's past, present, and future, while potentially answering the most pressing question: *Are we alone in the universe?* Elon Musk delves deeper into this topic, emphasizing that the efforts of Space Exploration Technologies Corporation (SpaceX) to take humans to Mars are crucial for making humanity a multi-planetary species, which he argues is essential for our long-term survival in the possibility of a mass-extinction event [7, 8].

SpaceX's website features a quote by its Chief Executive officer (CEO), Elon Musk, that, in my opinion, perfectly addresses what being a multi-planetary species means to humankind [9]:

> "You want to wake up in the morning and think the future is going to be great—and that's what being a spacefaring civilization is all about. It's about believing in the future and thinking that the future will be better than the past. And I can't think of anything more exciting than going out there and being among the stars."
>
> Elon Musk

On a personal note, I believe that people's interest in exploring the cosmos is an inherent part of being human. It stems from humankind's boundless curiosity about what exists beyond Earth and, perhaps even more intriguingly, what possibilities lie ahead. The future of humankind's presence across the solar system is bright, but the journey is long and invokes our tenacious nature.

1.1 Problem Statement

Exploring planetary surfaces, particularly in challenging terrains such as gorges, canyons, craters and caves presents significant challenges for current robotic systems like rovers. While these rovers have shown capabilities in traversing diverse

landscapes, they often face limitations due to environmental obstacles and the complexities of Martian terrain. Rovers, such as Perseverance, will avoid terrain that would tilt the rover more than 30 degrees [10], for instance, but many planetary features exceed this steepness, making direct exploration difficult. In response to these challenges, NASA is developing innovative rover concepts, such as the DuAxel, as shown in Figure 1.1, which can rappel down steep walls and access otherwise unreachable areas [11, 12]. These advancements are promising, yet they still underscore the limitations of ground-based rovers in the exploration of complex geological formations.

Figure 1.1: DuAxel prototype rover (credit: NASA/JPL-Caltech)

Aerial vehicles, such as drones, do not face the same constraints as rovers, since they, for instance, are restricted to ground surface movement. In contrast, drones can operate freely in three-dimensional space, allowing them to bypass obstacles entirely and access areas that are otherwise unreachable. This capability significantly expands the range of environments drones can explore. Their freedom of movement allows them to traverse terrain much faster than any ground-based Martian rover, which currently moves at a top speed of approximately 152 $\mathrm{m\,h^{-1}}$ (as with Perseverance [13]). The applicability of drones in planetary exploration by NASA is just beginning to unfold, notably with the recent deployment of Ingenuity and the anticipated launch of Dragonfly by the end of the decade, which will be explored in detail in Section 4.3. Aerial vehicles, however, face unique challenges, such as shorter operational times compared to rovers. As a result, they often serve as complementary tools rather than outright superior alternatives, with each platform offering distinct advantages, depending on the mission requirements and objectives.

Consequently, it is essential to develop Guidance, Navigation, and Control (GNC) architectures that enable these drones to operate autonomously, independent of human intervention, in various Planetary-Mass Object (PMO) environments. Section

4.1 will delve deeper into the environmental specifics addressed in this thesis and provide further insights into the associated challenges.

1.2 Objectives

As indicated by NASA [14], the success of the Ingenuity mission opens up a new realm of ambitious future Mars missions utilizing aerial vehicles, enabling access to terrain that is difficult for rovers to reach and facilitating the transportation of payloads between sites, for instance. This thesis will focus on one of these potential missions, addressing its associated challenges while remaining faithful to the historical context and technological advancements relevant to Mars exploration. It is crucial to ensure that the selected technologies are practical and applicable for future Mars missions, thereby avoiding choices that lack relevance or feasibility in this context.

The primary objective of this thesis is to develop a GNC system for an autonomous drone designed to access terrains that are inaccessible to rovers, such as gorges, steep cliffs, and deep craters. While Mars is the primary focus of this research, many components of the proposed system could be adapted for use on other PMO such as Europa, Titan, and the Moon. Thus, the overarching goal of this research is to equip a drone with semi-autonomous capabilities for planetary exploration. Building upon these goals, the specific objectives of this research include:

- Conduct an analysis of system requirements and constraints;
- Develop a GNC architecture;
- Focus on error handling and recovery strategies;
- Integrate sensor fusion techniques;
- Design and implement vision algorithms;
- Simulate the GNC algorithms to validate performance;
- Test and evaluate the system's capabilities in semi-autonomous mode.

It must be noted that the primary focus of this thesis lies in the guidance and navigation modules, while control falls outside the intended scope of this work. Although a control module will be developed, it will serve as a simplified implementation designed solely for the purpose of validating the other modules. Considering the complexities associated with operating in the atmosphere of another planet, the control system warrants its own dedicated research effort, ideally conducted by a specialized engineer.

1.3 Tests and Validation

Throughout the development of this project, several tests must be conducted, both independently and in an integrated manner. That is, performance should be evaluated in isolation as well as in synchrony with the complete system.

All tests are to be conducted in a simulated environment that closely resembles the operational conditions expected for the drone. Additionally, sensor noise levels should reflect those of previously tested and validated sensors.

The final validation phase includes the following requirements:

- The drone's performance must not fall below the worst-case performance recorded by Ingenuity, of 5 m from the desired landing location, which occurred during a failure to properly deliver camera images to the navigation module [15].
- The drone must be capable of navigating through a predefined set of waypoints before reaching its final destination. These waypoints may include altitude changes, simulating tasks or mission objectives that require operating at varying heights.

1.4 Dissertation Outline

This dissertation is organized into the following chapters:

- **Chapter 1: Introduction**
  Presents the research objectives, the problem that boosted this thesis, and the structure of the document.
- **Chapter 2: Overview of GNC Systems**
  Presents the reader some important theoretical information that is relevant to GNC systems.
- **Chapter 3: Literature Review**
  Reviews relevant research on PMO exploration and GNC systems. Identifies gaps in the existing work.
- **Chapter 4: Mission and System Requirements**
  Defines the mission objectives, system requirements and constraints based on prior/future missions.
- **Chapter 5: System Design**
  Describes the design and implementation of the GNC algortihms;
- **Chapter 6: Tests and Results**
  Presents the results and performance evaluation of the GNC system.

- **Chapter 7: Conclusion**
  Summarizes the research contributions and suggests directions for future work.

Part I

Background and Foundations

Chapter 2

Overview of GNC Systems

Although space robotics is a prominent theme of this thesis, NASA's robotic system will be explored later in Section 4.3. This chapter serves as a foundation for the concepts that will be explored later on.

GNC systems, sometimes referred to as GN&C, are designed to control the movement and positioning of a system—ranging from autonomous vehicles to missiles and aircraft—while simultaneously determining its localization. These systems encompass three main key components [16, 17, 18]:

- **Guidance:** Responsible for determining the desired path of travel from the vehicle's original location to a designated target. It addresses the fundamental question: *How do I reach the destination?* This includes solving problems related to identifying the optimal trajectory to achieve the desired location, known as path planning.

- **Navigation:** Responsible for determining various state information, which encompasses the system's current location, velocity, and orientation, depending on the context. The navigator primarily focuses on fundamental questions such as: *Where am I? What is my velocity? Where am I facing?*

- **Control:** Defined as the manipulation of forces and steering controls necessary to execute guidance commands while ensuring that the system operates smoothly and remains stable throughout its operation.

Figure 2.1 presents the general high-level architecture of GNC systems. Here, Navigation holds a central role, as all other modules rely on it. This centrality arises from the fact that the system's understanding of its environment is developed within the Navigation module, as will be further explained in the following sections.

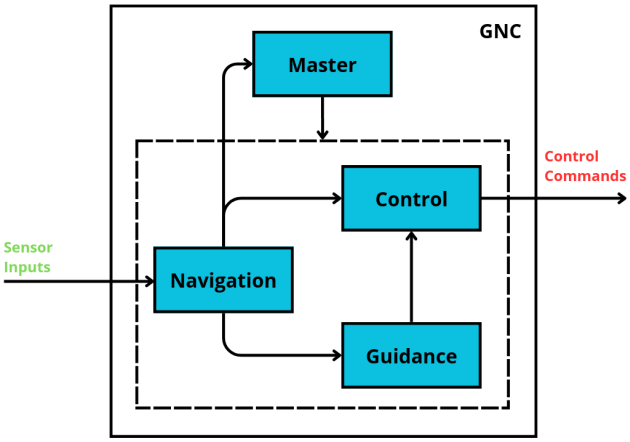


Figure 2.1: GNC architecture

Additionally, over the course of the following sections, a few other fundamental concepts to GNC system will be introduced.

2.1 Perception

Perception involves interpreting and understanding an environment through the use of sensors [19], a task that remains one of the most challenging in the fields of robotics and autonomous systems. This is particularly crucial in space environments, as well as in any robotic setting, because if a mobile robot cannot accurately perceive its surroundings, its ability to operate effectively becomes significantly compromised. GNC systems rely heavily on the data acquired from these sensors, especially within the Navigation module.

Robotic environments are often inherently unpredictable, with various constraints complicating perception. For instance, software can only process a finite amount of information due to storage and computational limitations, while sensors, such as cameras, are restricted by their field of view and resolution, and are inherently limited by what they can observe—they cannot, for example, see through walls [20]. Similarly, Light Detection And Rangings (LiDARs) are less effective in adverse weather conditions, as their beams can be scattered or deviated by water droplets, making them suboptimal in such environments. Additionally, sensors are subject to noise and fluctuations, which introduce uncertainty and errors in the data they provide, making it theoretically impossible to achieve perfectly accurate readings.

To mitigate the disadvantages associated with individual sensor types, one effective strategy is to incorporate heterogeneous sensors—those with independent error characteristics [21]. This approach enhances overall perception reliability by leveraging the strengths of different sensor modalities to compensate for each other's limitations. Another effective strategy is to employ sensor fusion, which aims to combine data from multiple sensors to produce a more accurate and reliable outcome with reduced uncertainty.

Despite the numerous techniques that may be employed to reduce error and increase the system's confidence in a given reading, the fundamental reality remains: perfect accuracy cannot be achieved.

2.2 Estimation

Since a deterministic approach to robotics doesn't deal well with uncertainties, the focus is on probabilistic approaches. The central point around estimation is the need to describe uncertainty, which is crucial since, as explained the previous Section, sensors have errors. Estimation is at the heart of the Navigation module, as it will allow the system to determine its state.

This section will introduce several key concepts that form the foundation of estimation techniques in the field of robotics.

Bayes filters

Bayes filters aim to calculate the belief distribution, $Bel(x)$, using both control and measurement data [20]. This probabilistic framework enables the system to continuously update its understanding of the state x_t as new measurements z_t and control inputs u_t become available, effectively accounting for the uncertainties present in sensor readings and control actions. One might naturally assume that a given state x_t depends on all previous states as well as all past measurements and controls. Thus, the evolution of the state could be expressed as:

$$Bel(x_t) = p(x_t|x_{0:t-1}, z_{0:t-1}, u_{0:t}) \tag{2.1}$$

However, Bayes filters leverage the Markov Assumption, which posits that the previous state x_{t-1} encapsulates all relevant information about the system's history, including $z_{0:t-1}$ and $u_{0:t-1}$ [22]. Therefore, it is sufficient to know x_{t-1}, where the past is accounted for, along with the current control action u_t. Consequently, the belief distribution can be simplified to:

$$p(x_t|x_{0:t-1}, z_{0:t-1}, u_{0:t}) = p(x_t|x_{t-1}, u_t) \tag{2.2}$$

The same principle applies if one attempts to model the process by which measurements are generated, where the knowledge of the state x_t is sufficient for predicting an observation z_t [20]. This relationship can be expressed as:

$$p(z_t|x_{0:t}, z_{0:t-1}, u_{0:t}) = p(z_t|x_t) \tag{2.3}$$

Algorithm 2.1 outlines the key steps involved in implementing a Bayes filter for state estimation [20]. For a mathematical derivation of the equations presented in the algorithm, it is recommended to refer to the work of S. Thrun, W. Burgard, and D. Fox [20], which provides a comprehensive review of probability laws.

1 **procedure** BAYESFILTER($bel(x_{t-1}), u_t, z_t$)
2 **for** each state x_t **do**
3 $\overline{bel}(x_t) \leftarrow \int p(x_t \mid u_t, x_{t-1}) \cdot bel(x_{t-1})\, dx_{t-1}$ ▷ Prediction step
4 $bel(x_t) \leftarrow \eta \cdot p(z_t \mid x_t) \cdot \overline{bel}(x_t)$ ▷ Correction step
5 **end for**
6 **return** $bel(x_t)$ ▷ Return the updated belief
7 **end procedure**

Algorithm 2.1: Bayes filter

The Bayes filter, along with its various adaptations, operates on a similar principle: it involves both a prediction phase and a correction/update phase. Line 3 integrates the prior belief with the system model to forecast the expected belief about the current state, prior to incorporating any new observations. Subsequently, Line 4 refines this prediction by adjusting it based on actual measurements, thereby enhancing the belief to more accurately align with the observed data.

Linear Kalman Filter

Sometimes referred to as just Kalman filter, this method is an optimal estimator and is likely the most studied Bayes filter derivation in linear Gaussian systems. This means that the Kalman filter represents its belief by the moments parameterization, which corresponds to a mean $\boldsymbol{\mu}_t$ and a covariance $\boldsymbol{\Sigma}_t$ [20].

Similar to the Bayes filter, the Kalman filter consists of both a prediction and an update step, as can be seen in Algorithm 2.2. The parameters in the algorithm are presented in matrix form, with $\mathbf{R}_t$ and $\mathbf{Q}_t$ representing error modeling. It's worth noting, though, that some books and papers may use the opposite notation, so the reader should be mindful of this. The Kalman filter uses the same key elements as the Bayes filter: the previous state, denoted by $\boldsymbol{\mu}_{t-1}$ and $\boldsymbol{\Sigma}_{t-1}$, the control input $\mathbf{u}_t$, and the observation $\mathbf{z}_t$ [20]. Lines 3 and 4 employ a state model to predict the system's state and associated error based on the prior state and control input.

In Line 6, the Kalman Gain is calculated, representing the ratio between the model's error and the sensor's error. The formula could be represented by:

1 **procedure** KALMANFILTER($\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1}, \mathbf{u}_t, \mathbf{z}_t$)
2 **Prediction Step:**
3 $\bar{\boldsymbol{\mu}}_t \leftarrow \mathbf{A} \cdot \boldsymbol{\mu}_{t-1} + \mathbf{B} \cdot \mathbf{u}_t$ ▷ Predict the state mean
4 $\bar{\boldsymbol{\Sigma}}_t \leftarrow \mathbf{A} \cdot \boldsymbol{\Sigma}_{t-1} \cdot \mathbf{A}^\top + \mathbf{R}_t$ ▷ Predict the state covariance
5 **Correction Step:**
6 $\mathbf{K}_t \leftarrow \bar{\boldsymbol{\Sigma}}_t \cdot \mathbf{C}_t^\top \cdot (\mathbf{C}_t \cdot \bar{\boldsymbol{\Sigma}}_t \cdot \mathbf{C}_t^\top + \mathbf{Q}_t)^{-1}$ ▷ Compute Kalman Gain
7 $\boldsymbol{\mu}_t \leftarrow \bar{\boldsymbol{\mu}}_t + \mathbf{K}_t \cdot (\mathbf{z}_t - \mathbf{C}_t \cdot \bar{\boldsymbol{\mu}}_t)$ ▷ Update the state mean
8 $\boldsymbol{\Sigma}_t \leftarrow (\mathbf{I} - \mathbf{K}_t \cdot \mathbf{C}_t) \cdot \bar{\boldsymbol{\Sigma}}_t$ ▷ Update the state covariance
9 **return** $\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t$ ▷ Return updated state mean and covariance
10 **end procedure**

Algorithm 2.2: Kalman filter

$$\mathbf{K}_t = \frac{E_{model}}{E_{model} + E_{sensor}} \tag{2.4}$$

This means that if the model's error is much larger than the sensor's error, $\mathbf{K}_t \approx 1$, while if the sensor's error is much greater, $\mathbf{K}_t \approx 0$. In Line 7, this ratio—essentially a measure of confidence in both the model and the observations—is applied. Imagine a scenario where the system's model is highly accurate but the sensors are unreliable. As suggested by Equation 2.4, when the model outperforms the sensors, $\mathbf{K}_t \approx 0$. In such a case, it would make little sense to allow poor sensor data to degrade a highly accurate model prediction. Line 7 addresses this automatically. The innovation, $\mathbf{z}_t - \mathbf{C}_t \cdot \bar{\boldsymbol{\mu}}_t$, represents the difference between the sensor's observation and the expected observation [20]. If the sensors are inaccurate, this innovation will be large and would otherwise degrade the reliable prediction $\bar{\boldsymbol{\mu}}_t$. However, the Kalman Gain, being close to zero, effectively mitigates this, preventing the flawed sensor data from distorting the model's accurate prediction.

The same principle holds in the opposite scenario, where the model is highly inaccurate while the sensors are very reliable. In this case, the predicted mean $\bar{\boldsymbol{\mu}}_t$ would be a poor estimate of the true state. The Kalman Gain, with $\mathbf{K}_t \approx 1$, responds by heavily weighting the accurate sensor readings, effectively compensating for the model's inaccurate prediction.

Finally, since the Kalman filter operates under the assumption of a Gaussian distribution, the process concludes with the computation of the updated covariance. This step adjusts the uncertainty based on the newly integrated information from sensor data.

In conclusion, while the standard Kalman filter provides a powerful framework for estimating the state of a dynamic system, numerous variants exist to enhance its applicability across various scenarios. These adaptations, each tailored to specific challenges, such as non-linearities and high-dimensional systems, further expand the versatility and effectiveness of the Kalman filtering.

2.3 Path Planning

Dijkstra's algorithm, introduced in 1959, is widely regarded as the foundational method in the field of path planning [23]. Its influence extends far beyond its initial conception, laying the groundwork for numerous algorithms in network routing, robotics, geographic information systems, and artificial intelligence.

Dijkstra's method was revolutionary because it introduced an efficient way to determine the shortest path between nodes. Its core concept—systematically exploring paths based on accumulated cost—has become a blueprint for many modern algorithms, such as A* and various heuristic-driven methods. While the algorithm itself is deterministic and non-heuristic, its logic remains a fundamental component in both theoretical research and practical applications.

The primary use case of Dijkstra's algorithm is to compute the shortest path from a given source node to all other nodes in a weighted graph. This characteristic makes it highly applicable in contexts such as route optimization in Global Navigation Satellite Systems (GNSS) systems and motion planning in autonomous systems [24]. Although more specialized or faster algorithms may be used in specific scenarios, Dijkstra's algorithm remains a critical baseline against which others are often measured. Conceptually, the algorithm begins at a source node and iteratively explores neighboring nodes, always selecting the node with the smallest tentative distance. It maintains a priority queue to manage exploration order, ensuring that the most promising path is always evaluated next. This method guarantees that once a node's shortest path has been determined, it will not be updated again—a property that ensures correctness and efficiency for graphs with non-negative weights.

Summary

As shown, GNC systems encompass multiple stages and represent a multidisciplinary challenge. While this is demanding in itself—requiring familiarity with topics ranging from computer vision to Pulse-Width Modulation (PWM) control—the true difficulty often lies in developing a cohesive system in which all modules operate in harmony to ensure robust and reliable performance.

Moreover, the vast number of possible approaches for each module makes the notion of an "optimal" solution difficult to define, let alone select. In practice, it is often not about choosing the single best-performing method in isolation, but rather selecting methods that function effectively in synergy with one another.

Among these components, navigation plays the most critical role. It is responsible for positional awareness and, in some cases, the mapping and processing of

sensor data. All subsequent modules depend on the output of the navigation system, making its function central and essential to the overall correctness and stability of system behavior.

Chapter 3

Literature Review

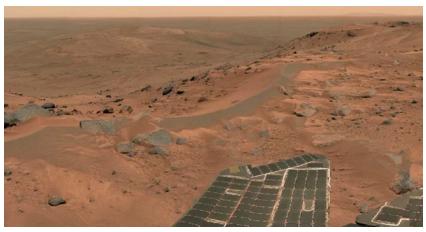
The rapid advancement of drone technology has paved the way for their integration into various sectors, ranging from infrastructure inspection [25] to environmental monitoring [26]. Autonomous drones, which operate without direct human intervention, are particularly promising due to their ability to perform complex tasks in diverse and often hazardous environments that could pose risks to humans. While the technology has matured, deploying autonomous drones in resource-constrained scenarios presents unique challenges that require innovative solutions. These scenarios—characterized by limitations in power availability, computational resources, communication bandwidth, and payload capacity—often necessitate tailored strategies to ensure that drones can effectively complete their missions. With the rapid advancement of technology, increasingly powerful processors, more precise sensors, and longer-lasting batteries are being integrated into robotic systems, including drones. While this evolution presents numerous opportunities, it is important to consider scenarios where new, advanced technologies may not be the most suitable or applicable. For instance, in missions requiring high reliability, where traditional technologies might offer more robustness compared to cutting-edge innovations.

This literature review will focus on the specific challenges of deploying autonomous robotic systems in resource-limited scenarios. Understanding and addressing these challenges will be vital to ensuring that future advancements in drone technology are both sustainable and applicable in a diverse range of situations, ultimately expanding the horizons of what autonomous systems can achieve.

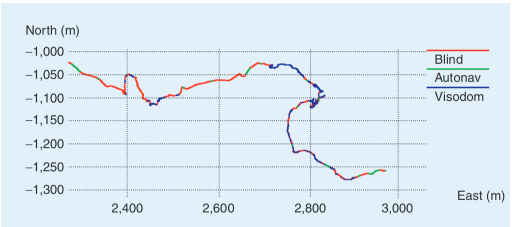
3.1 Vision Outside Earth

Rovers have a limited time window in which they can communicate with Earth. More often than not, the Mars Exploration Rover (MER) communicated with Earth only once per day, which made engineers look for ways for the rover to precisely keep track of its locomotion, as Earthly corrections were not feasible [27].

The MER was capable of estimating its motion using wheel odometry at 8 Hz; however, due to the occurrence of slippage, visual odometry was introduced to mitigate the resulting errors when the rover lost traction [27]. According to Cheng *et al.*, the visual odometry system estimates the rover's 6-Degrees of Freedom (DoF) pose by tracking terrain features across stereo image pairs in both 2-Dimensional (2D) and 3-Dimensional (3D) [27]. A maximum likelihood method is used to compute the motion estimate; however, if this estimate is considered unreliable, the system falls back on the initial pose derived from wheel odometry and the Inertial Measurement Unit (IMU). The rover will come to a halt if an excessive number of unreliable estimates or convergence failures—referred to by the authors as "unconvergences"—occur. Figure 3.1b demonstrates the extensive use of the visual odometry software for climbing Spirit through the Columbia Hills, on Mars.



(a) Columbia Hills (credit: NASA/JPL/Cornell)



(b) Plot of Spirit's traverse history using visual odometry in the Columbia Hills, on Mars [27]

Figure 3.1: Spirit and its whereabouts

The algorithm employed by the authors follows a relatively standard approach [27, 28]: it begins by detecting features across the stereo pair using a corner detector and then tracks these features across multiple images. To improve computational efficiency, several optimizations are introduced. Notably, the image is divided into multiple grids, each representing the minimum distance allowed between features. Within each grid cell, only one feature is permitted to be detected. This ensures a more uniform distribution of features across the image and reduces the formation of feature clusters. After stereo matching, the 3D positions of the detected features are computed using epipolar geometry. The uncertainty associated with each 3D point is estimated based on the feature's location, the quality of the match, and image sharpness, incorporating Jacobians and curvature data obtained from subpixel interpolation. As the rover moves, previously selected features are reprojected into the new stereo image pair using wheel odometry, and their positions are refined

through correlation-based matching. A rigidity test is applied to filter out outliers. Finally, by comparing the updated and previous 3D positions of the tracked features, motion estimation errors are corrected, and the rover's updated pose is inferred.

This whole process takes MER an average time of 3 min to compute, using their limited 20 MHz RAD6000 Central Processing Unit (CPU), hence its limited and specific use [28]. However, the employment of visual odometry not only improved target approach efficiency, it also proved crucial to maintaining vehicle safety. Given the precarious extraterrestrial test environment, the authors proclaim their ability to define the accuracy of the system is limited. However, by comparing high-resolution Panoramic Camera (PanCam)-derived motion estimates to low-resolution Navigation Camera (NavCam)-based Visual Odometry during short drives, they found the average onboard update error to be under 2 mm—within ground truth measurement uncertainty. The authors emphasize that in a qualitative sense, they know that their algorithm performs well because the rover's behavior matches their predictions better when visual odometry is enabled. Maimone *et al.* [28] finalize by stating the biggest problem is the processing speed, not the accuracy, making runtime optimization the top priority.

By shifting focus from Mars to the Moon, new concepts have been proposed for future rover navigation, while aerial drones remain a secondary consideration. Recently, driven by the renewed interest surrounding the Artemis missions, Daftry *et al.* [29] and Matthies *et al.* [30] introduced novel lunar navigation systems that rely on building a database of crater landmarks from orbital imagery—an approach that appears to follow the principles of Simultaneous Localization And Mapping (SLAM).

The proposed sensor suite for the rover includes wheel odometry, an IMU, either stereo cameras or LiDAR, and either a sun sensor or a star tracker for absolute heading measurements. According to the authors, this configuration can achieve typical position errors of less than 2%.

Perhaps more interestingly, Daftry *et al.* [29] describe their simulation methodology, which includes testing their system in a Gazebo environment developed by NASA [31], as well as in a custom-generated world built using Blender [32], among other platforms.

The algorithm follows a structured pipeline consisting of crater detection—where Matthies *et al.* [30] suggest that a monocular approach could also be feasible with the aid of a Convolutional Neural Network (CNN)—crater matching, and state estimation, the latter of which is performed using a particle filter. Del Prete and Renga [33] also explore a similar pipeline for lunar spacecraft state estimation, employing a Mask Region-based Convolutional Neural Networks (R-CNN) for crater detection and an Extended Kalman Filter (EKF) for state estimation.

Matthies *et al.* identify several limitations and areas needing further development

before the LunarNav system can be deployed on lunar missions. The algorithms are currently at an intermediate Technology Readiness Level (TRL) and require improvement—particularly in enhancing stereo-based crater localization to handle diverse and partial crater views. LunarNav also depends on sensors like LiDARs, which are not fully developed and matured for space robotics applications, but might be used in the future. The system's computing demands are expected to be manageable, but this still needs validation. Lastly, proper testing and validation are hindered by the lack of high-quality lunar terrains on Earth, with the only available site—Cinder Lake (comparison to the Moon in Figure 3.2)—being small, poorly maintained, and seasonally limited in accessibility.

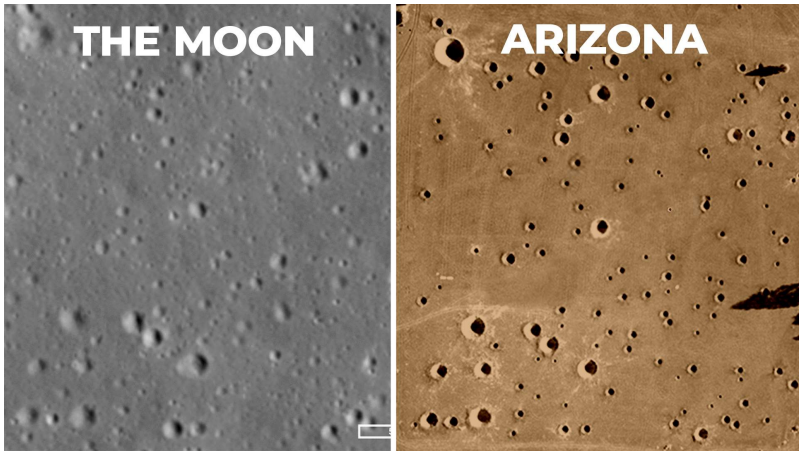


Figure 3.2: Comparison between the Moon and Cinder Lake, in Arizona, USA (credit: NASA/JPL)

3.2 Optimized SLAM Approaches

Over the years, numerous SLAM approaches have been developed, each modified to address specific constraints or requirements. Recently, as of the writing of this thesis, Chen *et al.* [34] introduced a novel edge-assisted SLAM approach. Their method involves offloading non-real-time-critical computationally intensive processes to edge servers – initially a local Dell computer with an i7-9700K processor at 3.6 GHz and an NVIDIA GTX 1080 Graphics Processing Unit (GPU), later transitioning to a laptop with slightly better specifications – via wireless communication.

The main advantage of this edge-computing-based approach is the significant reduction in the computational load on the mobile robot, helping the authors achieve a 62% decrease in tracking error. One key contribution of the work is the authors' development of an algorithm to minimize uncertainty by selecting optimal keyframes for constructing both local and global maps—a factor they emphasize as often overlooked in the literature [34]. However, this improvement came with a slight performance trade-off when compared to the original ORB-SLAM3.

Despite these promising results, the novelty of using edge computing also presents a significant limitation, particularly for autonomous robotic applications in remote environments where on-board systems must operate independently without access to external computational resources. While edge computing may be highly effective in indoor or urban scenarios, where access to edge servers is feasible, its applicability becomes questionable in the context of space missions. Even assuming the possibility of edge computing in space—a concept that is still far from established—the fundamental issue persists. Resource sharing between devices in space, where computational power and memory are severely constrained and high bandwidth with low latency is unavailable, may not adequately support the demands of full-scale SLAM implementations, such as ORB-SLAM3 [35]. The limited processing capabilities of space-based systems, combined with the need to maintain real-time functionality and system integrity, make this approach unsuitable for space exploration scenarios.

Niculescu *et al.* [36] introduced NanoSLAM, an algorithm specifically designed for micro-robots using parallel computing. Their work specifically targets ultra-compact nano-robots (Figure 3.3) designed for operation in confined indoor spaces and near humans, for instance. A key distinction from the previously presented work is that their approach eliminates reliance on external computing resources. Instead, they employ an integrated, self-contained system. By focusing on minimizing power consumption, the authors achieved notable results – mapping accuracy of 4.5 cm and a trajectory estimation error reduced by up to 67%, all with a drone weighing less than 45 g, consuming 87.9 mW, and requiring only 500 kB for the mapping pipeline, per Loop Closure (LC). The proposed solution utilizes the recent Parallel Ultralow Power (PULP) GAP9 System on Chip (SoC) by GreenWave Technologies [37] as a co-processor to support their STM32F405 unit, along with four novel VL53L5CX Time of Flight (ToF) depth sensors [38].

Figure 3.3: Drone Crazyflie 2.1 used in NanoSLAM [39]

A primary limitation of this work, when considering potential space applications,

lies in its reliance on emerging State-of-the-Art technologies. Niculescu *et al.* utilize depth sensors, asserting that vision-based SLAM approaches are not optimal for depth estimation in nano-Unmanned Aerial Vehicles (UAVs) [36]. In fact, the authors acknowledge that the use of depth cameras is a prerequisite for their system. While their solution demonstrates robust performance within a highly constrained system, the space industry tends to be cautious about adopting significant changes due to its sensitivity to failures. Additionally, given the limitations of processors used in NanoSLAM, the graph-based SLAM algorithm can simultaneously optimize up to 440 poses, resulting in a mapping capability of up to 317 m^2 for their drone. With some simple math, we can verify how this approach is not suitable for a space-mission scenario.

For a square with an area $A = 317\ m^2$, the perimeter P can be calculated as:

$$P = 4 \cdot \sqrt{A} = 4 \cdot \sqrt{317} \approx 71.12\,m \tag{3.1}$$

Using the formula for time t based on distance d and speed v, and assuming a speed of 10 $m\,s^{-1}$, we have:

$$t = \frac{d}{v} = \frac{P}{v} = \frac{71.12}{10} \approx 7.11\,s \tag{3.2}$$

This calculation indicates that the drone could complete a mission around an area of 317 m^2 in approximately 7.11 s, assuming an average velocity of 10 $m\,s^{-1}$. For comparison, an average football field has an area of over 7000 m^2. Therefore, the application of their algorithm for space UAVs is limited, as these vehicles are often designed to cover long distances.

Taking an alternative approach without relying on advanced technologies, Andreasson *et al.* introduced MiniSLAM [40]. This method uses economical components, leveraging only odometry and an omnidirectional camera for computations. Unlike conventional vision-based SLAM techniques, it avoids estimating specific landmark positions in the environment. Instead, it assesses image similarity to deduce the relative pose between frames, along with the associated uncertainty of this estimate. Rather than recording exact landmark coordinates, the authors utilize geometric correlations. In MiniSLAM, a Multilevel Relaxation (MLR) algorithm is employed, where each relation $r \in R$ represents a probabilistic description—specifically, a likelihood distribution—of the relative pose or spatial relationship between two frames, denoted as frames a and b. This probabilistic distribution provides an estimate of the position of frame a relative to frame b, expressed not as an exact distance or orientation but as a range of possible values with associated probabilities, modeled by a Gaussian distribution. These relational estimates can be derived from both odometric and visual data. Their results were not as fascinating as the previous ones, obtaining a Mean Square Error (MSE) of over 4 m in one case.

One potential limitation of this approach lies in the assumption that odometry remains highly accurate for short travel distances. While this assumption holds in most cases, it is important to note that, as will be discussed, drones operating in space scenarios may travel at high speeds (e.g., 10 $\mathrm{m\,s^{-1}}$) while having extremely limited frame rates (e.g., 1 Hz). This results in visual odometry that processes frames with considerable gaps between them, which means the distance traveled between consecutive frames is not necessarily short. Additionally, omnidirectional cameras have not been used in space drones, which tend to focus on monocular cameras and Terrain Relative Navigation (TRN). An important consideration in implementing the MLR algorithm is its assumption of a flat, two-dimensional world, which restricts the previously mentioned relations to that 2D plane.

Summary

Although technological advancements in the space industry continue to emerge—such as the implementation of visual odometry and future ambitions for SLAM-based navigation—these developments tend to progress slowly, methodically, and with great caution.

The aerospace sector is highly sensitive to failure, necessitating a high degree of robustness and extensive validation. Moreover, the unique characteristics of extraterrestrial environments require algorithms and systems to be finely tailored, making it difficult to find terrestrial systems that fully meet the demands of PMO.

Chapter 4

Mission Scenarios and Requirements

In this chapter, the system/simulation requirements and constraints will be outlined for clarity. Given the significance of these requirements and constraints for this project, they were examined in detail. Each subsequent section will specify one or more explicit requirements and constraints that guided the development of this project. For simplicity, each requirement will be denoted using the notation S-X, where X represents the specific number assigned to that requirement, and each constraint will be denoted as C-X, where X represents the specific number assigned to that constraint. The same idea applies to M-X, the mission requirements.

It is important to emphasize that any type of drone—be it a helicopter, quadcopter, hexacopter, or others—can be utilized for this work. This flexibility arises from the high-level nature of the control framework proposed in this thesis, which is intended to be a basis for future research. Low-level motor control, such as PWM of the motors, falls outside the scope of this thesis and should be the subject of future work by subsequent researchers. A detailed and dedicated research effort should be conducted on the control aspects of drones for Mars, as its atmospheric parameters—such as reduced gravity and rarefied air—present unique challenges.

The most critical consideration in the development of this thesis is ensuring reliability. In space, systems cannot afford to fail, and errors must be eliminated during development, as the space industry is highly sensitive to even the smallest mistakes. Additionally, computational resources are severely limited, as will be

demonstrated in an upcoming chapter, making it essential to carefully select the libraries integrated into the source code. Optimizations often need to be achieved in non-trivial ways, which is a major concern for this work, as it aims to be as closely aligned with real-world constraints as possible.

- **S-1: Every piece of developed software must adhere to NASA's guidelines of safe and critical code [41].**
  Reasoning: Ensuring the code's reliability is a priority for "space-code." NASA's guidelines are designed to minimize errors and enhance robustness in critical systems.

4.1 Overview Of The Target Environment

Mars has been the primary target of robotic exploration over the past few decades [42, 43, 44]. Key motivations for this exploration include the search for signs of life, the presence of water, understanding the planet's evolution, and preparing for future human exploration [6]. Among the many geological features of the red planet, Valles Marineris stands out as a remarkable structure. It is the largest valley in the solar system, nearly the size of the United States of America (USA), extending over 3,000 km, spanning up to 600 km across, and reaching depths of up to 8 km, as illustrated in Figure 4.1 [45, 46, 47].

Figure 4.1: Valles Marineris size comparison with the USA (modified)
(credit: NASA/JPL-Caltech)

One prevailing theory regarding its origins suggests that its formation resulted from a combination of vertical subsidence along normal faults, minor extensional forces associated with the nearby Tharsis volcanoes, and sedimentary processes. These geological activities likely began in the late Noachian period (4.1 to 3.7 billion

years ago) and continued through to the Amazonian period (2.9 billion years ago to the present) [48, 49].

Given its vast size and intricate geological history, Valles Marineris presents a compelling location for exploration. The Japan Aerospace Exploration Agency (JAXA), for instance, has considered a future lander or rover for the region of East Melas Chasma [50], within Valles Marineris, as shown in Figure 4.2 taken by High Resolution Stereo Camera (HRSC) onboard ESA's Mars Express spacecraft.

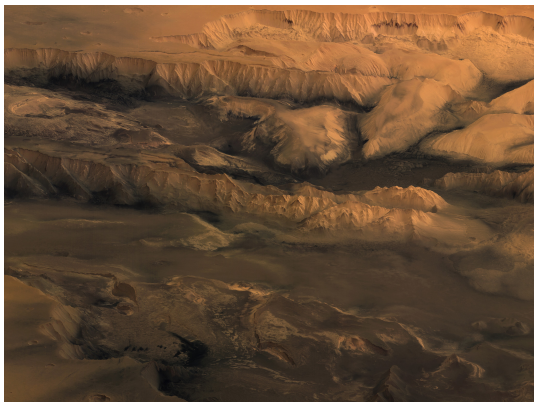


Figure 4.2: Melas, Candor and Ophir Chasmas in perspective
(credit: ESA/DLR/FU Berlin (G. Neukum))

McEwen *et al.* further discuss this region [51], indicating that it meets all landing site and rover constraints, while also emphasizing its scientific value due to factors such as:

- High preservation potential for past habitability;
- Promising locations for present habitability;
- Aqueous and groundwater processes.

Other landing sites have been proposed, such as Coprates Chasma, where the authors identify several geological Region of Interest (ROI) [52]. It is suggested that the canyon walls within the Exploration Zone (EZ) have a higher probability of detecting microbial life, should it have ever existed on Mars, as well as its early crust formation [53]. Consequently, Mojarro *et al.* propose that humans or drones could be utilized to access these hard-to-reach ROIs. Additionally, Miyamoto *et al.* propose another potential landing site, further expanding the options for exploration in this region [54]. Separately, at the 10th International Conference on Mars 2024, Fraeman *et al.* proposed a drone specifically designed for exploring the canyon, building on the success of Ingenuity [53]. This proposal further solidifies the region's potential for studying the planet's climatic conditions and assessing water availability throughout its history.

Given Valles Marineris' significance as a target for further exploration, the first mission requirements can be outlined as follows:

- **M-1: The simulated environment will closely resemble that of Valles Marineris**;
- **M-2: Communications between "Earth" and the platform are strictly limited to the beginning of the mission**;
  Reasoning: This restriction is necessary to simulate the autonomous operation of the platform, as it would occur in a real mission scenario on Mars. The vast distance between Earth and Mars results in significant communication delays, which can range from several minutes to over twenty minutes one way [55].

In compliance with M-1, the following environment was created using Blender and subsequently integrated into Gazebo:

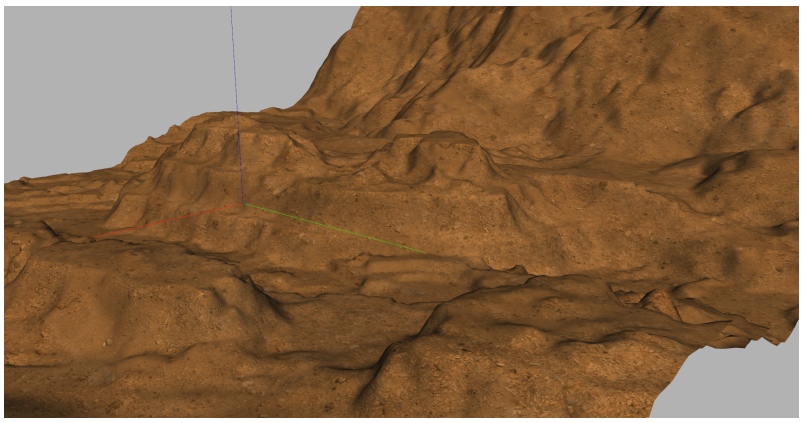


Figure 4.3: Simulated Valles Marineris environment

4.2 Environmental And Terrain Considerations

Aside from Valles Marineris' general topography which has been discussed before, there are other relevant environmental considerations that are important to simulate.

Soil texture is a critical aspect that must be simulated, as its absence could complicate the functionality of vision algorithms. Environments that are texture-less or feature repetitive textures can lead to suboptimal outputs from feature detection algorithms. A prime example of this is a smooth, texture-less white wall, where discernible features—distinct parts of an image—are challenging to identify due to the homogeneous nature of the surface. As shown in Figure 4.4, the orange surface of Mars is characterized by numerous rocks and stones. The presence of these environmental objects can significantly contribute to providing texture, enhancing the effectiveness of vision algorithms.

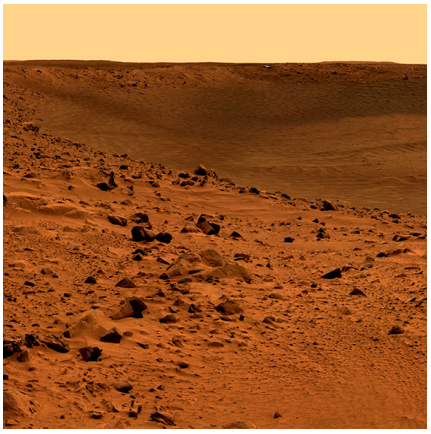


Figure 4.4: Martian surface (credit: NASA/JPL-Caltech)

Mars' solar irradiance is approximately 43% of that on Earth, making luminosity another important parameter to simulate. Haze and fog can also occur in Valles Marineris [56], though drone missions would logically be suspended during severe weather conditions. Therefore, only light occurrences of fog and haze would be relevant for simulation. Dust is another critical factor on Mars, as demonstrated by the fate of the Opportunity rover, which ceased operations after a dust storm [57].

4.3 Allowed Technologies And Constraints

In order to evaluate the technology considered in this thesis, an analysis of both past and future NASA missions to the red planet, as well as other PMO initiatives, was conducted. This is crucial because the simulated drone has to incorporate technology that is feasible and relevant for future Mars missions. With this in mind, three robotic platforms were considered for analysis: Perseverance [58], Ingenuity [59], and Dragonfly [60]. It is important to note that only components relevant to this thesis, specifically those pertaining to GNC system, were studied.

Perseverance

As the latest and most advanced Martian rover, Perseverance incorporates several improvements over its predecessors. It is equipped with two color stereo NavCams, separated by 42 cm, each providing a Field of View (FoV) of 96 ° × 73 ° at 0.33 mrad px^{-1}, capable of detecting a golf ball at a distance of 25 m [13, 61]. Additionally, it features six color stereo Hazard Avoidance Cameras (HazCams), with four positioned at the front and two at the rear, each providing a FoV of 136 ° × 102 ° at 0.46 mrad px^{-1} [13, 61]. These cameras are responsible for detecting rocks, trenches, and any potential hazards in the rover's path. The disposition of these cameras can be seen in Figure 4.5.

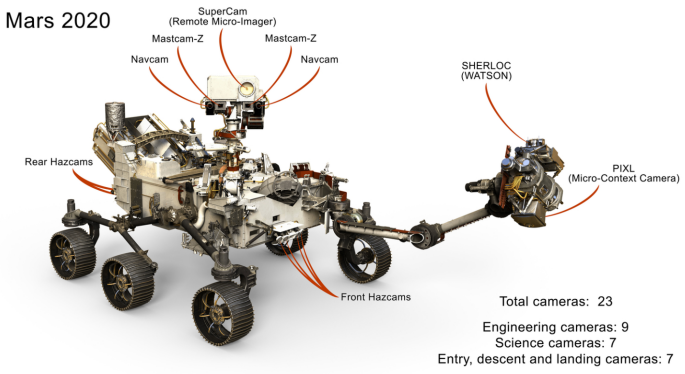


Figure 4.5: Location of Perseverance's cameras
(credit: NASA/JPL-Caltech)

The rover features a Data Storage Unit (DSU) powered by one of the Atom-based COMEX-IE38 modules from CompuLab, running Linux. The images from all of Perseverance's cameras are fed raw to the DSU via an ethernet link, where they are compressed by the COMEX-IE38 module and saved to a 480 GB Solid-State Drive (SSD) [61, 62]. An important aspect is that the vision algorithms operate on a dedicated computer, the Vision Compute Element (VCE), which includes a RAD750 processor and a Xilinx Virtex-5QV Field Programmable Gate Array (FPGA), whose computational power enhances its vision algorithm's performance compared to its predecessor [63]. The main computer, called Rover Compute Element (RCE) and the VCE communicate via both Military Standard 1553 (MIL-STD-1553) and bi-directional high-speed Low-Voltage Differential Signaling (LVDS) serial at 6 Mbit s^{-1} [63].

Perseverance is equipped with waypoint navigation, allowing it to receive a target location and calculate the optimal path to its destination. It can perform terrain mapping and maintain a 2.5-Dimensional (2.5D) elevation map of the terrain. Additionally, the rover is integrated with an IMU provided by Northrop Grumman [64].

Ingenuity

The Ingenuity helicopter functions as a Proof of Concept (PoC) designed to evaluate the feasibility of powered flight on Mars [14]. This distinction is essential when comparing it to other missions, as it represents a test rather than a fully operational project like a rover. The helicopter's sensors predominantly consist of Commercial Off-The-Shelf (COTS) products developed for the drone market. It is equipped with two cameras: a downward-looking 640 px × 480 px grayscale NavCam that utilizes an Omnivision OV7251 global-shutter sensor, providing images at a rate of 30 Hz and featuring a FoV of 166.5 ° ± 3 °. A Return-To-Earth (RTE) camera features a

rolling shutter with a resolution of 4208 px × 3120 px. This camera provides a FoV of 47 ° × 47 ° at 0.26 mrad px^{-1}. Additionally, the helicopter incorporates a 40 m ± 2.5 cm range Laser Rangefinder (LRF) at distances greater than 1 m [65], IMU, and an inclinometer used to calibrate the accelerometer prior to a flight [59, 66, 67].

The primary navigation computer is powered by a Snapdragon 801 processor, featuring an Adreno 330 GPU running at 578 MHz, equipped with 8 Texture Mapping Units (TMUs) and 32 pipelines [68]. It is connected via Universal Asynchronous Receiver/Transmitter (UART) to two redundant Texas Instruments Micro Controller Unit (MCU) TMS570LC43x, from the ARM Cortex-R5 family, that act as the Flight Control (FC) computers. As with its rover companion, NASA has opted for the use of an FPGA, operating at 500 Hz, for low-level control, power, and fault-tolerant considerations [59]. This architecture is illustrated in Figure 4.6.

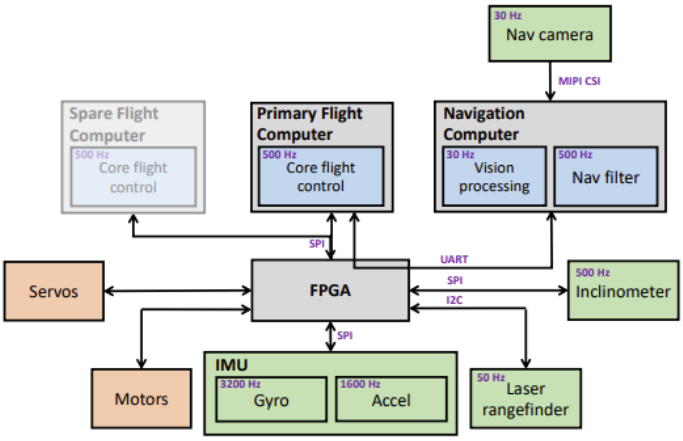


Figure 4.6: Ingenuity avionics architecture [66]

Ingenuity is capable of vision-inertial navigation; however, its path planning is human-based. As indicated by Grip *et al.* [67], a team on Earth establishes a sequence of waypoints on an orbital map. Additional parameters, such as rotor speed, must also be selected. In fact, Ben Pipenberg, AeroVironment's engineering lead on the Ingenuity project, stated in [69]:

> "There's no [navigation] decisions being made onboard [...] It's doing its own simple autonomy. But certainly no sophisticated mission planning or decision-making."

The operational time constraints necessitate that projected energy consumption be calculated based on defined parameters and validated against the available energy resources. The operational flexibility of Ingenuity is limited, as turn maneuvers and vertical or horizontal translations can only be performed sequentially. Additionally, the helicopter is restricted to flying in straight paths between waypoints, in which

it must come to a complete stop before proceeding. During flight, Ingenuity utilizes Perseverance as a ground station, transmitting data regarding its flight, including Ingenuity's estimated position, speed, and warnings emitted by the FC system.

Dragonfly

The dual-quadcopter (artist design shown in Figure 4.7), of over 400 kg, is set to explore a variety of locations on Titan. Since it is only scheduled to begin its long journey in 2028, the quantity of information available is very limited.

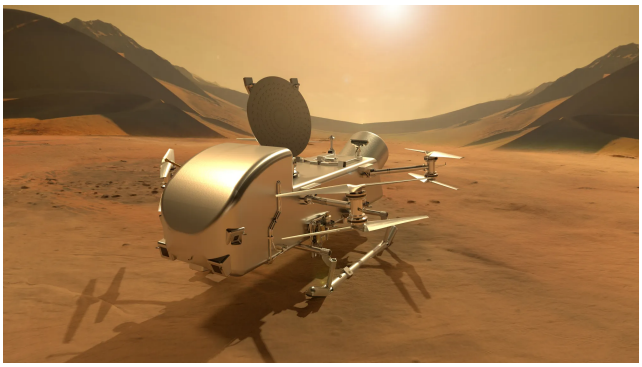


Figure 4.7: Dragonfly artist's impression
(credit: NASA/Johns Hopkins APL/Steve Gribben)

Dragonfly will feature various sensors operating at different frequencies, as summarized in Table 4.1.

Table 4.1: Dragonfly sensor specifications (adapted from [70, 71])

Sensor	**Ammount**	**Frequency** (Hz)
IMU	2	200
NavCam	1	1
Pressure Sensors	2	10
3D Flash LiDAR	1	5-10

The Ocellus LiDAR system was developed by NASA Goddard Space Flight Center to address the need for improved sensing capabilities in autonomous planetary missions, specifically for determining safe landing zones [72]. This LiDAR system can function as an altimeter at high altitudes and, at lower altitudes, perform terrain mapping and imaging [72]. NASA also explains that with this new system, Dragonfly will be able to autonomously generate 3D point clouds that are converted into elevation maps and images with centimeter scale. For a detailed explanation of the operating modes, refer to Table 4.2. The +X and +Y Line of Sight (LoS) represent +7.5 ° forward of boresight and +7.5 ° to the right of boresight, respectively.

Table 4.2: Ocellus LiDAR modes (adapted from [70, 72])

	Pyramid Mode	**Altimetry Mode**
Rate (Hz)	5	10
LoS Measurements	Boresight, +Y, -Y, +X, -X	Boresight
Mode Functionality	Terrain mapping and imaging	Range-finding
Altitude (m)	15 - 200	200 - 2000

By the means of a more thorough investigation, additional information regarding Dragonfly's sensors was uncovered. NASA's official Reddit page responded to several fan inquiries about the project [73] and stated that "the flight sensors include inertial measurement units, cameras, radar, lidar, and an ultrasonic altimeter," providing deeper insight into the drone's capabilities. This is also supported by McGee *et al.* [71].

Limited additional information is available. However, Schilling *et al.* [70] clarify that, similar to Perseverance, the selected processor is the radiation-hardened BAE RAD750.

Summary

Finally, a comparison of the three robots can be made. It is important to note that, due to limited computational power (see Table 4.3) and the continuous state growth from observing new landmarks, none of the featured robotic systems have implemented SLAM—a computationally intensive technique—which could result in a significant impact on their capabilities. However, Shaukat *et al.* [74] have proposed a future rover GNC architecture that incorporates SLAM. This approach relies on a coarse map of the terrain obtained from High Resolution Imaging Science Experiment (HiRISE) orbital imagery, and for this reason, it will not be considered for this project. Table 4.4[1] highlights the most important comparison aspects considered in the preparation of this thesis.

Table 4.3: Comparison between main computers[2]

	BAE RAD750 [75, 13]	**Snapdragon TM 801 [76, 59]**
Max frequency	200 MHz	2.5 GHz
Cores	1	4
Flash memory	2 GB	32 GB
RAM/DRAM	256 MB	2 GB
EEPROM	256 kB	-
L2 Cache	1 MB	2 MB

[1]Table acronyms: Radio Detection And Ranging (RADAR).

[2]Table acronyms: Random Access Memory (RAM), Dynamic Random Access Memory (DRAM), Electrically Erasable Programmable Read-Only Memory (EEPROM).

Table 4.4: Comparison between NASA robotic system [77, 78]

	Perseverance	**Ingenuity**	**Dragonfly**
Main Computer	BAE RAD750 [13, 64]	Snapdragon TM 801 [66, 59]	BAE RAD750 [70]
Cameras	2x color stereo NavCams, 6x color HazCams [13]	0.5 Mpx Grayscale NavCam @ 30 Hz 13 Mpx RTE [66]	NavCam @ 1 Hz [70]
Sensors	LN-200S 6-axis IMU [79]	2x BMI-160 6-axis IMUs @ 1600/3200 Hz LiDAR-Lite-V3 LRF @ 50 Hz SCA100T-D02 Inclinometer @ 500 Hz [66, 59]	2x 6-axis IMUs @ 200 Hz Flash LiDAR @ 5-10 Hz 2x Pressure Sensors @ 10 Hz Ultrasonic altimeter RADAR [70, 73]
Autonomy Level	Semi-autonomous Autonomous [63]	Autonomous [59]	Autonomous [70]
Operation Duration	-	167 s [80]	30-60 min [73]
Operating Systems	VxWorks Linux [61]	Linux [66, 59]	Undisclosed
Approximate Max Speed	4 $\mathrm{cm\,s^{-1}}$ [13, 63]	10 $\mathrm{m\,s^{-1}}$ [66]	10 $\mathrm{m\,s^{-1}}$ [78]
Max Altitude	-	24 m	4000 m [78]

With this in mind, one can lay out some more constraints and requirements:

- **C-1: The application of SLAM is limited**;
  Reasoning: As mentioned before, SLAM has been studied to integrate rovers in the future, but its implementation on the drone must be limited due to the constraints of onboard computational power. Fully leveraging SLAM would exhaust the system's resources and potentially lead to disastrous damages to the platform. However, ongoing research into optimizing it for resource-constrained scenarios, like those indicated earlier, suggest that, with further advancements, partial or scaled-down implementations may be feasible. Thus, while SLAM is not entirely excluded, its use must be carefully managed to avoid overburdening the drone's processing capabilities.

- **C-2: Sensor types are limited to those that have either been previously utilized or show promise to be used in the future**;
  Reasoning: This constraint ensures fidelity to the proven and space-qualified technologies, aligning with both the historical successes, future ambitions of space missions, all the while not forgetting technological advancements.

- **S-2: The system is assumed to be equipped with an SSD, allowing for greater image capture**;
  Reasoning: By equipping the system with an SSD, it reduces concerns related to storage constraints.

- **S-3: Vision algorithms are assumed to run on a dedicated computer**;
  Reasoning: Allocating a dedicated computer for running vision algorithms is essential to ensure optimal performance and not further clutter the main computer.

- **S-4: Sensor frequencies will fall within the ranges of the sensors previously studied**;
  Reasoning: By adhering to the frequency ranges of previously validated sensors, unrealistic or excessively high values that may compromise the system's operational integrity and feasibility, are avoided.

- **S-5: The system is expected to incorporate a down-facing monocular camera**;
  Reasoning: Navigating through Valles Marineris requires comprehensive environmental awareness. The down facing camera will provide critical navigation capabilities.

- **S-6: The sensors employed will possess characteristics consistent with those analyzed in Table 4.4**;

Reasoning: The aim is to enhance the system's fidelity to real-world applications by utilizing sensors with characteristics closely resembling those outlined in Table 4.4.

- **S-7: The drone will not exceed the speed of the studied aerial platforms**;
  Reasoning: NASA has chosen to cap its drones' velocity at 10 $\mathrm{m\,s^{-1}}$, which likely reflects the speed engineers consider safe.

Part II

Core Systems Design and Development

Chapter 5

System Design

Throughout this chapter, the complete system design will be presented, encompassing the sensors and their specifications. Additionally, this chapter provides an overview of the GNC algorithms developed for the project.

A simplified drone concept was designed to serve as the foundation for this project. Again, it should be emphasized that the goal of this work was not to design a complex control system, which would require a highly detailed and sophisticated drone model. The design adopted in this work serves as a basic foundation upon which the guidance and navigation modules were built. Therefore, the simplified control system will not be detailed in this dissertation, as it was developed solely for testing purposes.

For simplicity and clarity, the system has been named Fenix and will henceforth be referred to as such. Table 5.1 provides an overview of the sensors used and their characteristics. This selection was strongly influenced by previous and upcoming NASA missions, as discussed in Section 4.3, which explains the similarity between Fenix's sensor suite and those employed by NASA.

To meet the S-1 requirement (refer to Chapter 4 as needed), it was essential to enable all types of compiler warnings for all developed code. The compiler settings used are detailed in Appendix A.1. The code was compiled without generating any warnings.

[1]Noise values reported without units (*e.g.*, IMU and Monocular Camera) come from Gazebo, which does not specify units explicitly; these values represent standard deviations of additive Gaussian noise.

Table 5.1: Fenix's sensors and characteristics[1]

	Characteristics	Detail
Monocular Camera	Resolution	640 px × 480 px
	FoV	62.7 °
	Noise	0.0005
	Frequency	1 Hz
	Color	Monochromatic
IMU	Noise	0.005
	Frequency	200 Hz
LRF	Noise	2 cm
	Frequency	5 Hz
	Accuracy	2 cm
	Range	2000 m

Figure 5.1 illustrates the simplified architecture of the interconnected system. As clearly shown, the structure adheres to the same interconnection principles presented in the general GNC architecture discussed in Chapter 2. Throughout the following sections, each of these modules will be thoroughly explained.

The red circles around the Guidance modules in Figure 5.1 highlight blocking states—conditions under which these components may trigger a mission abort, before it even starts.

5.1 Guidance System

Achieving efficient path planning required careful evaluation of numerous factors. However, to ensure the effectiveness of this module, certain assumptions also had to be made. This is because, for the module to operate efficiently, it is essential to clearly define its objectives and understand the nature of the input data it will receive.

Given the terrain in the Valles Marineris region, it was assumed that the drone — flying no higher than 2000 m — would encounter both open areas free of obstacles and regions with obstructions along its trajectory. For this reason, a simple straight-line trajectory between the starting point and the final destination was deemed unsuitable for certain scenarios.

Moreover, it was assumed that the drone could make several intermediate stops along its trajectory. The various operating modes supported by the drone are outlined in Table 5.2.

To generate a trajectory, a mission file must be created by following the sequence specified below:

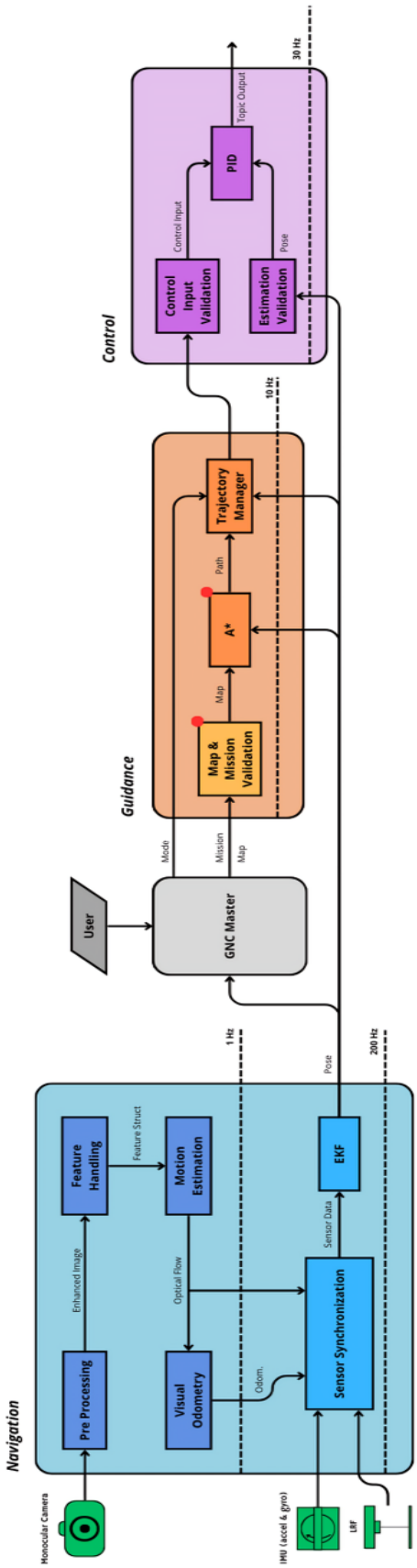


Figure 5.1: System architecture

1. Final position in the x-axis;
2. Final position in the y-axis;
3. Final position in the z-axis;
4. Total number of stops;
5. n-th stop position in the x-axis;
6. n-th stop position in the y-axis;
7. n-th stop position in the z-axis;
8. Final mode of operation;
9. Mode of operation at the n-th stop;
10. Traverse speed;
11. Curiosity speed;

Table 5.2: Fenix's operating modes

Mode	**Description**
Landing	Fenix initiates the landing sequence
Taking-off	Fenix begins the take-off procedure
Shutdown	Fenix powers down all operations
Hover	Fenix maintains a stable position in the air
Curiosity	Fenix explores the surrounding environment
Traverse	Fenix moves between designated points (*e.g.*, A to B)

Before generating the trajectory, the mission file undergoes a verification process to ensure its content and structural integrity. For a detailed explanation, refer to Appendix B.

The A* algorithm [81] was selected as the path planning approach. As demonstrated in [82, 83], which present comparative analysis of popular path planning algorithms, A* achieves the highest success rate while also producing one of the lowest cell counts—*i.e.*, the shortest paths—among the evaluated methods. Additionally, it offers relatively fast computation times.

The choice of A* was influenced by several factors. Primarily, the environment was assumed to be static, which ruled out the need for dynamic replanning algorithms such as D*. Rapidly Exploring Random Tree (RRT) was also considered but quickly discarded. This algorithm does not guarantee the shortest path and relies on random sampling, which can lead to increased computational time. Moreover, because it selects the first valid trajectory it finds, the resulting path is often suboptimal and, in some cases, significantly inefficient or poorly suited for navigation.

There are optimizations to this algorithm, such as those proposed in [84], which are referenced in Chapter 7 for potential future implementation and comparison. However, A* was deemed more suitable for the current task due to its determinism, efficiency, and path optimality in static environments.

5.1.1 The Grid and Map

Given the nature of the drone's operational environment, a 2D grid-based method was chosen for path planning. For the drone to function effectively, a map is required—this can either be generated onboard or provided beforehand, based on what mission planners on "Earth" deem most suitable. The drone is capable of both constructing its own map autonomously and utilizing a preloaded one. Naturally, the map must correspond to the region Fenix is set to explore. When a preloaded map is used, the drone does not validate its accuracy against the actual environment—it operates under the assumption that the provided map is correct.

When the map is constructed by the drone, it utilizes the coordinates obtained during mission file verification to determine the appropriate map dimensions. Specifically, it identifies the maximum and minimum values along the x and y axes in order to calculate the spatial extent of the area to be mapped. Additionally, a border is added around the entire map to provide leeway and ensure safe margins during navigation.

The ability to control the map resolution has been introduced. Naturally, if the map's dimensions in pixels were equivalent to the real-world dimensions in meters, each pixel would correspond to 1 m. The resolution parameter allows the user to define how many pixels represent a single meter. While this resolution could theoretically be adjusted dynamically by Fenix, it is currently a user-defined setting and not modifiable by the drone during operation. In future prototypes, enabling dynamic resolution adjustment could serve as an effective optimization. However, this would require careful evaluation of the onboard computer's processing capabilities, as higher resolutions result in larger maps, increased computational load, and greater memory usage. If the map resolution is set to 1 $\mathrm{px\,m^{-1}}$, then each pixel in the grid represents a 1 $\mathrm{m^2}$ area in the real world. As a result, the position of an object or the robot within a pixel cannot be determined more precisely than within that 1 $\mathrm{m^2}$. The center of a pixel is typically used to represent its coordinates. Therefore, any position reported as belonging to a given pixel could actually lie anywhere within its boundaries. This introduces an inherent quantization error of up to half a pixel in any direction, which translates to a maximum possible error of ± 0.5 m along both the x and y axes. While higher resolutions (*e.g.*, 2 $\mathrm{px\,m^{-1}}$) can reduce this quantization error, they require significantly more memory and computational resources.

Subsequently, it was necessary to assign weights to the cells of the grid. This step is crucial for generating an optimal trajectory, as the performance of the A* algorithm is highly dependent on the quality of these weights. While A* will always find the shortest path given the weights, suboptimal or inaccurate weight distribution can result in poor trajectory planning, making this step critical to the overall path quality. These weights were calculated as the Euclidean distance between a given cell and the drone's destination. The weights were then rounded to integers to allow for visual representation using `rqt`, as shown in Figure 5.2.

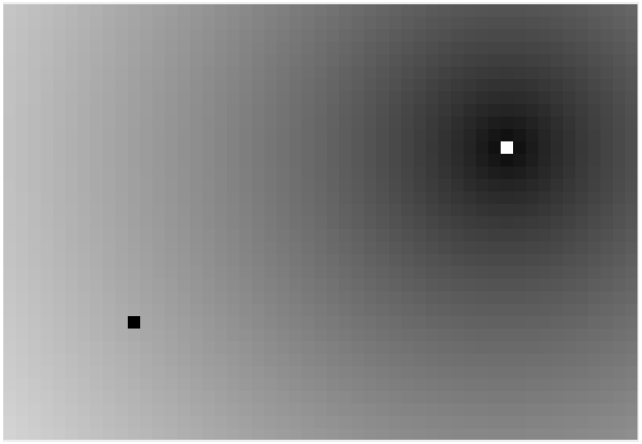


Figure 5.2: Generated map with final position at $(x, y) = (15, 7)$ m, using double resolution, *i.e.*, 2 px m^{-1}

In this visualization, darker cells correspond to lower weights, indicating areas with less cost for the drone to navigate. The starting position corresponds to the black pixel, while the white pixel represents the destination. It should be noted that using the rounded integer values directly would be a critical mistake, potentially leading to poor paths. This is because adjacent cells may end up with the same integer weight, even though they should represent distinct values in reality, which would significantly affect the quality of the trajectory.

Manhattan distance was also tested and produced functionally acceptable results[2]. However, it placed disproportionately high costs on diagonal movements, causing the system to favor straight-line trajectories and only turn when strictly necessary. A comparison between these two methods is presented in Appendix C, highlighting the differences in path behavior and overall performance.

5.1.2 A* Algorithm

The algorithm itself is conceptually simple and straightforward. It relies on two primary data structures: a list of nodes that have been discovered but not yet

[2]"Acceptable" meaning that, while the resulting paths are not the shortest possible, this metric still offers several practical advantages.

visited, referred to as the *TO_SEE_LIST*, and a list of nodes that have already been explored, referred to as the *VISITED_LIST*. Additionally, the algorithm relies on a cost function $f(n)$ defined as

$$f(n) = g(n) + h(n), \tag{5.1}$$

where $g(n)$ denotes the cumulative cost from the starting node to the current node n, and $h(n)$ is the heuristic estimate of the remaining cost to the goal. As previously mentioned, $h(n)$ can be computed using either the Euclidean or Manhattan distance[3]. The A* algorithm is illustrated in Algorithm 5.1.

```
1  Initialize current_node to the starting pose.
2  while current_node is not the goal and TO_SEE_LIST is not empty do
3      for all neighbors of current_node do
4          if neighbor ∈ VISITED_LIST then
5              continue
6          else if neighbor ∈ TO_SEE_LIST then
7              Compute new cost g(n) from start via current_node
8              if new g(n) < stored g(n) then
9                  Update g(n)
10                 Set parent of neighbor to current_node
11             end if
12         else
13             Add neighbor to TO_SEE_LIST
14             Set parent of neighbor to current_node
15             Compute f(n) = g(n) + h(n)
16         end if
17     end for
18     Initialize best_node ← first element of TO_SEE_LIST
19     for all node in TO_SEE_LIST do
20         if f(node) < f(best_node) or
            (f(node) = f(best_node) and h(node) < h(best_node)) then
21             best_node ← node
22         end if
23     end for
24     Move best_node to VISITED_LIST
25     Remove best_node from TO_SEE_LIST
26     Set current_node ← best_node
27 end while
```

Algorithm 5.1: A* path planning algorithm

It is crucial to ensure that the heuristic function $h(n)$ is admissible—that is, it must never overestimate the true cost to reach the goal. This guarantees the optimality of the A* algorithm. In other words, $h(n)$ must be either exact or optimistic.

[3]Manhattan distance is not a good metric since it overestimates the cost of diagonal movements. Later in this Section, this will be explained.

If this condition is violated, the algorithm may fail to explore potentially optimal paths, leading to suboptimal solutions. Fortunately, the Euclidean distance satisfies this condition and is thus a valid heuristic for A* in this context.

Figure 5.3 illustrates two scenarios: one where the heuristic is accurate and one where it is optimistic. Consider the cell at position $(2, 2)$, which is diagonally right from the green starting position. In both cases, the Euclidean and Manhattan distances remain unchanged, as the relationship between this cell and the goal does not alter. However, in the second scenario, where the heuristic is optimistic, it underestimates the true cost. Since the cell at $(3, 3)$ is obstructed, the path must be adjusted to circumvent the obstacle. As a result, the actual cost to reach the goal is greater than the optimistic estimate.

Figure 5.3: Two 5×5 grids. Green represents the start position and red represents the goal. The second grid has obstacles represented in black.

Once the goal is reached, the final step is to backtrack from the goal to the starting position. This process is facilitated by the parent ID variable, which records the predecessor of each node, allowing the system to trace the path back to the origin. The result of the implementation of this algorithm is shown in Figure 5.4.

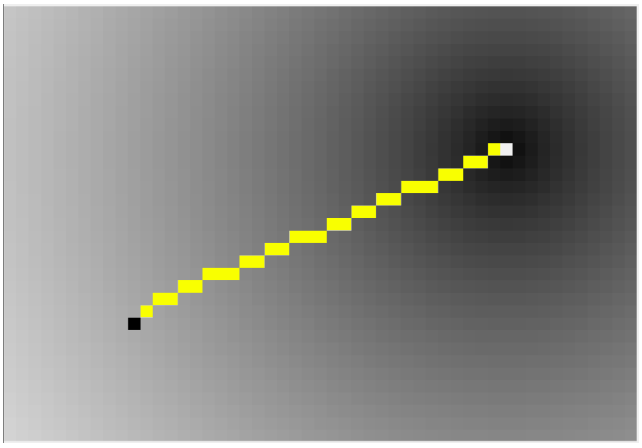


Figure 5.4: Generated path given the map in Figure 5.2 with Euclidean distance heuristics

Even though the algorithm is straightforward, it is crucial to ensure proper data updates, as any inconsistencies can lead to erroneous paths and cause the system to crash due to a core dump. The conditional statement in Line 6 of Algorithm 5.1 is especially critical, as it plays a vital role in distinguishing between sub-optimal and

optimal paths. The objective of this step is to determine whether a more efficient route exists to reach a given node, compared to the previously considered path. Another critical consideration when generating the map—whether constructed by the drone itself or pre-loaded—is the need to inflate obstacles beyond their actual dimensions as a safety measure. This precaution stems from the tendency of the planned trajectory to "hug" obstacles closely, which could result in hazardous flight paths if obstacles are represented with their true size. Naturally, the lower the map resolution, the greater the degree of inflation required to maintain a safe distance from potential collisions.

Limitations of Manhattan Distance as an Heuristic in Path Planning

Manhattan distance, in theory, overestimates diagonal distances. As a result, the heuristic $h(n)$ would not be optimal, as it only accounts for cardinal (axis-aligned) distances.

However, it was found through intensive testing that, if $g(n)$ is also Manhattan distance-based, then the minimum possible number of steps toward the goal will also be achieved.

It is entirely possible to determine *a priori* the minimum number of steps required to move from one coordinate to another by using the following equation:

$$s = \max\left(|x_2 - x_1|, |y_2 - y_1|\right), \tag{5.2}$$

where (x_1, y_1) and (x_2, y_2) are the initial and final coordinates, respectively, and s is the minimum number of steps, assuming 8-connected grid movement (*i.e.*, allowing diagonal movement).

It is true, however, that paths generated using the Manhattan distance heuristic result in longer trajectories compared to the actual shortest path, due to the heuristic's inability to account correctly for diagonal movement, even if the step count is the same. Nevertheless, as will be shown later, Manhattan distance offers several advantages—most notably in terms of computational efficiency—which justified its inclusion.

5.2 Navigation System

In this Section, the navigation system will be introduced. This module comprises the camera as well as the state estimation.

5.2.1 Monocular Odometry

While addressing the challenges presented by this project, the first problem tackled was navigation, with a specific focus on computer vision considerations. Among its various functionalities, the most prominent are:

- Image pre-processing;
- Feature detection, extraction, and matching;
- Outlier rejection;
- Monocular navigation.

As mentioned previously, OpenCV was not utilized in this project. Excluding standard C++ libraries, the only external library employed was Eigen, which was used for mathematical operations. Consequently, all vision algorithms were implemented raw from scratch. This approach not only provided a deeper and more meaningful learning experience but also allowed for the customization of the algorithms to suit the specific requirements of the scenario. Out of curiosity, the computer vision module is composed of about 3600 lines of code.

Over the course of the next sections, a detailed explanation of all these points will be made clear.

Image Pre-Processing

Before implementing any navigation-related algorithm, it was essential to pre-process the available images, preparing them to be better suited for the subsequent stages of the code.

The pipeline for the camera involves stabilizing the image to facilitate feature detection and subsequent navigation. Figure 5.5 illustrates the pipeline for this module. The primary objective of this pipeline was to ensure robustness and reliability. The system was designed with sufficient alternatives and fail-safes to minimize the risk of failure and potential hazards if this component did not succeed. Although the failure of any of these algorithms is highly unlikely, potential issues could arise. For example, each algorithm first checks whether the image is corrupted by verifying its dimensions. This is done by comparing the size of the image array with the maximum possible value based on its dimensions, ensuring coherence between the two. Additionally, both the Gaussian and Average filters verify whether the kernels' operations are within the dimensions of the images. It is also ensured that kernel operations remain within image bounds. In cases where a method fails due to image corruption rather than an algorithmic issue, a previously stored placeholder image—identical to the initial image—is used as a fallback.

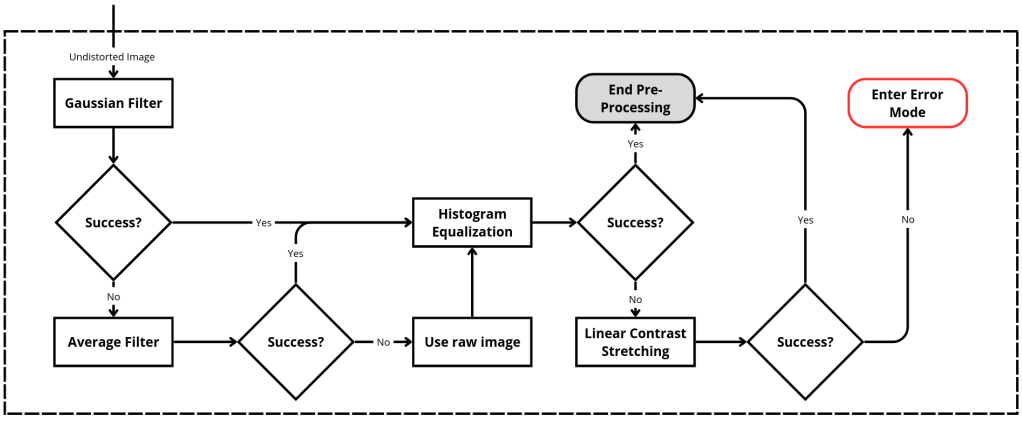


Figure 5.5: Monocular pre-processing pipeline

In cases where it is not possible to reliably execute the monocular odometry algorithms, this module withholds odometry data and does not transmit it to the estimation unit.

Table 5.3 presents all the algorithms that were considered and also tested, along with their status—whether included or excluded—and the reasoning behind these decisions.

Table 5.3: Pre-processing algorithms tested

Algorithms	**Status**	**Reasoning**
Histogram Equalization	Included	Great contrast increase
Linear Contrast Stretching	Included	Good contrast increase
Gaussian Filter	Included	High quality denoise
Average Filter	Included	Simple denoise alternative
Zero-Mean Equalization	Removed	Image becomes too dark to detect features reliably
Gamma Correction	Removed	Feature sharpness decreases

To demonstrate the results of these operations, images captured by the monocular camera, both before and after processing, were stored. The original raw image is presented in Figure 5.6. A few algorithms were considered for smoothing the image and reducing unwanted noise, including: Gaussian filter, Average filter, and Median filter.

The Gaussian filter was the first option selected, as it is a cornerstone in robotics and computer vision. This method involves weighting the pixels within an $N \times N$ kernel using a Gaussian distribution, as its name suggests [85]. Consequently, the current central pixel contributes more significantly to the final intensity value of that pixel location. The algorithm employed for this filter is described in Algorithm 5.2.

Figure 5.6: Raw monocular image

An interesting detail taken into account was that, as evident in Line 11, the kernel values were scaled up to integers to minimize floating-point operations. Alternatively, instead of using a static kernel with predefined values, it is entirely valid to generate the Gaussian kernel dynamically using the equation:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad x,y \in \left\{-\frac{N}{2},\ldots,\frac{N}{2}\right\}, \tag{5.3}$$

where x, y are the coordinates within the $N \times N$ kernel, and σ corresponds to the standard deviation. This parameter controls the strength of the smoothing effect.

However, the dynamic approach posed several challenges that ultimately led to the decision to integrate the static version. Firstly, it involves complex mathematical operations, including exponentials, squares, and divisions that may result in floating-point numbers, making it challenging for limited processors. Secondly, this method introduces additional computational overhead, which was deemed unnecessary given its marginal advantages over the static approach.

In contrast, the Average filter also utilizes a $N \times N$ kernel but, instead of weighting the pixels, computes the average intensity value of all pixels within the window [85]. While this approach requires less computational time, it comes at the expense of being less adaptive and precise. Its implementation can be seen in Algorithm 5.3.

The Median filter was also considered as an alternative to the Gaussian filter. This method stores the intensity values of the pixels within a $N \times N$ kernel, orders them ascendingly, and then retrieves the median value [85]. This, however, comes at the computational cost of ordering an array, which was, then again, considered unnecessary given it's a backup method and doesn't provide clear big advantages

```
1  function GaussianFilter(undistorted_image)
2      Validate undistorted_image
3      processedImage = undistorted_image
4      for each pixel coordinate (i, j) in undistorted_image where 1 ≤ i <
       height − 1 and 1 ≤ j < width − 1 do
5          pixelValue ← ApplyKernel(undistorted_image, i − 1, j − 1)
6          Store pixelValue at processedImage(i, j)
7      end for
8      undistorted_image ← processedImage
9  end function
10 function ApplyKernel(image, i, j)
11     k ← [16, 32, 16, 32, 64, 32, 16, 32, 16]
12     result ← 0
13     N ← 3
14     for each coordinate (x, y) in the N × N kernel do
15         pixelValue ← image[ (y + i) · IMAGE_WIDTH + (x + j) ]
16         result ← result + k[y · N + x] · pixelValue
17     end for
18     return result / 256
19 end function
```

Algorithm 5.2: Gaussian filter employed algorithm

over the others.

The comparison between these two filters is shown in Figure 5.7. Both filters produced very similar results, with the Gaussian filter offering a slightly sharper image. However, this difference may be difficult to discern in the provided images.

It is entirely possible to control the smoothing effect of these algorithms by adjusting the kernel size N. Larger kernel sizes result in stronger smoothing, which can potentially over-correct and lead to noticeably blurry images. It is worth mentioning that, regardless of the kernel size, some degree of blurriness is always introduced as a natural consequence of the smoothing process. Since these methods take surrounding pixels into account to compute the final pixel value, the transitions between these become less sharp, resulting in smoother and more diffused images.

The second step in the pre-processing unit involved enhancing the contrast of the image, which, in turn, increased the number of detected features. Additionally, as previously explained, the filtering stage causes images to blur, reducing the sharpness of features. By implementing a method to enhance contrast, it was possible to mitigate this effect while simultaneously improving the sharpness of the image. Histogram equalization was the method that provided the best results, as the image was changed from dominant in monotonous gray to a wider spectrum of values. An example of this algorithm is going to be provided. Consider the image illustrated in Figure 5.8, with pixel intensities ranging from 0 to 9:

```
1  function AverageFilter(undistorted_image)
2      Validate undistorted_image
3      processedImage = undistorted_image
4      for each coordinate (i, j) in undistorted_image where 1 ≤ i < height − 1
       and 1 ≤ j < width − 1 do
5          pixelValue ← AverageBlock(undistorted_image, i − 1, j − 1,
           MONOCULAR_WIDTH)
6          Store pixelValue at processedImage(i, j)
7      end for
8      undistorted_image ← processedImage
9  end function
10 function AverageBlock(image, i, j, width)
11     result ← 0
12     N ← 3
13     for each coordinate (x, y) in the N × N kernel do
14         pixelValue ← image[ (y + i) · width + (x + j) ]
15         result ← result + pixelValue
16     end for
17     return result/(N×N)
18 end function
```

Algorithm 5.3: Average filter employed algorithm

9	1	8
3	3	6
9	0	2

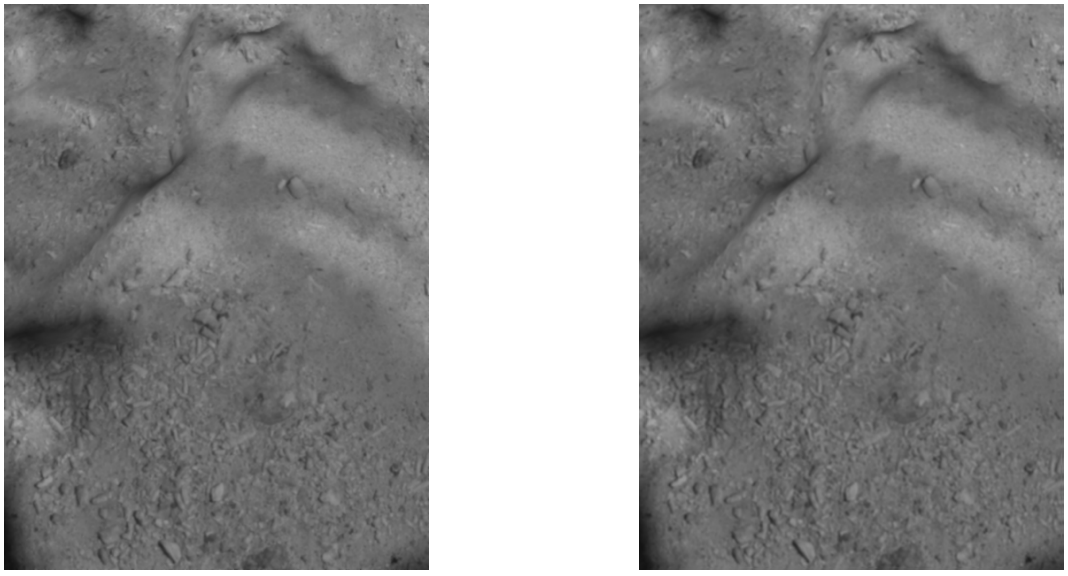
Figure 5.8: Example image with intensity values ranging from 0 to 9

The first step involves creating a histogram of the image, where the x values represent all possible pixel intensities, and the y values correspond to the number of occurrences.

After this, the cumulative sum of the histogram must be computed. The result represents, for each pixel intensity value, the cumulative number of occurrences up to and including that intensity, instead of the count for that specific intensity alone. Subsequently, the following equation is applied:

$$cdf(i) = \frac{cdf(i-1) \cdot mpv}{np}, \tag{5.4}$$

where $cdf(i)$ is the cumulative distribution function at intensity i, mpv represents the maximum possible value for the pixel intensity, and np is the total number of pixels. Naturally, $cdf(0) = histogram(0)$ since there is no term $i - 1$. The final step involves reassigning the pixel values using $cdf(p)$, where p represents the



(a) Monocular image with Gaussian filter

(b) Monocular image with Average filter

Figure 5.7: Filtering techniques employed

pixel intensity of the corresponding pixel. The comparison between the original and equalized histogram is shown in Figure 5.9.

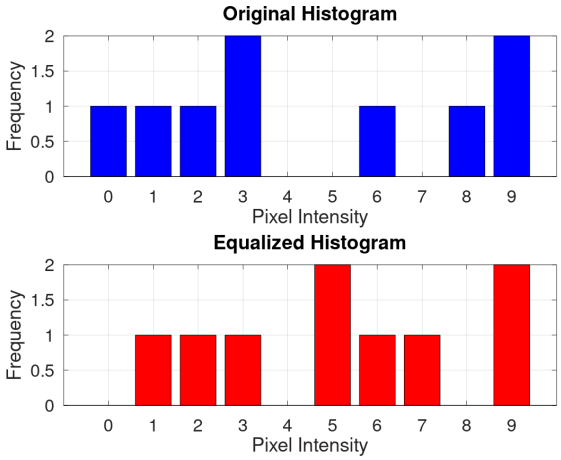


Figure 5.9: Comparison between original and equalized histograms

While the result might not be particularly elucidative given such a small image, it becomes much clearer with Fenix's monocular camera, as can be seen in Figure 5.10. Very often, the results of the equalized histogram are represented on a scale of $[0, 1]$; however, it was scaled up in this case to depict the exact pixel intensities.

Notice how, in the original image, the vast majority of pixel intensities are concentrated within a narrow range between 75 and 150. This results in a raw image with a somewhat monotonous gray tone, lacking distinction. However, after the

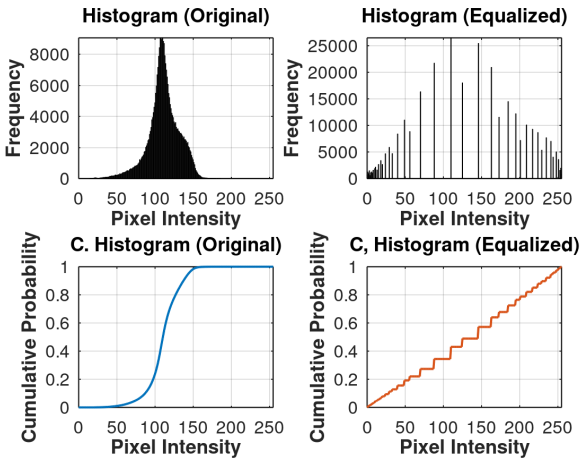


Figure 5.10: Plots of the histograms of both the original and equalized monocular image

process of equalization, this concentration is redistributed across the entire spectrum of intensities. Consequently, the cumulative histogram grows linearly rather than abruptly. Figure 5.11 demonstrates the resulting image after applying this algorithm. This image was compared with Octave's implementation and found to be similar, without any discernible differences.

Figure 5.11: Fenix's monocular image after histogram equalization

To address the potential—though unlikely—failure of this method, such as when the cumulative sum of the last histogram value is equal to zero, for instance, linear contrast stretching was also implemented. This technique involves identifying the minimum and maximum pixel values within an image [86], and then adjusting the pixel intensities of each pixel according to the following equation:

$$S = \frac{(R - \min(R)) \cdot 255}{\max(R) - \min(R)}$$

Here, S represents the output pixel value after the contrast stretching transformation, and R denotes the original pixel value. The term $\min(R)$ indicates the minimum intensity value in the original image, while $\max(R)$ represents the maximum intensity value. Figure 5.12 illustrates the result of this operation. However, one significant limitation of this method arises when the image's minimum and maximum values coincide with the spectrum's minimum and maximum. In such cases, where $\min(R) = 0$ and $\max(R) = 255$, the equation simplifies as follows:

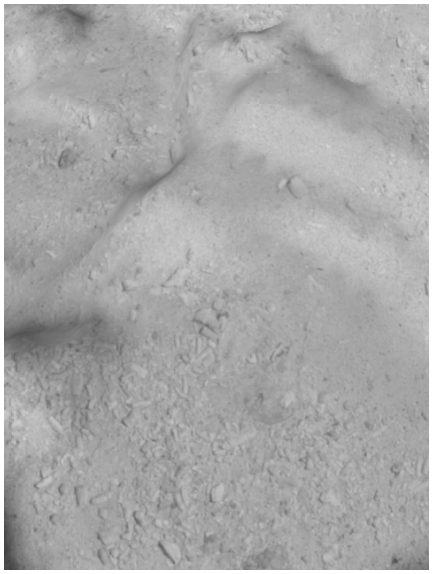
$$S = \frac{(R - \min(R)) \cdot 255}{\max(R) - \min(R)} = \frac{R \cdot 255}{255} = R$$

Thus, the output pixel values remain unchanged, *i.e.*, $S = R$.

Figure 5.12: Fenix's monocular image after linear contrast stretching

Lastly, zero-mean equalization and gamma correction were both tested in integration with the remainder of the code but yielded poor performance. Zero-mean equalization operates by computing the average intensity of all pixels within an image and subtracting this mean from all pixel values. As a result, the pixel intensities are reduced compared to the original image, leading to a darker appearance, as clearly illustrated in Figure 5.13. This algorithm was tested both before and after histogram equalization, but neither approach produced successful results.

(a) Monocular image with zero-mean equalization



(b) Monocular image with gamma correction

Figure 5.13: Filtering techniques not employed

On the other hand, gamma correction led to excessively bright images, with the effect intensifying as the gamma value increased. This caused a reduction in sharpness and significantly degraded the overall detectability of features.

Next, the pipeline for the monocular odometry, illustrated in Figure 5.14, will be described, starting from the image processing stage onward.

Feature Detection and Extraction

Feature detection and feature extraction are sometimes confused and used interchangeably, though they are distinct processes. Feature detection refers to identifying interesting and distinguishable pixels or neighborhoods within an image, ultimately extracting its coordinates. In contrast, feature extraction involves describing the regions surrounding a detected pixel, effectively creating a unique "footprint" that allows the feature to be identifiable [87]. Table 5.4 provides an overview of the most popular feature detectors and descriptors.

Table 5.4: Feature detection and descriptor algorithms

Method	**Detector**	**Descriptor**
ORB	Yes	Yes
FAST	Yes	No
BRIEF	No	Yes
SURF	Yes	Yes
SIFT	Yes	Yes

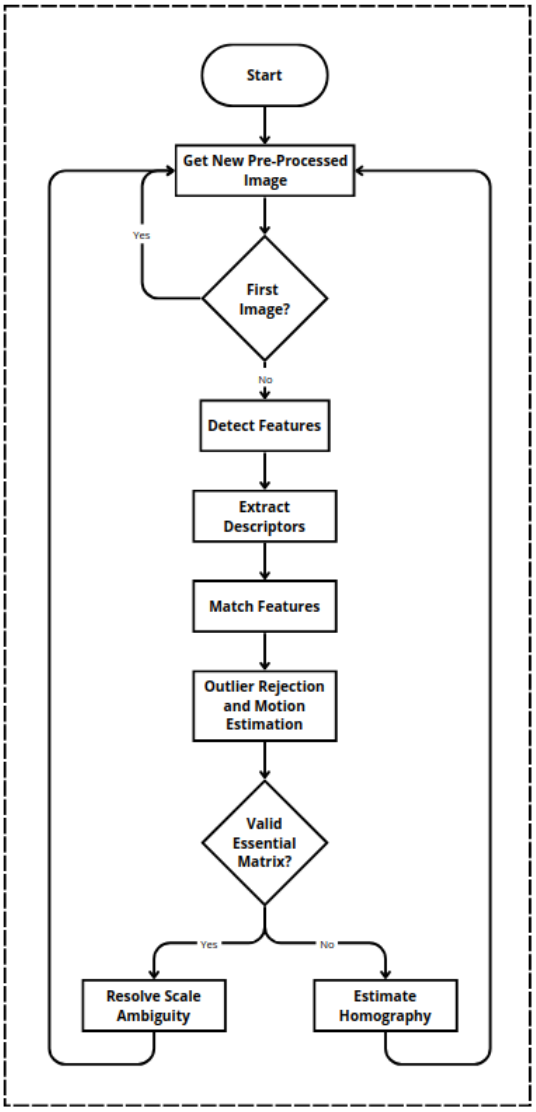


Figure 5.14: Pipeline for monocular odometry

Among the methods analyzed, ORB, Features from Accelerated Segment Test (FAST), and Binary Robust Independent Elementary Features (BRIEF) were selected for inclusion in this project. Conversely, Speeded-Up Robust Features (SURF) and Scale-Invariant Feature Transform (SIFT) were excluded due to their higher computational cost and suboptimal performance for real-time applications, as the study by Tareen *et al.* [88] suggests.

The project incorporates a method based on comparing descriptors and calculating their Hamming distance for tracking features across frames, which will be explored in the next sections. Optical flow methods, such as the Lucas-Kanade method, were also considered, as it was included in Ingenuity [66]. However, Fenix and Ingenuity are two vastly different systems, and the assumptions that work for

one do not necessarily translate easily to the other. For instance, the Lucas-Kanade method assumes small pixel displacements between consecutive frames [89], which is feasible for a drone like Ingenuity operating at 30 Hz. However, Fenix's slower frame rate makes this assumption unreliable, even with a slower velocity compared to Ingenuity. Another issue is the assumption of brightness consistency [89]. While this assumption may hold if the frame rate is very high, it is less reliable in other contexts. For instance, Ingenuity primarily flew in a desert region, where lighting conditions are relatively consistent. On the other hand, Fenix is designed to navigate canyons, where the walls, valleys, and natural geography can cast shadows and create varying lighting conditions between frames. Additionally, histogram equalization can significantly alter the pixel intensities of different areas in the image. Combined with the slower frame rate, this can cause variations in brightness between frames, even if not drastically, making the assumption of brightness consistency more problematic. There are potential workarounds, primarily using pyramid-based methods, known as coarse-to-fine optical flow. In this approach, the algorithm is applied iteratively on progressively higher resolution images [90]. In lower resolution images, the motion is inherently smaller due to fewer pixels, making it possible to apply the Lucas-Kanade method. An estimate is computed at each level and passed to the next, until the original resolution is reached [90]. However, despite these methods, descriptor-based approaches were considered more reliable and robust for the task at hand.

The ORB algorithm operates by optimizing and integrating the functionalities of FAST and BRIEF. To provide a comprehensive understanding of the system, each of these algorithms will be explained in detail. Furthermore, the optimizations implemented specifically for this project will also be discussed.

As the acronym suggests, FAST [91] is well-suited for real-time systems where rapid image processing is essential. The algorithm achieves this efficiency by utilizing an initial condition that must be satisfied in order for further processing to occur. Additionally, unlike other feature detectors, FAST relies exclusively on simple arithmetic operations, making it particularly suitable for processors with limited computational resources. This characteristic is likely one of the reasons it was employed in the Ingenuity helicopter [66]. The pipeline of the FAST algorithm is remarkably straightforward and follows these steps:

1. Select a pixel p in the image and retrieve its intensity I_p;
2. Define a threshold t;
3. Consider a Bresenham circle of radius 3 around p, as shown in Figure 5.15;
4. Compare the intensity of p with the pixels directly above, below, left, and right (commonly referred to as positions 1, 5, 9, and 13). If, at least, three of these

four pixels are either all brighter ($\geq I_p + t$) or all darker ($\leq I_p - t$), proceed to the next step; otherwise, skip to the next pixel.

5. Compare the intensity of p with the remaining pixels within the Bresenham circle. Among all 16 pixels, at least 12 must be either all brighter ($\geq I_p + t$) or all darker ($\leq I_p - t$). If this condition is satisfied, classify p as a corner.

6. Iterate through all pixels in the image.

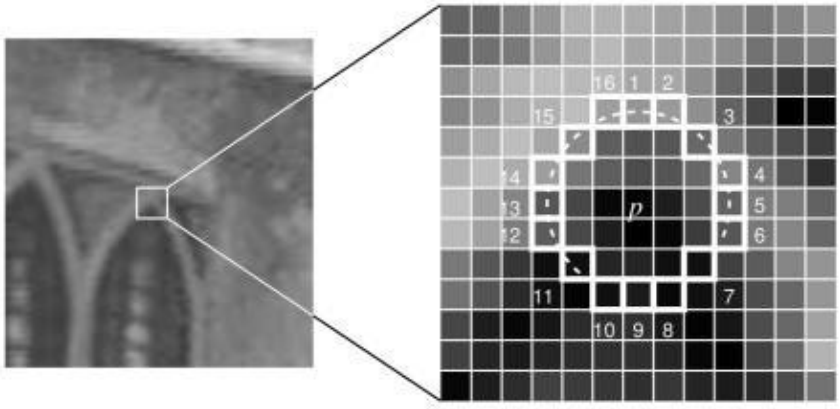


Figure 5.15: 12 point segment test corner detection in an image patch [91]

Numerous optimizations were made in order to enhance the accuracy and performance of the system. Ingenuity detects a total of 252 features per image at a frequency of 30 Hz [66]. This project aimed to reduce these numbers, ultimately detecting a maximum of 200 features per image at a frequency of 1 Hz.

Similar to Ingenuity [66], the image is divided into a 3×3 grid to ensure a homogeneous distribution of features across the entire image. The system also addresses the challenges posed by varying terrain and light conditions, where certain parts of the image might be too dark or too bright for the FAST algorithm to reliably detect features. To mitigate the loss of information in such cases, a strategy was implemented to preserve the overall feature count. If the number of features detected in a given cell is less than 25% of 22 (*i.e.*, 200 features divided by 9 cells), then the maximum number of features that can be detected in the subsequent cell is adjusted as follows:

$$f = fpc + c, \tag{5.5}$$

where f is the total number of features for that cell, fpc is the standard maximum number of features per cell, and c is the difference between the number of features detected in the previous cell and its maximum allowable feature count. The choice of 25% was based on testing, as this threshold could not be set too high. A higher value would result in frequent alternation between no compensation and compensation in subsequent images, leading to information loss between frames. This

occurs because corresponding features would fail to be consistently detected across contiguous images.

Once an array of possible feature matches within a given cell is populated, the next step is to select the best features. This is primarily achieved through a variation of Non-Maximum Suppression. Each feature, in addition to its coordinates, is associated with a value that represents the cumulative norm sum of the differences between the pixels within the Bresenham circle and the central pixel. The goal is to select features where this value is the highest, effectively indicating that the surrounding neighborhood is significantly different from the central pixel.

This approach alone, however, is insufficient, as features can be clustered too closely together. It is crucial not only to distribute features evenly across the entire image but also uniformly within each cell. Failure to address this issue can result in several problems, including degeneracy due to collinearity, coplanarity, numerical instabilities, and a lack of diversity in the hypotheses generated during RANdom SAmple Consensus (RANSAC), which will be discussed later. Additionally, closely spaced features increase the likelihood of ambiguity in matching feature descriptors. To address this issue, each feature is compared with all previously stored features before being added to ensure a minimum Euclidean distance of 17 px. Values between 12 and 24 px were found to be optimal based on testing (refer to Appendix D), but the minimum of 17 px was chosen as it provided a good balance. This threshold ensures that features are sufficiently spread out to improve distribution while avoiding excessive restrictions that could hinder the feature detection process.

One particularly successful test (refer to Appendix E), conducted in this project, involved iterating over every other pixel, as opposed to iterating over every pixel as described above. This modification significantly increased the system's speed while maintaining the number of features detected without any noticeable performance deficiencies. The improvement in performance can be attributed to a key observation. When iterating over every pixel, an unintended phenomenon occurred in regions with extremely high contrast, which are particularly prone to feature detection. The feature search progresses from left to right within each grid cell. However, in high-contrast areas, the array of possible contender features—of size 500 per cell—often fills up quickly before even half the cell's width is covered. As a result, all selected features (maximum of 22) tend to be clustered on the leftmost side of the cell. This clustering was counterproductive, as the features were no longer evenly distributed across the image grid. Furthermore, since a minimum threshold distance between features is required, this often led to a reduction in the total number of detected features. The optimization of iterating over every other pixel effectively mitigated this issue, ensuring a more uniform distribution of features and achieving the desired detection counts without compromising performance.

BRIEF allows the system to generate a blueprint for a given coordinate. In

the literature, BRIEF [92] is typically accompanied by a number k, which indicates its size in bytes. The variant employed in this case was BRIEF-16, as smaller variants did not provide sufficient robustness and led to many erroneous matches. The underlying principle of BRIEF is straightforward:

1. Define a window of size N;
2. Create $k \times 8$ pairs of coordinate pairings from the range $\left[-\frac{N}{2}, \frac{N}{2}\right]$ and store them;
3. Detect feature coordinates;
   (a) Conduct the following steps iteratively over every feature:
      i. Position the patch such that the feature is at the center;
      ii. For each pair of coordinates stored, compute the intensity difference between the pixel at the first and second coordinates;
      iii. If the intensity of the first pixel is greater than the second, store a 1; otherwise, store a 0. This can be expressed mathematically as:

$$value = \begin{cases} 1 & \text{if } p(n_1) > p(n_2) \\ 0 & \text{otherwise} \end{cases}$$

      iv. Repeat the process for every feature, storing the corresponding values.
4. Go back to Step 3.

In summary, this method produces a binary string composed of 1s and 0s, which serves as a distinctive signature or "footprint" for each feature.

ORB [93] builds upon the principles of the two preceding algorithms and enhances them. The first significant improvement involves adapting FAST to achieve scale invariance. This is accomplished by downsampling the image to a lower resolution, extracting features at that resolution, and then scaling it back up, gradually. This process is referred to as pyramid feature detection, where the top of the pyramid corresponds to lower-resolution images, and the base represents the original image. This project implements three levels of resolution: 1, 2, and 4, corresponding to the original image, half the resolution, and one-fourth the resolution, respectively. An additional level at one-eighth the resolution was also tested, but the trade-off between computational time and the number of detected features was deemed insufficiently beneficial. This layer provided a maximum of three additional features across the entire image. More often than not, it detected only one or even zero features, and, as a result, it was deemed unsuitable. The algorithm used for downsampling is presented in Algorithm 5.4.

```
1  function DOWNSAMPLING(image, N)
2      Initialize d_image
3      badIndices ← 0                                  ▷ Counter for invalid indices
4      for i ← 0 to MONOCULAR_HEIGHT step N do
5          for j ← 0 to MONOCULAR_WIDTH step N do
6              startX ← j, startY ← i
7              avgPixel ← AverageBlockPixel(image, startX, startY, N)
8              downsampledIndex ← (i / N) * d_image.width + (j / N)
9              if downsampledIndex < H / N * W / N then
10                 d_image.data[downsampledIndex] ← avgPixel
11             else
12                 badIndices ← badIndices + 1
13             end if
14         end for
15     end for
16     if badIndices > 10 then
17         return false          ▷ Downsampling failed due to excessive bad indices
18     end if
19     return true                                     ▷ Downsampling successful
20 end function
```

Algorithm 5.4: Image downsampling employed algorithm

The function in Line 7 operates similarly to the one in Line 10 of Algorithm 5.3, and thus is not explicitly shown. It became necessary to determine the optimal number of features to be detected at each level. Given that the total number of features was limited to 200, it was essential to allocate a maximum number of features for each level. To address this, a dynamic approach was chosen over a fixed allocation, allowing for flexibility in the event of changes to the maximum global feature count. Equation 5.6 illustrates the approach adopted:

$$features = \max\left(MinFPL,\ \frac{MaxFP \times \frac{4}{level}}{7}\right), \tag{5.6}$$

where *MinFPL* denotes the minimum number of features per level, set to nine (one per cell), and *MaxFP* represents the maximum number of feature points. Similar to the compensation applied for a lack of features within grids, features are also compensated at the next level if the previous one fails to detect the maximum number. However, this approach is not percentage-based. Unlike the grid-level compensation, which required a minimum of 25%, the difference is instead added directly to the next level. This is because there is no substantial impact or noticeable consequences from assigning slightly more or significantly more features to the next level, allowing for a direct approach. A slight optimization was introduced to prevent features from being detected at the bottom of the image, as this area would not contribute to any overlap between consecutive frames.

ORB also enhances the FAST method by determining the orientation of features using an intensity centroid [93]. This enables BRIEF to become rotation-aware. Instead of using only the original set of $k \times 8$ pairings, as previously explained, ORB generates additional sets by rotating the initial (zero-rotation) pairings in increments of 12.5 °. Depending on the feature orientation, different steered pairings are utilized. Figure 5.16 illustrates the outcome of the ORB implementation, with the image divided into a 3×3 grid.

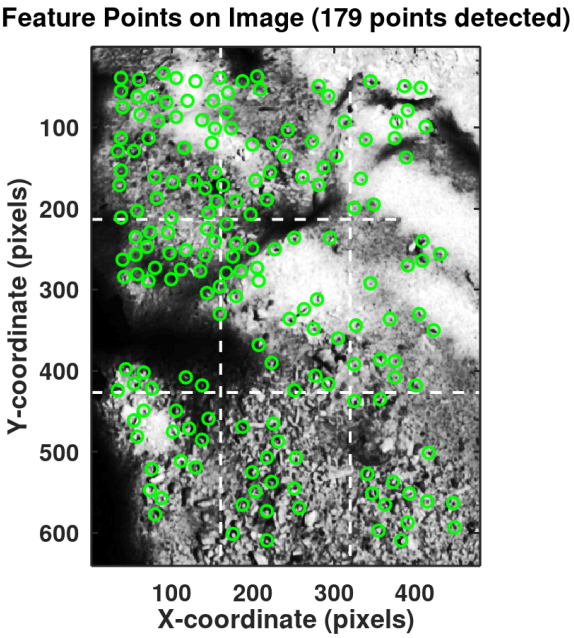


Figure 5.16: ORB implementation in Fenix

Feature Matching

At this stage, the feature coordinates and their footprints are identified. The next step is to determine which features in the first image correspond to those in the second image, *i.e.*, identifying similar points in consecutive images. The method used for the descriptor-based approach was the Hamming distance. Given two binary descriptor arrays, one from the first image and one from the second, the Hamming distance is defined as the number of differing elements between the arrays. For instance, the Hamming distance between 1110001 and 1110011 is 1, as only the second bit differs. If the Hamming distance between these arrays falls below a specified threshold (set to 25), the features are considered a match. Setting this threshold too low would result in an insufficient number of matches, as it would be more challenging to meet the criteria. Conversely, setting it too high would lead to the detection of numerous erroneous matches. While multiple features could theoretically satisfy this threshold, this project only retains the match with the lowest Hamming distance for a given feature.

However, comparing the descriptor of a feature against all features in the other image would be highly inefficient. To address this, descriptors are only compared between features located within a predefined window. In other words, features are only compared if they are located within the same region. This greatly reduces the computational time and improves the performance of the system.

Outlier Rejection and Motion Estimation

Inevitably, erroneous matches will occur, where the system identifies two features as a match when they are not. Removing these incorrect matches is crucial, as they lead to inconsistent epipolar geometry. To address this, RANSAC [94] was employed to eliminate outliers. Specifically, the Essential matrix was estimated using the 8-point algorithm. Although Nistér's 5-point algorithm [95] theoretically requires fewer points, it is relatively more complex and less straightforward. The 8-point algorithm, on the other hand, provided reliable results while simplifying the estimation process.

RANSAC is an iterative algorithm that operates by selecting the minimum number of random samples required to estimate a model. This model is then tested against the entire dataset to identify its consensus set, which consists of points that fit the model within a predefined tolerance. The iteration with the largest consensus set is deemed the best estimate, and its model is selected. Dataset points that do not align with this model are classified as outliers, while those that do are considered inliers. The number of iterations can be calculated by:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}, \tag{5.7}$$

where N is the number of iterations required, p is the desired probability of selecting at least one inlier-only subset, ϵ is the proportion of outliers in the dataset, and s is the size of the subset required to estimate the model [96]. In this thesis, p was set to 0.995, ϵ to 0.4, and s to 8. Using Equation 5.7, the required number of iterations can be calculated:

$$N = \frac{\log(1-0.995)}{\log(1-(1-0.4)^8)} = 312.791 \approx 313 \tag{5.8}$$

The variable values were selected based on testing. For example, p was set as high as possible, but increasing it further resulted in slower performance, as even a small increment significantly increased the number of required iterations. Nevertheless, it is common to increase the number N for robustness, sometimes even by a factor of 10 [97]. This is not mandatory and is generally done for tasks with a high demand for accuracy. For this particular case, only 100 additional iterations were added, as multiplying by 10 would not be feasible on modest computers.

The 8-point algorithm initially fills an 8×9 matrix $\mathbf{A}$ with the following format [98, 99]:

$$\mathbf{A} = \begin{bmatrix} x_1 \cdot x'_1 & y_1 \cdot x'_1 & x'_1 & x_1 \cdot y'_1 & y_1 \cdot y'_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_8 \cdot x'_8 & y_8 \cdot x'_8 & x'_8 & x_8 \cdot y'_8 & y_8 \cdot y'_8 & y'_8 & x_8 & y_8 & 1 \end{bmatrix} \tag{5.9}$$

These coordinates were normalized using Equation 5.10, which maps the coordinates to the camera coordinate system [100]. Normalizing the coordinates is crucial, as failure to do so will result in significant numerical errors [99]. This occurs because the matrix $\mathbf{D}$ in the Singular Value Decomposition (SVD) decomposition would contain two large non-zero singular values and one zero [100]. In the context of SVD, $\mathbf{D}$ is a diagonal matrix that represents the singular values of the decomposed matrix, which indicate the magnitude of the principal components of the data. Proper normalization ensures that these singular values are appropriately scaled for reliable computation. Without the intrinsic camera matrix $\mathbf{K}$, it is not possible to estimate the Essential matrix $\mathbf{E}$, as it assumes calibrated cameras.

$$^{k}x_{\text{normalized}} = \mathbf{K}^{-1} \cdot x \tag{5.10}$$

Before proceeding, a major concern is verifying whether $\text{rank}(\mathbf{A}) = 8$. If this condition is not met, the algorithm cannot proceed, as the selected pairings exhibit linear dependencies and will not yield a reliable estimate. If, during a given run of the RANSAC algorithm, too many of these degeneracies are found, the process will be aborted, as reliable estimates likely cannot be obtained. Some of the causes for this include the absence of a baseline between frames (little or no motion), pure rotation, or too few matches. Epipoles are the points where the line joining two camera centers intersects the image planes. When there is no translation between the views (such as in pure rotation), the epipoles become degenerate. This absence of a well-defined baseline makes it impossible to accurately define epipolar geometry. If the rank is correct, Equation 5.11 is solved using SVD.

$$\mathbf{A} \cdot \mathbf{e} = 0, \tag{5.11}$$

where

$$\mathbf{e} = \begin{bmatrix} \mathbf{E}_{11} & \mathbf{E}_{12} & \mathbf{E}_{13} & \mathbf{E}_{21} & \mathbf{E}_{22} & \mathbf{E}_{23} & \mathbf{E}_{31} & \mathbf{E}_{32} & \mathbf{E}_{33} \end{bmatrix}^{\top} \tag{5.12}$$

The SVD algorithm decomposes the matrix as [100]:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^{\top}, \tag{5.13}$$

where the elements of $\mathbf{E}$ are located in the last column of $\mathbf{V}$. An additional important step is performing rank deprivation, as $\text{rank}(\mathbf{E}) = 2$ and ensuring that $\det(\mathbf{E}) = 0$ [98]. This was achieved by applying the SVD algorithm again to the $\mathbf{E}$ matrix and modifying $\mathbf{D}$ to a diagonal matrix where $\mathbf{D}_{11} = \mathbf{D}_{22} = 1$ and all other elements are set to zero [98, 99, 100].

The RANSAC algorithm then leverages the coplanarity constraints described in Equation 5.14 [100] to determine whether a given pair fits the estimated matrix $\mathbf{E}$. Naturally, since the system operates under uncertainties, this value will never be exactly zero but will be very close to it. Pairs that satisfy this condition are considered inliers, while those that do not are classified as outliers. The model with the most inliers is considered the best estimate.

$$ {}^{k}x''^{\top} \cdot \mathbf{E} \cdot {}^{k}x' = 0 \tag{5.14} $$

Finally, the model is re-estimated using all inliers to obtain the most accurate estimate. The matrix $\mathbf{A}$ is redefined, this time with dimensions InlierCount $\times$ 9, as opposed to its previous 8×9 configuration.

After estimating the Essential matrix $\mathbf{E}$, the next step is to retrieve the rotation matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$. The vector $\mathbf{t}$ is a unit vector, meaning its magnitude is equal to one, or $||\mathbf{t}|| = 1$. Therefore, if $\mathbf{t} = [1, 0, 0]^{\top}$, it does not imply that the motion was exactly 1 meter along the x axis. Instead, it indicates that the relative motion between the frames occurred entirely along the x axis, with the actual scale of the motion remaining undetermined [98, 100]. What this means is that $\mathbf{E}$ has a property where the relative orientation and the direction of the translation can be determined but its scale can't or, in other words, it is not possible to determine if the motion was, for instance, 10 cm or 10 m. To extract $\mathbf{R}$ and $\mathbf{t}$, the SVD decomposition of $\mathbf{E}$ is used. As previously discussed, $\mathbf{E}$ can be expressed as:

$$ \mathbf{E} = \mathbf{U} \cdot \mathbf{D} \cdot \mathbf{V}^{\top}, \tag{5.15} $$

where $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ are the matrices obtained from the decomposition. Using this, the following auxiliary matrices were defined [98, 100]:

$$ \mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} $$

With these matrices, the possible solutions for $\mathbf{R}$ and $\mathbf{t}$ are derived as [98, 100]:

$$ \mathbf{R} = \mathbf{U} \cdot \mathbf{W} \cdot \mathbf{V}^{\top} \quad \text{or} \quad \mathbf{R} = \mathbf{U} \cdot \mathbf{W}^{\top} \cdot \mathbf{V}^{\top}, \tag{5.16} $$

$$\mathbf{t} = \mathbf{U}[:, 2] \quad \text{or} \quad \mathbf{t} = -\mathbf{U}[:, 2]. \tag{5.17}$$

Here, $\mathbf{U}[:, 2]$ represents the third column of the matrix $\mathbf{U}$. These four different solutions can be seen visually in Figure 5.17.

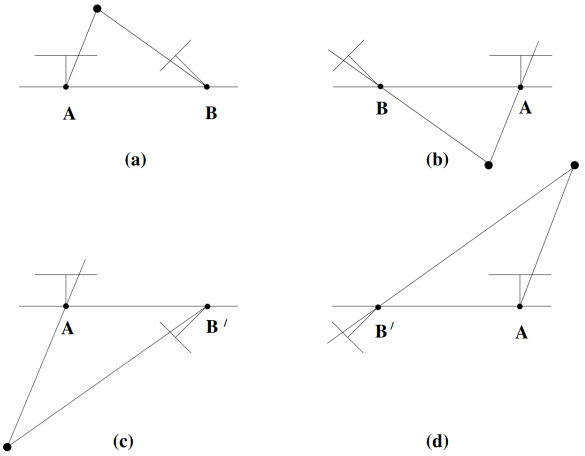


Figure 5.17: The four possible solutions for calibrated reconstruction from $\mathbf{E}$ [100]

The correct solution is selected by applying the cheirality condition, ensuring that triangulated points lie in front of both cameras. For clarity, the global tracking estimates will be denoted with the subscript "est" to distinguish them.

Before proceeding, it is checked whether the resulting rotation matrix satisfies $\det(\mathbf{R}) = 1$. To also verify that the matrix is orthonormal, Equation 5.18 is used, which should ideally result in the identity matrix. If the computed result deviates from the identity matrix beyond a small threshold, an error is reported, and the matrix is reset to the identity matrix. If both the orthogonality and determinant conditions are satisfied, the rotation matrix is considered valid, and the algorithm will continue.

$$\mathbf{R} \cdot \mathbf{R}^{\top} = \mathbf{I} \tag{5.18}$$

If the rotation matrix is valid, $\mathbf{R}$ and the translation vector $\mathbf{t}$ are updated using the following equations [101]:

$$\mathbf{R}_{\text{est}} = \mathbf{R} \cdot \mathbf{R}_{\text{est}} \tag{5.19}$$

$$\mathbf{t}_{\text{est}} = \mathbf{t}_{\text{est}} + \mathbf{R}_{\text{est}} \cdot (s \cdot \mathbf{t}), \tag{5.20}$$

where s represents the unknown scale. To determine the scale, the parallax effect was used iteratively for every inlier with the aid of the LRF sensor, as shown in Equation 5.21. The mean of the resulting values is then taken. This value also

represents the estimated velocity of the drone in meters per second, which aligns with the traveled distance per second, given the operating frequency of 1 Hz.

$$s = \frac{h \cdot \Delta p}{f}, \tag{5.21}$$

where h is the measured height, Δp is the Euclidean parallax distance, and f is the focal length of the image where the feature was acquired. This equation makes use of a triangular relationship between the internal camera and the outside, as part of a pinhole camera model, illustrated in Figure 5.18 [102].

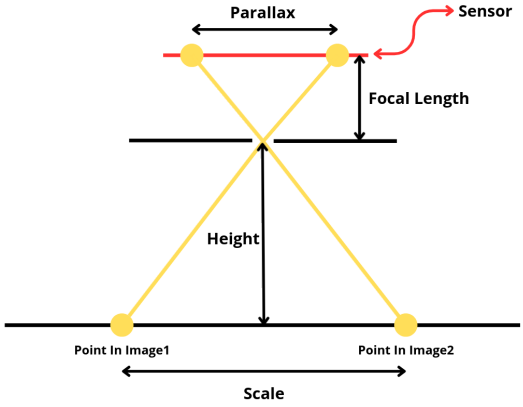


Figure 5.18: Triangular relationship

The smaller inverted triangle has a direct relationship with the larger one, meaning that:

$$\frac{s}{h} = \frac{\Delta p}{f} \tag{5.22}$$

which results in Equation 5.21 when solved for scale.

However, similar to many drone applications, this method assumes a planar scene. Despite this constraint, the raw monocular odometry demonstrated an Absolute Trajectory Error (ATE) not exceeding 5% of the traveled distance for unidimensional trajectories, which was deemed successful, with a drift of less than 0.05 $\mathrm{m\,m^{-1}}$ given the tests performed and presented in Chapter 6. It is important to note that these values are raw and have not undergone Kalman Filtering or any other estimation techniques.

As previously mentioned, pure rotation cannot be determined using the Essential matrix. This is because the estimation requires a translation component; without it, the system would be degenerate, and the system would not yield a reliable estimate.

To deal with such cases, the homography matrix $\mathbf{H}$ is estimated. If the Essential matrix cannot be determined, the system will proceed to estimate the homography matrix. A process similar to the estimation of the Essential matrix is conducted. Unlike this matrix, that encodes the epipolar geometry between two views and is

used to determine the relative pose between two calibrated cameras, the homography matrix is a projection matrix that relates corresponding points between two images of the same planar surface. In other words, pure rotations are related by homography, not epipolar geometry. It is important to note that this method cannot replace the estimation of $\mathbf{E}$ because homography assumes that the cameras rotate about the center of projection between frames [103]. This assumption does not hold in the presence of translation [104, 103]; therefore, it was implemented to aid with the downfalls of the Essential matrix.

A similar process is conducted using RANSAC to determine the best estimate of $\mathbf{H}$. Only four pairs of points are required, significantly reducing the number of iterations needed as calculated in Equation 5.23. Once again, additional iterations were added to this number. However, instead of 100, only 20 were included.

$$N = \frac{\log(1 - 0.995)}{\log(1 - (1 - 0.4)^4)} = 38.17 \approx 39 \tag{5.23}$$

Similarly to the estimation of the Essential matrix, a matrix $\mathbf{A}$ will be constructed in the format shown below:

$$\mathbf{A} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 \cdot x_1 & -x'_1 \cdot y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 \cdot x_1 & -y'_1 \cdot y_1 & -y'_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4 \cdot x_4 & -x'_4 \cdot y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4 \cdot x_4 & -y'_4 \cdot y_4 & -y'_4 \end{bmatrix} \tag{5.24}$$

SVD is then used to solve Equation 5.25.

$$\mathbf{A} \cdot \mathbf{h} = 0, \tag{5.25}$$

where the vector $\mathbf{h}$ is defined as:

$$\mathbf{h} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & \mathbf{H}_{13} & \mathbf{H}_{21} & \mathbf{H}_{22} & \mathbf{H}_{23} & \mathbf{H}_{31} & \mathbf{H}_{32} & \mathbf{H}_{33} \end{bmatrix}^\top \tag{5.26}$$

Then, the rotation matrix is extracted as the 2×2 matrix located in the top-left corner of $\mathbf{H}$:

$$\mathbf{R}_{2\times2} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix} \tag{5.27}$$

and subsequently converted into a 3×3 rotation matrix $\mathbf{R}_{3\times3}$ by appending a third row and column as follows:

$$\mathbf{R}_{3\times3} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & 0 \\ \mathbf{H}_{21} & \mathbf{H}_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.28}$$

The rotation matrix $\mathbf{R}_{3\times3}$ is decomposed using SVD. To correct it and aid with numerical stabilization, the product of the matrices $\mathbf{U}$ and $\mathbf{V}^\top$ is computed as follows:

$$\mathbf{R} = \mathbf{U} \cdot \mathbf{V}^\top \tag{5.29}$$

Similar verifications are performed using Equation 5.18, as well as ensuring that $\det(\mathbf{R}) = 1$. If the conditions are satisfied, the trajectory is updated according to Equation 5.19.

5.2.2 State Estimation

Implementing an Unscented Kalman Filter (UKF) was considered; however, a standard EKF was ultimately selected. While the UKF can offer higher estimation accuracy—being accurate up to third-order nonlinearities compared to the first-order accuracy of the EKF—this advantage comes with increased computational cost [105].

The EKF is designed to fuse data from all sensor sources relevant to navigation, including the LRF, IMU, and the monocular camera. It inherits the IMU's 200 Hz frequency, as it is event-triggered by the corresponding Robot Operating System (ROS) topic. This topic contains all the latest data from the sensors; therefore, the sensor information is presented synchronously.

Since the EKF is IMU-triggered, there will always be brand new inertial data. However, since the other sensors run at a significantly slower rate, there could be a risk of duplicate data. The system, however, manages this and guarantees no duplicates are sent.

Inertial Odometry For State Prediction

Although the monocular odometry performed successfully, as will be shown in Chapter 6, the IMU simulated in this project, which is inspired by Northrop Grumman's LN-200 [106], significantly outperforms it. This outcome is expected, given that the LN-200 is a high-quality, low-drift tactical-grade IMU. This will be reflected in the Kalman filtering, as more weight, or confidence, will be given to the IMU.

Given that this thesis assumes the drone only rotates about the z axis, the position can be derived relatively simply:

$$x = x' + v \cdot \cos(\theta) \cdot dt, \tag{5.30}$$

$$y = y' + v \cdot \sin(\theta) \cdot dt, \tag{5.31}$$

$$z = z' + v_z \cdot dt, \tag{5.32}$$

$$\theta = \theta' + \omega \cdot dt, \tag{5.33}$$

where dt represents the period and the forward velocity v, the velocity in the z axis, and the yaw rate w are given by:

$$v = v' + accel_x \cdot dt \tag{5.34}$$

$$v_z = {v_z}' + (accel_z - g) \cdot dt \tag{5.35}$$

$$\omega = gyro_z \tag{5.36}$$

The IMU is utilized in the prediction step, enabling the system to accurately estimate its state while awaiting relevant data from the other sensors.

Monocular Camera Integration For Covariance Reduction

The primary objective of the monocular camera is to reduce the covariance associated with the forward velocity v_x and the x, y position estimates. However, phenomena such as parallax can significantly degrade the accuracy of the monocular camera's measurements. Therefore, caution was exercised when updating the IMU's prediction, as its estimations are typically highly reliable and should not be compromised by the camera's comparatively noisier data.

To mitigate the potential impact of noisy measurements from the monocular camera, the innovation was evaluated prior to committing to the update. The innovation vector, **Innov**, is defined as:

$$\mathbf{Innov} = \begin{bmatrix} l_{x_{\text{pred}}} - \text{camera}_x \\ l_{y_{\text{pred}}} - \text{camera}_y \\ l_{v_{x_{\text{pred}}}} - \text{camera}_{v_x} \end{bmatrix}, \tag{5.37}$$

where $l_{x_{\text{pred}}}$, $l_{y_{\text{pred}}}$, and $l_{v_{x_{\text{pred}}}}$ are the estimated positions from the latest camera update and the camera-derived quantities are computed as:

$$\text{camera}_x = v \cdot \cos(\theta_{\text{pred}}), \tag{5.38}$$

$$\text{camera}_y = v \cdot \sin(\theta_{\text{pred}}), \tag{5.39}$$

$$\text{camera}_{v_x} = v, \tag{5.40}$$

and v denotes the velocity measurement obtained from the monocular navigation system. If the innovation in any row exceeds a given threshold, the measurement

is considered erroneous, and the system refrains from updating the state with the camera measurements.

Although the monocular system can estimate rotation, the estimates are significantly noisier than the highly accurate IMU gyroscope, as will be shown in Chapter 6. As a result, integrating the camera for θ correction would add minimal valuable data and primarily introduce noise. For this reason, it was decided not to incorporate the camera for θ correction.

LRF For Altitude Correction

Given that the camera does not provide depth estimation, the LRF aims to update the IMU altitude predictions. Since the LRF directly measures the state variable z, the innovation is simply defined as:

$$\mathbf{Innov} = z_{\text{pred}} - z_{\text{LRF}} \tag{5.41}$$

It is important to note that the altitude measured here refers to the instantaneous altitude at any given moment. While the absolute altitude, relative to the initial value, could theoretically be estimated and would be valuable for scenarios involving altitude limits, this feature has been reserved for future iterations, where such scenarios may be tested and further developed.

Extended Kalman Filter

The state is defined as a vector, **ST**, which includes the x, y, and z positions, θ, the yaw rate w, and the velocities v_x, and v_z. In the notation adopted throughout this work, the subscript *pred* denotes values computed during the prediction step, while the subscript *est* refers to values obtained during the update step. As mentioned previously, the prediction step is entirely conducted using the IMU's measurements, which is depicted in Equation 5.42.

$$\mathbf{ST}_{\text{pred}} = \begin{bmatrix} x \\ y \\ z \\ \theta \\ \omega \\ v_x \\ v_z \end{bmatrix} = \begin{bmatrix} x_{\text{est}} + v_{x_{est}} \cdot \cos(\theta_{\text{est}}) \cdot dt \\ y_{\text{est}} + v_{x_{est}} \cdot \sin(\theta_{\text{est}}) \cdot dt \\ z_{\text{est}} + v_{z_{est}} \cdot dt \\ \theta_{\text{est}} + gyro_z \cdot dt \\ gyro_z \\ v_{x_{\text{est}}} + accel_x \cdot dt \\ v_{z_{est}} + (accel_z - g) \cdot dt \end{bmatrix} \tag{5.42}$$

The algorithm proceeds by computing the following Jacobians:

$$\mathbf{F} = \frac{\partial f}{\partial \hat{\mathbf{x}}}, \quad \mathbf{G} = \frac{\partial f}{\partial \mathbf{u}}, \tag{5.43}$$

where $\hat{\mathbf{x}}$ represents the estimated state vector $\mathbf{ST}_{\text{est}}$, and $\mathbf{u}$ denotes the most recent IMU readings.

To finish the prediction step, the covariance is also predicted using:

$$\mathbf{P}_{\text{pred}} = \mathbf{F} \cdot \mathbf{P}_{\text{est}} \cdot \mathbf{F}^{\top} + \mathbf{G} \cdot \mathbf{R} \cdot \mathbf{G}^{\top}, \tag{5.44}$$

where $\mathbf{R}$ represents the diagonal covariance matrix associated with the Gaussian noise of the IMU.

The update step only works whenever the system detects new LRF or camera measurements. They function sequentially, where there are some nuances as explained above. The Jacobians of the sensor models are computed as well, denoted as $\mathbf{H}$, which is used to compute the innovation covariance, $\mathbf{S}$:

$$\mathbf{S} = \mathbf{H} \cdot \mathbf{P}_{\text{pred}} \cdot \mathbf{H}^{\top} + \mathbf{Q}, \tag{5.45}$$

where $\mathbf{Q}$ is the diagonal covariance matrix associated with either the LRF or the camera.

Then, the Kalman Gain, $\mathbf{K}$, is computed:

$$\mathbf{K} = \mathbf{P}_{\text{pred}} \cdot \mathbf{H}^{\top} \cdot \mathbf{S}^{-1} \tag{5.46}$$

The update step ends by updating the estimates:

$$\mathbf{ST}_{\text{est}} = \mathbf{ST}_{\text{pred}} + \mathbf{K} \cdot \mathbf{Innov}, \tag{5.47}$$

$$\mathbf{P}_{\text{est}} = (\mathbf{I} - \mathbf{K} \cdot \mathbf{H}) \cdot \mathbf{P}_{\text{pred}}, \tag{5.48}$$

where $\mathbf{I}$ is the identity matrix.

Part III

Results and Analysis

Chapter 6

Tests and Results

This chapter presents the results obtained from the simulation environment. Key performance metrics and observations are discussed to evaluate the effectiveness of the proposed GNC system. The outcomes provide insight into the system's behavior under representative planetary conditions. Figure 6.1 illustrates the simulated environment and surroundings where these tests occurred. The red arrow points to the drone.

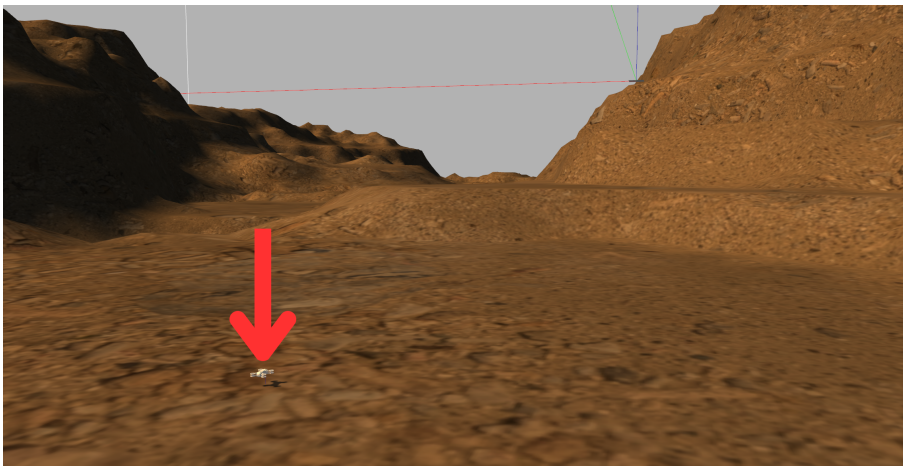


Figure 6.1: Terrain the drone is flying over and its surroundings

6.1 Path Planning

This section aims to evaluate more complex scenarios, including maps with obstacles and trajectories involving multiple stop points. The trajectories throughout this section are variations of a base scenario, shown in Figure 6.2, each introducing distinct characteristics to highlight specific behaviors and edge cases of the path planning algorithm. In the upcoming scenarios, the green dot marks the starting position, and the red dot marks the goal. Black pixels represent obstacles, while the yellow line traces the computed path. Additionally, in the performance figures for A*—captioned accordingly—gray and blue pixels are used to visualize the algorithm's internal process: gray represents all visited nodes, while blue highlights nodes that would have been explored if the system had not reached the destination.

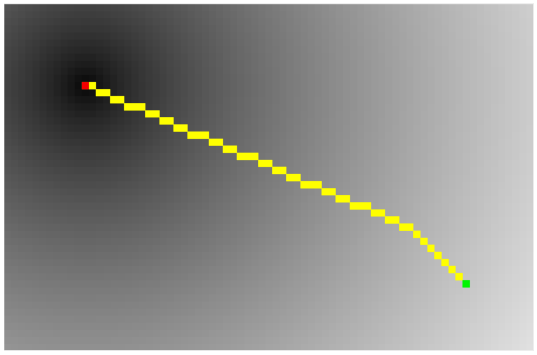


Figure 6.2: Generated trajectory with final position at $(x, y) = (-27, 14)$ m, 2 $\mathrm{px\,m^{-1}}$ and Euclidean distance heuristics

The next test introduces multiple obstacles arranged to resemble a maze. The objective was not to simulate a real-world scenario but to create an extreme case that pushes the A* algorithm to its limits[1].

Figure 6.3 provides detailed insight into the algorithm's performance.

A stop point at $(x, y) = (-25, 3)$ m was added to the previous trajectory, with the resulting path shown in Figure 6.4. The slightly darker gray pixels indicate the nodes visited during the second iteration, from the stop point to the final destination.

Notice the significant difference in the number of visited pixels across the different missions. Since the stop point lies primarily along the x-axis relative to the starting position, with minimal variation in the y-axis, the algorithm avoids exploring the upper region of the map. This is due to the fact that nodes in that area would result

[1] Animations illustrating the system's performance were created and can be found in https://github.com/tiagoassp/drone_gnc. These videos do not represent online performance and are intended solely as visualizations of the algorithm's behavior; they do not reflect the actual processing speed.

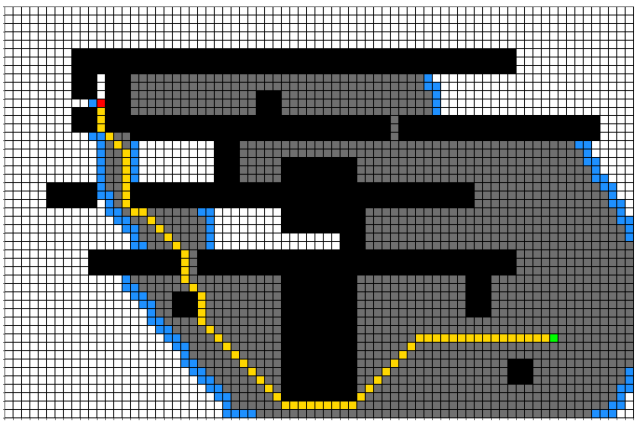


Figure 6.3: A* performance: Generated trajectory with several obstacles and Euclidean distance heuristics

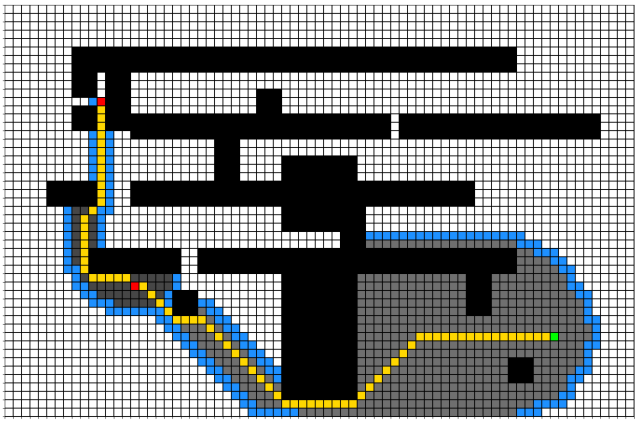


Figure 6.4: A* performance: Generated trajectory with several obstacles and Euclidean distance heuristics and stop point

in a much higher cost to reach the stop point, making them less favorable in the search process.

Limitations

The limitations of the A* algorithm—and, by extension, the limitations of this system—are primarily dictated by the capabilities of the processor employed. While the algorithm is capable of determining optimal paths even in maps comprising hundreds of pixels, this comes at a significant computational cost[2]. In compliance with requirement S-1 (refer to Chapter 4), the path planning process is bounded by a fixed iteration limit, as suggested by NASA, of 15,000 steps. In one specific test case involving a trajectory with no obstacles and a goal position at coordinates $(x, y) = (150, 77)$ m, the system was unable to complete the A* search before

[2]Although this is true for all heuristics, the Euclidean distance suffers a particularly large performance penalty.

reaching this iteration ceiling[3]. The total computation time in this scenario was approximately 68 s.

By increasing the maximum number of allowed iterations to ten times the previous value, the system was indeed able to compute an optimal path, as shown in Figure 6.5. The resulting trajectory has a length of 300 px, involved 26,945 visited nodes, and required over 10 min to complete. This process was so extensive that the log file generated by the system, which is used for visualization and animation scripts, contained over 15 million lines.

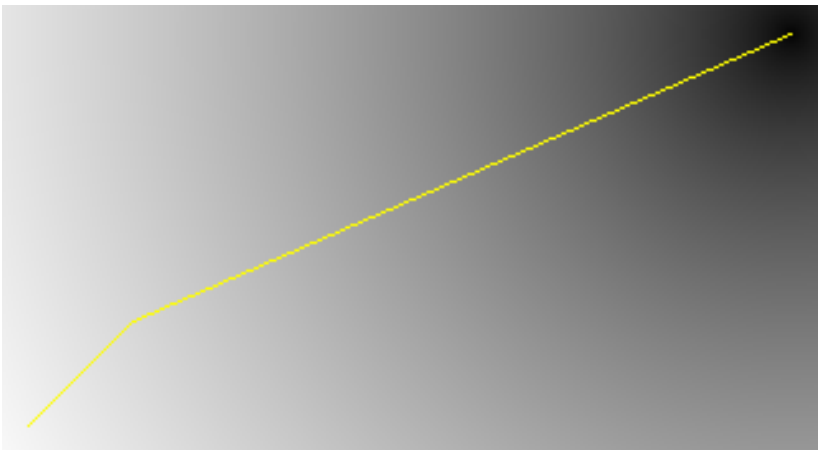


Figure 6.5: Generated trajectory for goal at $(x, y) = (150, 77)$ m and Euclidean distance heuristics

It is worth noting that obstacle-free maps are, perhaps counterintuitively, significantly more computationally demanding than maps containing obstacles. In the absence of obstacles, the number of pixels that can potentially be explored may increase substantially, leading the algorithm to consider a far greater number of possibilities. Conversely, obstacles act as natural constraints, reducing the search space and, therefore, the computational load. It is important to remember that the A* algorithm relies on two lists and, as the map size increases, the arrays corresponding to these lists grow larger. This results in an increase in both the computational cost and the time required to process the algorithm, as the system becomes slower with each iteration.

During the execution of this test, the computer running this project, equipped with an i5-14400F processor featuring 16 cores, was monitored, and some cores reached 100% utilization. As a result, solutions were sought to address the need for efficiently handling larger trajectories in more limited computers.

The first solution is to divide any larger trajectory into two, or potentially more, smaller trajectories, thereby reducing the computational load at any given time. The second step would involve the drone dynamically determining whether a given

[3]This scenario is only problematic with Euclidean distance heuristics. Manhattan distance computed the same trajectory in approximately 0.3 s.

destination is too far to compute reliably in a single run. If so, the drone could compute intermediate checkpoints along the way, effectively adding more stop points to distribute the computational load across multiple waypoints.

A trajectory whose final point was $(x, y) = (40, 20)$ m with 2 px m^{-1} was, however, very processor-friendly, taking only 1.34 s. A more thorough investigation would have to be made, specifically with hardware in the loop, in order to determine what the Fenix's processor could handle, at what cost, and how that would affect the rest of the system.

Despite all these potential solutions, the simplest approach is to use the Manhattan distance heuristic. For comparison, this method required only 1.41 s to compute a trajectory on an obstacle-free map, with final coordinates at $(x, y) = (300, 100)$ m and a resolution of 2 px m^{-1}. Extending the heuristic's capabilities even further, it took 47 s for a trajectory with final coordinates at $(x, y) = (1000, 300)$ m.

6.2 Monocular Odometry

Several tests were conducted to evaluate and validate the performance and behavior of the monocular odometry. Initially, one-dimensional trajectories were tested, which later progressed to more complex paths. All tests were conducted in the same region shown in Chapter 5 and followed the complete pipeline presented therein. The results obtained are presented in Table 6.1, and graphically in Figure 6.6.

Table 6.1: Raw performance metrics of monocular odometry across different trajectory lengths, with movement occurring along a single axis[4]

Length (m)	**RMSE** (m)	**Drift** (m m^{-1})	**ATE** (%)	**ETD** (m)	**RTD** (m)
30	0.687641	0.015498	2.3561	31.3497	29.1858
50	1.2345	0.019175	2.5294	52.4862	48.8055
75	1.8738	0.01986	2.5587	78.2081	73.2333
150	3.7368	0.021341	2.6301	150.9953	142.0758
200	5.2235	0.022562	2.7395	201.341	190.6715
300	8.5356	0.024583	2.9407	304.0605	290.2577

With these results, it is clear that the monocular odometry is functioning as expected prior to any filtration. All errors increase as the system travels larger distances, which is expected in odometry, as marginal errors accumulate over time, leading to progressively larger discrepancies. A deeper analysis was conducted to investigate certain phenomena observed during these tests. For instance, consider the 75 m trajectory. Figure 6.7a illustrates the estimated velocity, which, as explained earlier, also corresponds to the scale. Based on the logs acquired, the mean

[4]Table acronyms: Root Mean Square Error (RMSE), Estimated Travel Distance (ETD) and Real Travel Distance (RTD).

estimated velocity was approximately 2.65 $\mathrm{m\,s^{-1}}$, while the velocity retrieved from the `/odom` topic was around 2.45 $\mathrm{m\,s^{-1}}$. The actual velocity was 2.5 $\mathrm{m\,s^{-1}}$.

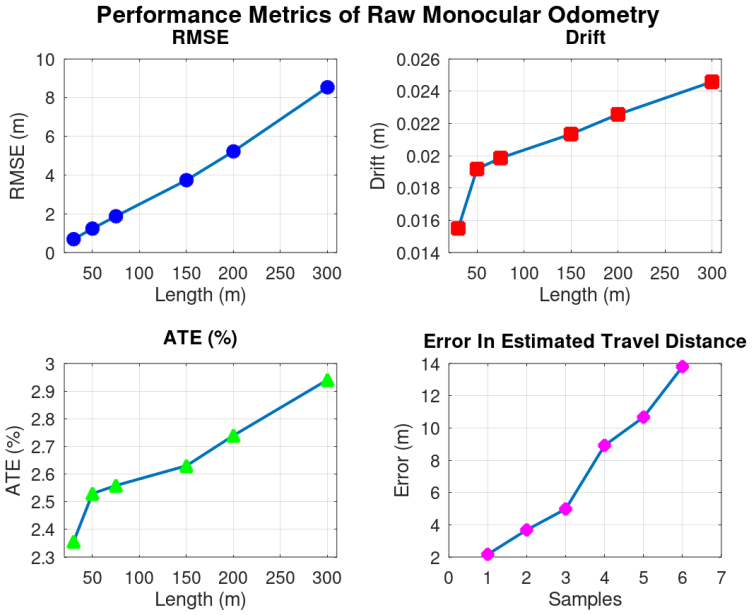
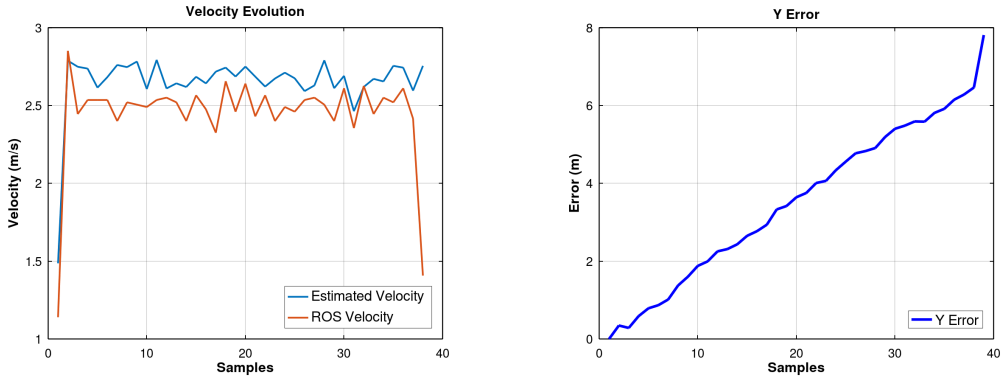


Figure 6.6: Graphical comparison of vertical of the performance metrics in Table 6.1



(a) Estimated velocity

(b) Y error

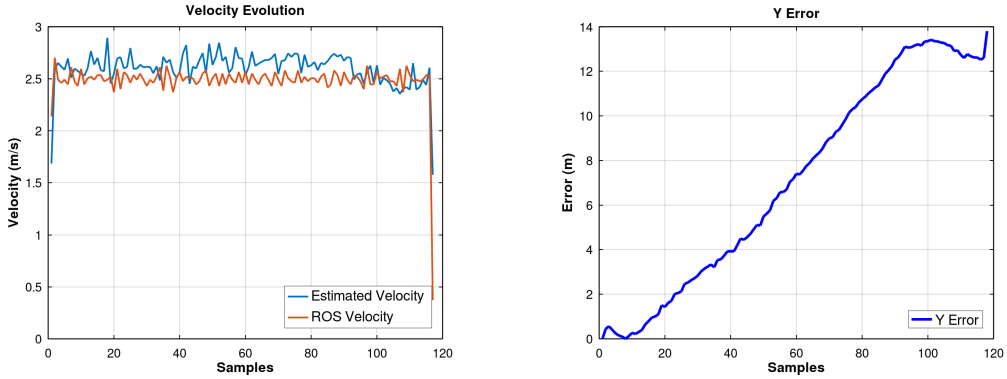
Figure 6.7: Velocity and Y error for the 75 m trajectory

This discrepancy is expected and arises due to noise in the image data and the LRF as well as the assumption of a flat surface. In reality, the terrain includes bumps and height fluctuations, which lead to varying parallax distances for different features[5]. Additionally, features are detected at multiple resolutions, requiring parallax distances to be converted back to the original resolution, further compounding

[5]Model used and its Gazebo configurations available at https://github.com/tiagoassp/drone_gnc/tree/master.

the error. As the system progresses along the trajectory, these cumulative errors become increasingly apparent, as shown in Figure 6.7b.

Let us now take a closer look at the 300 m trajectory, where the system exhibited a different behavior. Figure 6.8a shows the estimated velocity, which dips slightly below 2.5 $\mathrm{m\,s^{-1}}$ around the ninetieth sample—an unusual event compared to previous trials. This anomaly is also reflected in the trajectory error plotted in Figure 6.8b, where the error initially increases similarly to the 75 m case but then unexpectedly stabilizes and even begins to decrease.



(a) Estimated velocity

(b) Y error

Figure 6.8: Velocity and Y error for the 300 m trajectory

The reason behind this phenomenon is quite simple and further illustrates the particularities of the planar scene assumption. Around the ninetieth sample, the system was flying over the region depicted in Figure 6.9. Notice, on the left side of the image, the large depression in the terrain, resembling a deep canyon. Although it might not appear so at first glance, this region is approximately 50 m deeper than the surrounding area. The parallax of the features in that region will be significantly smaller than those on the ground directly below the drone, as it is inversely proportional to the altitude. Consequently, the average parallax of all features will decrease, leading to a corresponding reduction in the estimated scale, according to Equation 5.21. As can be seen in the right plot of Figure 6.9, there is a significant cluster of features in this region.

Conversely, the constant overestimation of the 75 m trajectory is also likely due to the approximation to the wall on the right side of the image, which is less visible in the 300 m trajectory. Features detected in this region would exhibit a larger parallax, increasing the mean, and therefore the scale.

The next step involved testing more complex trajectories that included multiple rotations. Table 6.2 summarizes the tests conducted and the system's performance. A significant increase in all error metrics was observed, which is expected. In one-dimensional trajectories, errors grow due to variations in a single variable. In contrast, for these tests, the system operates in a two-dimensional space, introducing

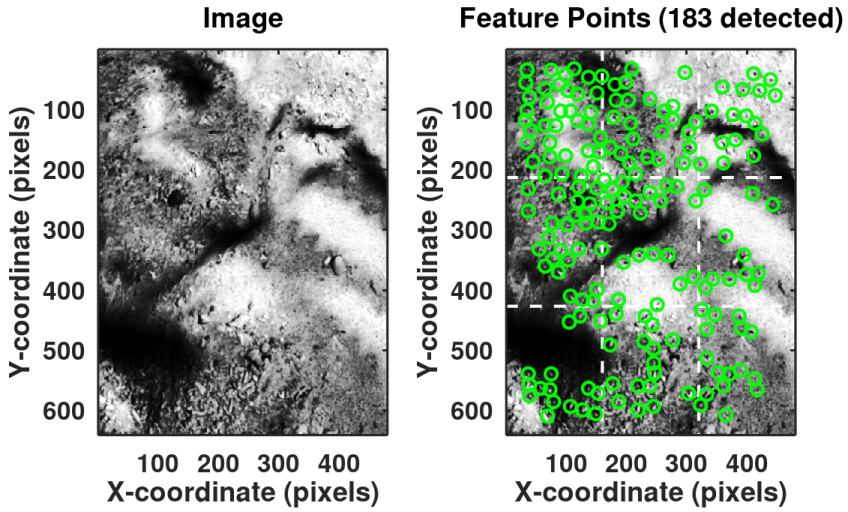


Figure 6.9: Region where the drone flew over and the features detected

errors not only in the y direction but also in the x direction and yaw.

Table 6.2: Raw performance metrics of monocular odometry across rotational paths[6]

Trajectory Name	**RMSE** (m)	**Drift** ($\mathrm{m\,m^{-1}}$)	**ATE** (%)	**ETD** (m)	**RTD** (m)	**RE** (°)
Straight-20 °	2.6221	0.059436	7.1285	42.1798	36.7839	3.5659
Straight-45 °	3.6327	0.095994	11.243	37.8699	32.3107	14.6206
Multiple Rotations	19.9714	0.091964	10.8592	213.0303	183.9123	15.7698
Full Test	43.7391	0.24403	29.2558	172.6181	149.5057	8.5234

One particular observation stood out when analyzing the results: the system encounters significantly more difficulty in handling large rotations compared to smaller ones. This behavior is expected, as errors tend to compound and grow with increased rotational complexity. Figure 6.10 illustrates this phenomenon, where the general shape of the trajectory is preserved, but the errors result in highly distorted estimated paths.

Readers should exercise caution when interpreting the values in Table 6.2. For example, the "full test" trajectory exhibits a lower RE, which can give the misleading impression that the system performs better under extensive rotational motion than it actually does. As shown in Figure 6.10, the system consistently underestimates rotations. However, since the trajectory involved both positive and negative rotations, these underestimations ended up getting caught up with the true final rotation, reducing the apparent rotational drift. This effect creates the false impression of better performance, even though the system continuously underestimated rotations throughout the trajectory.

[6]Table acronym: Rotational Error (RE)

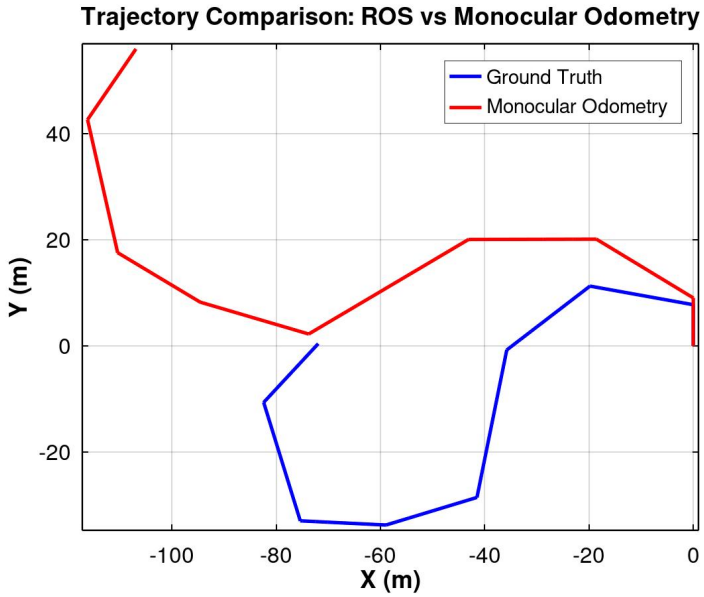
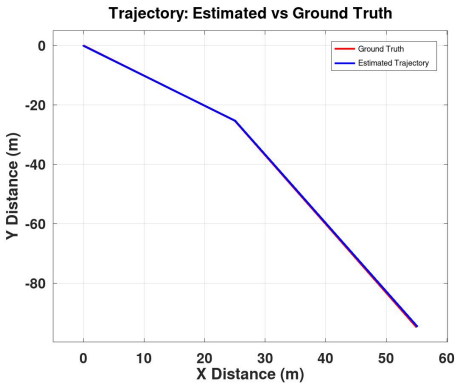


Figure 6.10: Trajectory of the "full test" path

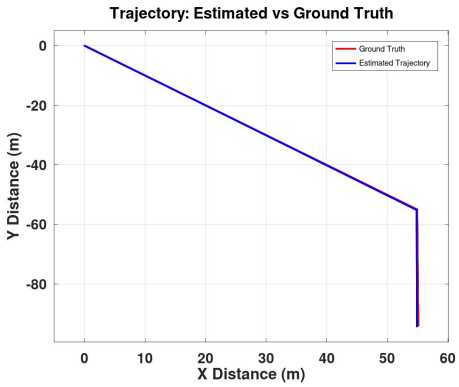
6.3 System Performance in Semi-Autonomous Mode

This section aims to present the performance of the overall system, with a particular focus on the EKF. A variety of trajectories were tested, both with and without stop points. At the end of this section, performance statistics will also be provided.

The first test involved a simple trajectory, without any stop points, with a final target point at $(x, y) = (55, -94)$ m. In this scenario, both Euclidean and Manhattan heuristics were employed. The results of this initial test are illustrated in Figure 6.11.



(a) Fenix's trajectory using Euclidean distance heuristics in the first test

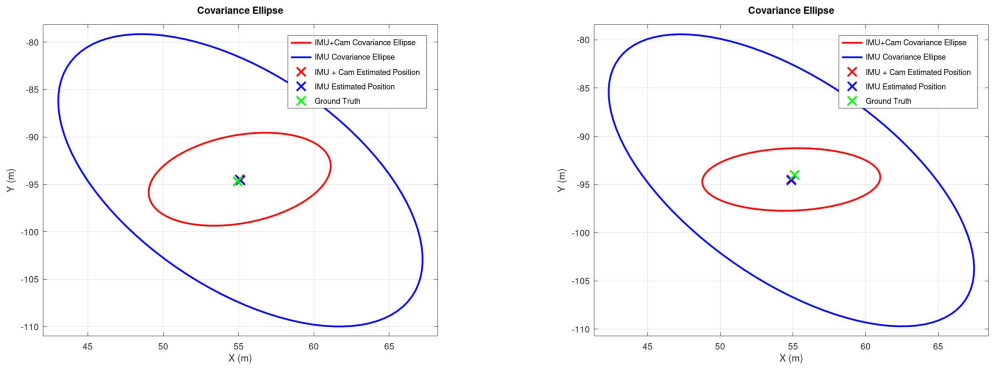


(b) Fenix's trajectory using Manhattan distance heuristics in the first test

Figure 6.11: Comparison of Fenix's trajectory using different distance heuristics in the first test

The primary objective of incorporating the monocular camera, as previously discussed, was to reduce the state covariance, namely the x and y coordinates.

As shown in Figure 6.12, the inclusion of camera data in the sensor fusion process significantly decreases the covariance associated with the x and y position estimates.

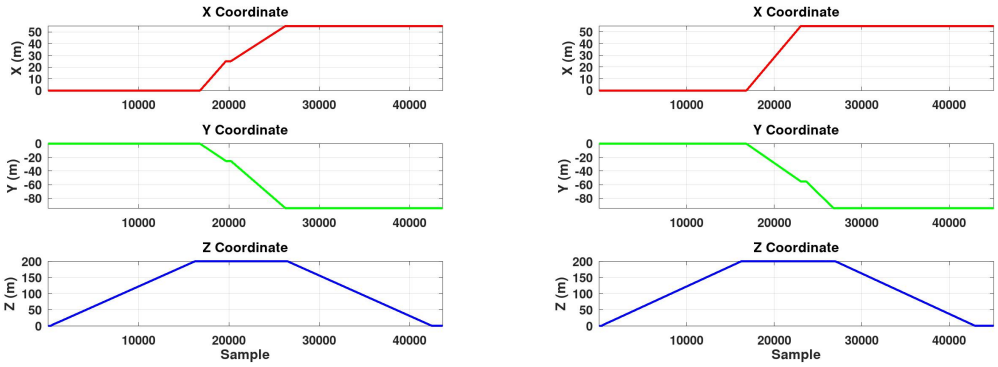


(a) Fenix's covariance reduction using Euclidean distance heuristics in the first test

(b) Fenix's covariance reduction using Manhattan distance heuristics in the first test

Figure 6.12: Comparison of Fenix's trajectory using different distance heuristics in the first test

It is important not to be misled by the slightly lower covariance—shown in red—in Figure 6.12b. As will be demonstrated later, this particular case, which used the Manhattan distance heuristic, happened to include more camera updates, since it also traveled a larger distance. These additional updates naturally contributed to a greater reduction in covariance, as expected. This example highlights that the algorithm is functioning correctly, incorporating camera updates that effectively reduce the covariance. Figure 6.13 presents the trajectory along the x, y, and z axes.



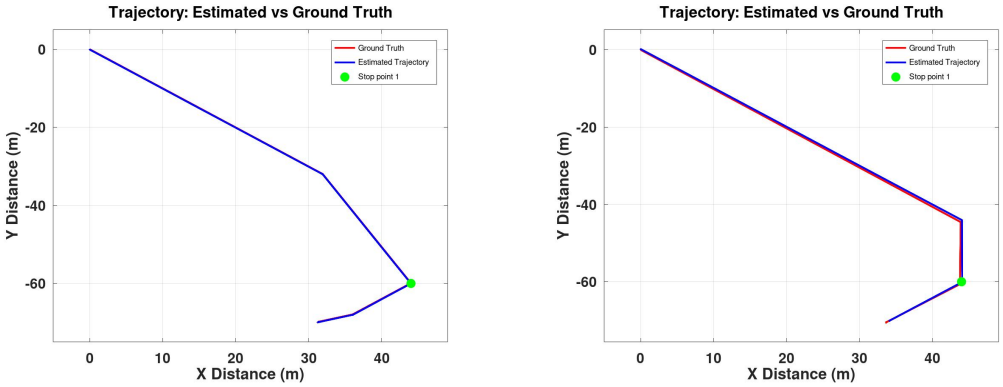
(a) Fenix's progression along the axis using Euclidean distance heuristics in the first test

(b) Fenix's progression along the axis using Manhattan distance heuristics in the first test

Figure 6.13: Comparison of Fenix's progression along the axis using different distance heuristics in the first test

The next test involved a single stop point at $(x, y) = (44, -60)$ m along the trajectory, with the final destination set at $(x, y) = (31, -70)$ m. The altitude, represented by the z coordinate, remained constant at 200 m throughout the entire

trajectory. Figure 6.14 illustrates the trajectory comparison using both heuristic metrics.

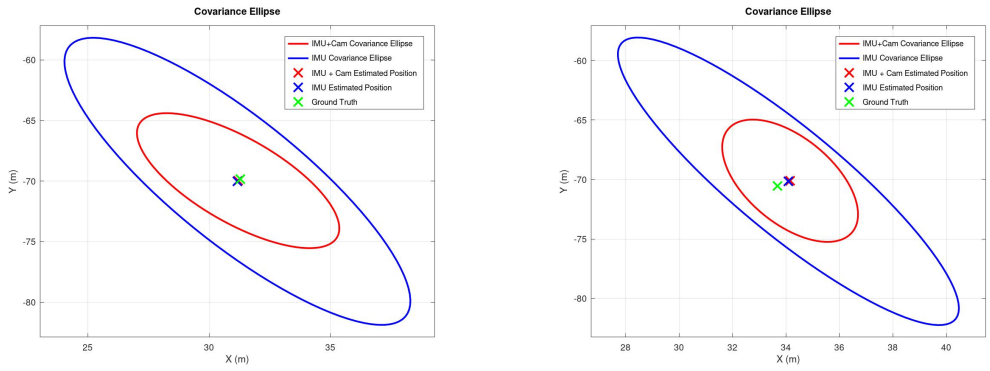


(a) Fenix's trajectory using Euclidean distance heuristics in the second test

(b) Fenix's trajectory using Manhattan distance heuristics in the second test

Figure 6.14: Comparison of Fenix's trajectory using different distance heuristics in the second test

Similar to the first test, the behavior observed in this scenario remained consistent: the system maintained a lower error when using Euclidean distance heuristics. However, the use of Manhattan distance resulted in a greater number of camera updates, which in turn led to slightly smaller covariances, as shown in Figure 6.15. There are a few contributing factors to this behavior. Firstly, trajectories computed using Manhattan distance tend to be longer, which naturally increases the opportunity for camera updates along the path. Secondly, the terrain over which the drone navigates differs between trajectories. Since the paths diverge, the drone is exposed to different visual inputs and environmental conditions, leading to variations in image processing and state estimation outcomes.

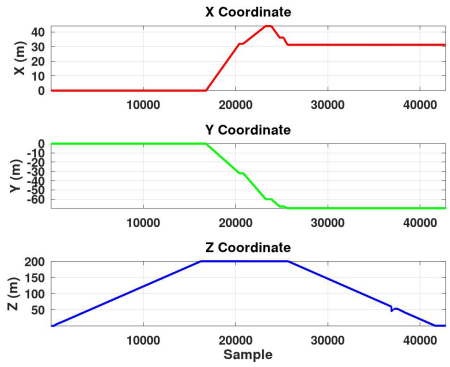


(a) Fenix's covariance reduction using Euclidean distance heuristics in the second test

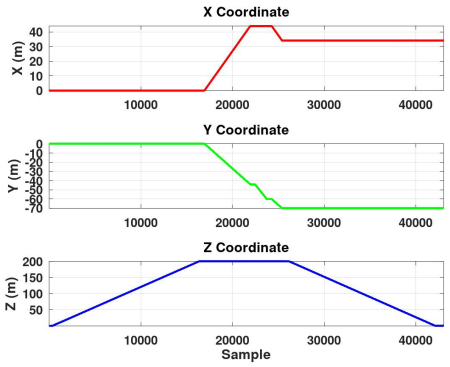
(b) Fenix's covariance reduction using Manhattan distance heuristics in the second test

Figure 6.15: Comparison of Fenix's trajectory using different distance heuristics in the second test

Figure 6.16 illustrates the progression along the axis during the trajectory.



(a) Fenix's progression along the axis using Euclidean distance heuristics in the second test



(b) Fenix's progression along the axis using Manhattan distance heuristics in the second test

Figure 6.16: Comparison of Fenix's progression along the axis using different distance heuristics in the second test

For the next test, only the Euclidean distance heuristic was utilized. The trajectory includes two stop points located at $(-48, -60)$ m and $(-60, -77)$ m, with the final destination at $(-85, -130)$ m. Unlike the previous test, this trajectory involved altitude changes: upon reaching the first stop point, the drone was required to descend to an altitude of 150 m, and to 175 m at the second. Figure 6.17 illustrates the trajectory taken.

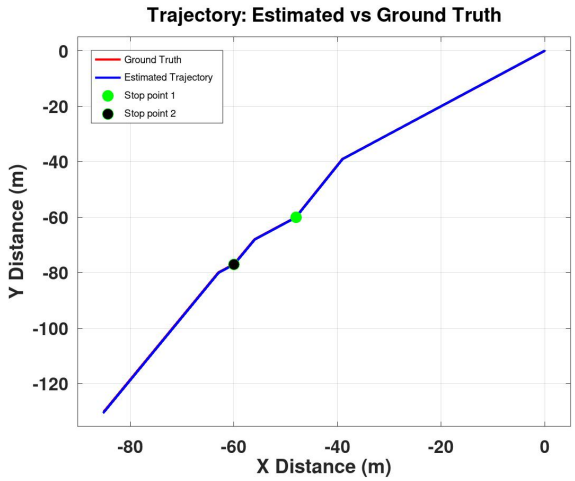


Figure 6.17: Fenix's trajectory using Euclidean distance heuristics for the third test

Following the same pattern observed in previous tests, the covariance, shown in Figure 6.18, was significantly reduced—more so in this particular example. This can be attributed to several factors. First, the trajectory was considerably longer, which, under normal circumstances, would result in greater covariance growth due to error accumulation over time. However, in this case, the more frequent camera updates—both in absolute terms and as a percentage of the trajectory—effectively

counteracted this growth. In other words, the longer the system operates within a single run, the more pronounced the effect of covariance reduction becomes.

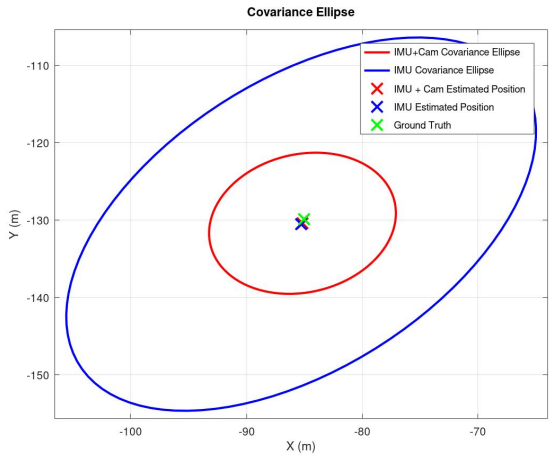


Figure 6.18: Fenix's covariance using Euclidean distance heuristics for the third test

Figure 6.19 illustrates the progression along the axis, where the most notably different result is the change in altitude when reaching the stop points.

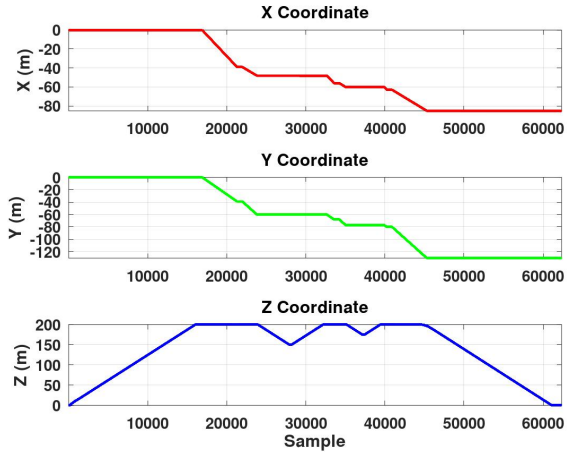


Figure 6.19: Fenix's progression along the axis using Euclidean distance heuristics for the third test

Table 6.3 presents the data pertaining to these tests, with emphasis on both the Euclidean and Manhattan heuristics. It is possible to conclude, based on the similar table presented in the previous section, that the IMU, functioning as the primary source of odometry and being vastly superior in terms of reduced errors, significantly enhanced the system's accuracy. Furthermore, the inclusion of monocular data as a means to reduce covariance made the system considerably more confident in its estimated position, without compromising the overall accuracy or introducing excessive error when compared to using only IMU-based odometry.

Table 6.3: Statistics of the final tests[7]

Heuristic	Path	RMSE (m)	Drift ($m\,m^{-1}$)	ATE (%)	RE (°)	MU (%)	RFC (m)	EFC (m)	GC (m)
Euclidean	Test 1	0.18275	0.00060824	0.16343	2.1695e-05	23.40	(54.94, -94.68)	(55.07, -94.46)	(55,-94)
	Test 2	0.079487	0.00073845	0.086225	2.6749e-06	22.22	(31.29, -69.84)	(31.2, -69.96)	(31,-70)
	Test 3	0.17317	0.0007601	0.10952	6.211e-05	34.25	(-85.03, -129.88)	(-85.17, -130.40)	(-85, -130)
Manhattan	Test 1	0.237	0.0015265	0.20194	3.1276e-05	28.57	(55.12, -94.02)	(54.89, -94.57)	(55,-94)
	Test 2	0.47664	0.0042822	0.51257	0.10941	26.32	(33.68, -70.54)	(34.15, -70.1)	(31,-70)

Ingenuity vs Fenix Comparison

As mentioned before, Ingenuity's worst trajectory caused an error of 5 m from the designated landing site. This trajectory involved the drone ascending to an altitude of 10 m before flying 150 m southwest at a speed of 4 $m\,s^{-1}$. Upon reaching this point, the drone moved an additional 15 m south while capturing images towards the west, followed by a 50 m flight northeast before landing, resulting in a total round trip of 215 m.

The final conducted test aimed to replicate this trajectory using Fenix. Figure 6.20 illustrates the differences between the trajectories of both systems. Naturally, since the exact details of Ingenuity's trajectory are unavailable, its representation remains speculative.

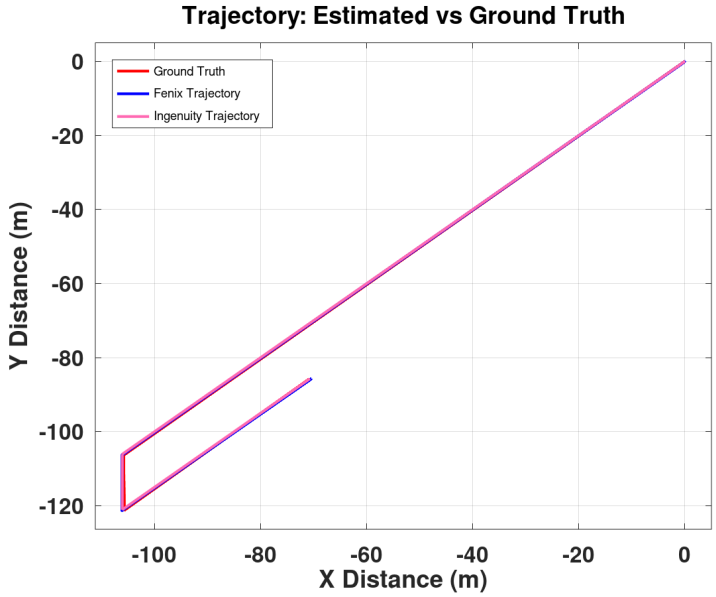


Figure 6.20: Trajectory comparison between Ingenuity and Fenix

As with the previous tests, the fusion with the camera data greatly reduces the covariance, shown in Figure 6.21. Table 6.4 provides the statistical data of this

[7]Table acronyms: Monocular Usage (MU), Real Final Coordinates (RFC), Estimated Final Coordinates (EFC), Goal Coordinates (GC)

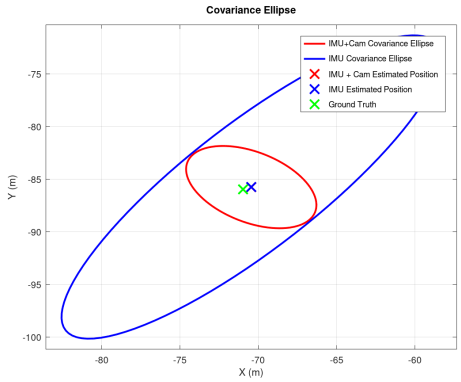


Figure 6.21: Fenix's covariance for the trajectory similar to Ingenuity

flight. Out of curiosity, the system estimated a traveled distance of 215.8121 m, which is a very close estimate to Ingenuity's 215 m.

Table 6.4: Statistics of Fenix's trajectory similar to Ingenuity's

Heuristic	**Path**	**RMSE** (m)	**Drift** ($\mathrm{m\,m^{-1}}$)	**ATE** (%)	**RE** (°)	**MU** (%)	**RFC** (m)	**EFC** (m)	**GC** (m)
Euclidean	Ingenuity Replica	0.24469	0.00090525	0.114	0.00016899	43.68	-85.95, -70.98	-85.73, -70.45	-86, -71

Summary

All components of the system behaved as expected. The A* algorithm effectively determined the shortest path, even with multiple obstacles. While monocular odometry yielded modest results on its own and exhibited the anticipated increase in error over time, the optical flow values consistently provided accurate measurements. These, when incorporated with the accuracy of the IMU, helped contain the otherwise unbounded exponential growth of uncertainty.

The system as a whole is capable of maintaining an accurate position estimate, utilizing an average of $\approx 29.7\%$ of camera updates, while accurately following the trajectories generated by the A* Algorithm.

Chapter 7

Conclusion

Given the objectives outlined in Chapter 1, the system developed throughout this work was deemed successful. The primary goals of designing and validating a Guidance and Navigation system suitable for planetary drone operation—specifically on terrain and under conditions similar to those of Mars—were met.

Through a combination of deliberate system design, integration of multiple sensor modalities, and extensive testing in simulated environments, the system demonstrated strong performance across a variety of scenarios. The ability to semi-autonomously navigate over large distances, adapt to changes in altitude, and maintain accurate estimations of position and trajectory confirms the effectiveness of the implemented methods.

Particularly, the fusion of IMU data with monocular visual odometry proved beneficial: the IMU ensured accurate motion prediction, while the camera measurements significantly reduced the covariance, contributing to greater certainty in Fenix's estimated position. Furthermore, heuristic-based path planning methods, such as those using Euclidean and Manhattan distances, were compared and validated, showing the trade-offs between path optimality and computational efficiency.

The system also met all the designed *a priori* criteria to ensure fidelity with real-life systems. In other words, the following were the requirements, along with their respective justifications regarding whether and how they were fulfilled:

- **S-1:** Although not explicitly shown, the development of this project was conducted with strong adherence to NASA's software guidelines, such as strict

compiler warning checks, as indicated in Appendix A;

- **S-4:** Sensor frequencies used in the system matched or were lower than those seen in previous iterations or anticipated future missions;
- **S-6:** The selection and configuration of sensors were heavily inspired by those outlined in Chapter 4, aiming to mimic realistic mission conditions;
- **S-7:** To support effective monocular navigation, the drone's velocity was limited to below 3 $\mathrm{m\,s^{-1}}$;
- **M-1:** All algorithms were tested within a simulated environment inspired by the Martian region of Valles Marineris;
- **M-2:** Fenix operated completely autonomously following its initialization, requiring no human intervention during the mission;
- **C-1:** No form of SLAM was used, in line with the design constraints;
- **C-2:** Only sensors with proven or proposed applications in planetary missions—namely, the IMU, monocular camera, and LRF—were employed.

The remaining **S-** criteria, while documented, represent higher-level or abstract conditions that could not be explicitly demonstrated or tested within the scope of this document. As such, they are not directly addressed here but were acknowledged during the system's conceptual design phase.

As for the intended tests proposed in Chapter 1, all modules were tested in isolation, each demonstrating stable behavior and satisfactory performance. The complete system was also evaluated in integrated mode, operating semi-autonomously. During these integrated tests, the system consistently outperformed the benchmark set by NASA's Ingenuity, which recorded a maximum deviation of 5 m from the designated landing location.

7.1 Future Work

Given the scope of this thesis, some components were either simplified or omitted entirely. Several improvements have been envisioned, not only as potential continuations to address unresolved aspects but also as enhancements that I personally wished to implement but ultimately did not have the opportunity to pursue.

Developing a fully functional drone model and a physical prototype would be essential for adapting the GNC system to the specific dynamics of the platform. Alongside this, hardware-in-the-loop testing would be fundamental for evaluating the actual system's performance and tailoring the algorithms if necessary. This

approach would provide a more realistic assessment of the system's behavior under operational conditions and enable fine-tuning to enhance performance.

Implementing a landing site selection mechanism would be crucial to ensure that, prior to touchdown, the drone could autonomously identify the safest landing location based on monocular camera imaging.

The feasibility of the Lucas-Kanade method should also be further evaluated, as optical flow remains the primary source of monocular data relevant to the EKF.

Additionally, an interesting avenue for future testing would involve comparing the performance of the system in terms of both CPU load and navigation accuracy when using the EKF versus the UKF. While the EKF remains an industry standard, the UKF often offers superior accuracy. Evaluating both approaches and weighing their respective advantages would provide valuable insights for selecting the most effective method for this specific system.

RRT* could eventually be tested; however, as concluded in [107], it was found to consistently perform worse than A* in terms of both computational efficiency and path optimality. Additionally, the system was initially envisioned to include the capability of loading a pre-generated map of the operational region. Although this feature was considered during the design phase, there was not sufficient time to implement it within the scope of this work.

As mentioned previously, the drone is equipped with path planning capabilities that allow it to avoid obstacles. However, at this stage of development, the drone is not capable of adjusting its altitude dynamically to fly over large-scale terrain features such as canyons. It remains constrained to operate within the canyon walls, and in the event of encountering an obstacle of such dimensions, the system—as implemented in this work—would not be able to respond appropriately. It is important to note that, although beyond the scope of this thesis, the drone was originally envisioned to be equipped with either a stereo or a depth front-facing camera. Such a sensor would serve dual purposes: aiding in the geological, scientific exploration of the canyon walls and enabling the drone to locally detect large-scale obstacles. This would allow it to modify the global path planning dynamically and autonomously.

Other comparisons should also be made, specifically comparing the 8-point algorithm with the 5-point and even the 1-point algorithms.

References

[1] NASA, "Benefits stemming from space exploration." https://www.nasa.gov/wp-content/uploads/2015/01/benefits-stemming-from-space-exploration-2013-tagged.pdf?emrc=ca90d1, Sep. 2013. Accessed: Sep. 23, 2024. [Cited on pages 1 and 2]

[2] NASA, "Benefits from apollo: Giant leaps in technology." https://www.nasa.gov/wp-content/uploads/2019/08/466723main_NasaFacts_508c.pdf?emrc=88336a, 2007. Accessed: Oct. 25, 2024. [Cited on page 1]

[3] NASA, "Nasa technology is all around you." https://www.jpl.nasa.gov/news/nasa-technology-is-all-around-you/, Dec. 2016. Accessed: Sep. 24, 2024. [Cited on page 1]

[4] NASA, "Image sensors enhance camera technologies." https://spinoff.nasa.gov/Spinoff2010/cg_3.html, 2010. Accessed: Sep. 24, 2024. [Cited on page 1]

[5] NASA, "Humans to mars." https://www.nasa.gov/humans-in-space/humans-to-mars/, Sep. 2023. Accessed: Sep. 24, 2024. [Cited on page 2]

[6] ESA, "Why go to mars?." https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/Exploration/Why_go_to_Mars, 2019. Accessed: Sep. 24, 2024. [Cited on pages 2 and 26]

[7] J. Rauf, "Spacex 8 musk's mars vision, plans, challenges and summary." https://www.uc.edu/content/dam/refresh/cont-ed-62/olli/fall-23-class-handouts/SpaceX%208%20%20Mars%20%20Vision%20Summary.pdf, 2023. Accessed: Oct. 31, 2024. [Cited on page 2]

[8] SpaceX, "Making humans a multiplanetary species." https://www.youtube.com/watch?v=H7Uyfqi_TE8, Sep. 2016. Accessed: Sep. 24, 2024. [Cited on page 2]

[9] SpaceX, "Spacex." https://www.spacex.com/humanspaceflight/mars/. Accessed: Sep. 24, 2024. [Cited on page 2]

[10] NASA, "Nasa's perseverance rover to begin long climb up martian crater rim." https://www.nasa.gov/

missions/mars-2020-perseverance/perseverance-rover/nasas-perseverance-rover-to-begin-long-climb-up-martian-crater-rim/, Aug. 2024. Accessed: Sep. 24, 2024. [Cited on page 3]

[11] Jet Propulsion Laboratory, "Duaxel." https://www.jpl.nasa.gov/robotics-at-jpl/duaxel/, 2021. Accessed: Sep. 25, 2024. [Cited on page 3]

[12] NASA Jet Propulsion Laboratory, "Duaxel: A nasa prototype rover to explore the toughest terrain." https://www.youtube.com/watch?v=GUNWVroyys4, Oct. 2020. Accessed: Sep. 24, 2024. [Cited on page 3]

[13] NASA, "Perseverance rover components." https://science.nasa.gov/mission/mars-2020-perseverance/rover-components/, 2020. Accessed: Sep. 25, 2024. [Cited on pages 3, 29, 33, and 34]

[14] NASA, "Ingenuity mars helicopter landing press kit." https://www.jpl.nasa.gov/news/press_kits/mars_2020/download/ingenuity_landing_press_kit.pdf, Jan. 2021. Accessed: Sep. 26, 2024. [Cited on pages 4 and 30]

[15] NASA, "Surviving an in-flight anomaly: What happened on ingenuity's sixth flight." https://science.nasa.gov/blog/surviving-an-in-flight-anomaly-what-happened-on-ingenuitys-sixth-flight/?utm_source=chatgpt.com, May 2021. Accessed: May 09, 2025. [Cited on page 5]

[16] A. Isser and DSIAC, "Introduction to guidance, navigation, and control (gnc)." https://dsiac.org/wp-content/uploads/2022/10/AD1182620.pdf, Jul. 2021. Accessed: Oct. 05, 2024. [Cited on page 9]

[17] P. T. Kabamba and A. Girard, *Fundamentals of Aerospace Navigation and Guidance*. New York: Cambridge University Press, 2014. [Cited on page 9]

[18] M. B. Quadrelli, L. J. Wood, J. E. Riedel, M. C. McHenry, M. Aung, L. A. Cangahuala, R. A. Volpe, P. M. Beauchamp, and J. A. Cutts, "Guidance, navigation, and control technology assessment for future planetary science missions," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 7, pp. 1165–1186, 2015. [Cited on page 9]

[19] Y. Zhu, "Overview of robot perception." https://www.cs.utexas.edu/~yukez/cs391r_fall2020/slides/lecture_robot_perception.pdf, 2020. Accessed: Oct. 05, 2024. [Cited on page 10]

[20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2010. [Cited on pages 10, 11, 12, and 13]

[21] W. Elmenreich and R. Leidenfrost, "Fusion of heterogeneous sensors data," in *Proceedings of the WISES*, pp. 1–10, CiteSeer X (The Pennsylvania State University), Jul. 2008. [Cited on page 11]

[22] D. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE Pervasive Computing*, vol. 2, pp. 24–33, Jul. 2003. [Cited on page 11]

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec 1959. [Cited on page 14]

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, Massachusetts; London: MIT Press, 2007. Also distributed by Boston [etc.]. [Cited on page 14]

[25] E. Ebeid, R. H. Jacobsen, P. Guzzini, G. v. Bögel, P. Schneider-Kamp, N. Luminari, C. Casarotti, M. Mandirola, D. Taurino, and A. d. Sole, "Autonomous drones to ensure safety in transport: Concept and implementations," 2022. [Cited on page 17]

[26] Fly4Future, "Koldro - universal collaborative drone system." `https://fly4future.com/koldro/`, Apr. 2023. Accessed: Oct. 08, 2024. [Cited on page 17]

[27] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," in *Systems, Man and Cybernetics*, Oct 2005. [Cited on page 18]

[28] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007. [Cited on pages 18 and 19]

[29] S. Daftry *et al.*, "Lunarnav: Crater-based localization for long-range autonomous lunar rover navigation," *arXiv*, Mar 2023. [Cited on page 19]

[30] L. Matthies *et al.*, "Lunar rover localization using craters as landmarks." `https://ieeexplore.ieee.org/abstract/document/9843714`, Mar. 2022. Accessed: Apr. 25, 2023. [Cited on page 19]

[31] T. Fong, "Digital proving ground: Viper rover simulator (rsim)." `https://ntrs.nasa.gov/citations/20240014021`, Nov. 2024. Accessed: Jun. 01, 2025. [Cited on page 19]

[32] Blender Foundation, "Home of the blender project - free and open 3d creation software." `https://www.blender.org/`, 2019. [Cited on page 19]

[33] R. D. Prete and A. Renga, "A novel visual-based terrain relative navigation system for planetary applications based on mask r-cnn and projective invariants," *Aerotecnica Missili & Spazio*, vol. 101, pp. 335–349, Oct. 2022. [Cited on page 19]

[34] Y. Chen, H. Inaltekin, and M. Gorlatova, "AdaptSLAM: Edge-assisted adaptive slam with resource constraints via uncertainty minimization," in *Proc. IEEE INFOCOM*, 2023. [Cited on page 20]

[35] C. Campos, R. Elvira, J. J. Gómez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 1874–1890, 2021. [Cited on page 21]

[36] V. Niculescu, T. Polonelli, M. Magno, and L. Benini, "Nanoslam: Enabling fully onboard slam for tiny robots," *IEEE Internet of Things Journal*, vol. 11, no. 8, pp. 13584–13607, 2024. [Cited on pages 21 and 22]

[37] GreenWaves Technologies, "Key application areas easy development." `https://greenwaves-technologies.com/wp-content/uploads/2024/05/Product-Brief-GAP9-Sensors-General-V1_15.pdf`, 2021. Accessed: Nov. 03, 2024. [Cited on page 21]

[38] STMicroelectronics, "Vl53l5cx - stmicroelectronics." `https://www.st.com/en/imaging-and-photonics-solutions/vl53l5cx.html`, 2022. Accessed: Nov. 03, 2024. [Cited on page 21]

[39] Bitcraze, "Crazyflie 2.1 | bitcraze." `https://www.bitcraze.io/products/old-products/crazyflie-2-1/`, 2024. Accessed: Nov. 03, 2024. [Cited on page 21]

[40] H. Andreasson, T. Duckett, and A. Lilienthal, "Mini-slam: Minimalistic visual slam in large-scale environments based on a new interpretation of image similarity," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 4096–4101, 2007. Accessed: Nov. 12, 2024. [Cited on page 22]

[41] NASA and G. J. Holzmann, "The power of 10: Rules for developing safety-critical code." `https://spinroot.com/gerard/pdf/P10.pdf`, 2006. Accessed: 2024-10-04. [Cited on page 26]

[42] NASA, "Mars 2020: Perseverance rover." `https://science.nasa.gov/mission/mars-2020-perseverance/`, 2020. Accessed: Oct. 25, 2024. [Cited on page 26]

[43] NASA, "Insight lander." `https://science.nasa.gov/mission/insight/`. Accessed: Oct. 25, 2024. [Cited on page 26]

[44] CNSA, "Tianwen-1: China successfully launches probe in first mars mission." `https://www.cnsa.gov.cn/english/n6465652/n6465653/c6809882/content.html`, 2020. Accessed: Oct. 25, 2024. [Cited on page 26]

[45] ESA, "Exomars trace gas orbiter maps water-rich region of valles marineris." `https://www.esa.int/ESA_Multimedia/Images/2021/12/ExoMars_Trace_Gas_Orbiter_maps_water-rich_region_of_Valles_Marineris`, Dec. 2021. Accessed: 2024-10-01. [Cited on page 26]

[46] NASA, "Valles marineris: The grand canyon of mars." `https://www.nasa.gov/image-article/valles-marineris-grand-canyon-of-mars`, Mar. 2008. Accessed: Oct. 1, 2024. [Cited on page 26]

[47] A. Yin, "Structural analysis of the valles marineris fault zone: Possible evidence for large-scale strike-slip faulting on mars," *Lithosphere*, vol. 4, pp. 286–330, Jun. 2012. [Cited on page 26]

[48] J. Davis, P. Grindrod, S. Banham, N. Warner, S. Conway, S. Boazman, and S. Gupta, "A record of syn-tectonic sedimentation revealed by perched alluvial fan deposits in valles marineris, mars," *Geology*, vol. 49, Jun. 2021. [Cited on page 27]

[49] ESA, "Esa science & technology - the ages of mars." `https://sci.esa.int/web/mars-express/-/55481-the-ages-of-mars`, 2013. Accessed: Oct. 1, 2024. [Cited on page 27]

[50] A. McEwen, M. Chojnacki, H. Miyamoto, R. Hemmi, C. Weitz, R. Williams, C. Quantin, J. Flahaut, J. Wray, S. Turner, J. Bridges, S. Grebby, C. Leung, and S. Rafkin, "Landing site and exploration zone in eastern melas chasma," *Lunar and Planetary Science Conference*, vol. 1879, p. 1007, Oct. 2015. [Cited on page 27]

[51] A. McEwen, M. Chojnacki, H. Miyamoto, R. Hemmi, C. Weitz, R. Williams, C. Quantin, J. Flahaut, J. Wray, S. Turner, J. Bridges, S. Grebby, C. Leung, and S. Rafkin, "East melas chasma exploration zone: First landing site/exploration zone workshop for human missions to the surface of mars esp_034962_1670, recurring slope lineae in melas chasma." `https://www.nasa.gov/wp-content/uploads/2015/11/e-melas-ez.pdf`, 2015. Accessed: Oct. 1, 2024. [Cited on page 27]

[52] A. Mojarro, G. Ruvkun, M. T. Zuber, and C. E. Carr, "Human exploration of mars at valles marineris: The past, present, and future of life on mars," *Lunar*

and Planetary Science Conference, vol. 1879, p. 1036, Oct. 2015. [Cited on page 27]

[53] A. A. Fraeman, W. Rapin, J. Bapst, L. Matthies, B. L. Ehlmann, B. Langlais, R. Lillis, A. Mittelholz, B. Weiss, C. Quantin-Nataf, J. Flahaut, M. Golombek, K. L. Siebach, V. Payré, A. Udry, M. G. A. Lapôtre, J. Espley, R. O. Green, P. Sullivan, D. R. Thompson, and K. Sneeder, "A mars science helicopter mission to valles marineris: Unlocking clues to planetary formation and early evolution," *Journal of Planetary Science*, 2024. [Cited on page 27]

[54] H. Miyamoto, G. Komatsu, A. McEwen, M. Chojnacki, T. Usui, and A. Yamagishi, "Candidate landing sites in valles marineris: Ancient and modern habitability." `https://marsnext.jpl.nasa.gov/workshops/2014_05/20_VallesMarineris_Miyamoto.pdf`, 2014. Accessed: Oct 1, 2024. [Cited on page 27]

[55] T. Ormston, "Time delay between mars and earth." `https://blogs.esa.int/mex/2012/08/05/time-delay-between-mars-and-earth/`, Aug. 2012. Accessed: Oct. 10, 2024. [Cited on page 28]

[56] ESA, "Close-up of valles marineris from global mars colour mosaic (annotated)." `https://www.esa.int/ESA_Multimedia/Images/2023/05/Close-up_of_Valles_Marineris_from_global_Mars_colour_mosaic_annotated`, May. 2023. Accessed: 2024-10-02. [Cited on page 29]

[57] NASA, "Opportunity - mars exploration rover opportunity (mer-b)." `https://science.nasa.gov/mission/mer-opportunity/`, 2024. Accessed: Oct. 3, 2024. [Cited on page 29]

[58] NASA, "Mars perseverance press kit." `https://www.jpl.nasa.gov/news/press_kits/mars_2020/download/mars_2020_landing_press_kit.pdf`, 2021. Accessed: Oct. 25, 2024. [Cited on page 29]

[59] B. Balaram, T. Canham, C. Duncan, H. Grip, W. Johnson, J. Maki, A. Quon, R. Stern, and D. Zhu, "Mars helicopter technology demonstrator," Jan. 2018. [Cited on pages 29, 31, 33, and 34]

[60] R. Lorenz, E. Turtle, J. Barnes, M. Trainer, D. Adams, K. Hibbard, C. Sheldon, K. Zacny, P. Peplowski, D. Lawrence, M. Ravine, T. McGee, K. Sotzen, S. MacKenzie, J. Langelaan, S. Schmitz, L. Wolfarth, and P. Bedini, "Dragonfly: A rotorcraft lander concept for scientific exploration at titan," *Johns Hopkins APL Technical Digest (Applied Physics Laboratory)*, vol. 34, pp. 374–387, 10 2018. [Cited on page 29]

[61] J. N. Maki, D. Gruel, C. McKinney, M. A. Ravine, M. Morales, D. Lee, R. Willson, D. Copley-Woods, M. Valvo, T. Goodsall, J. McGuire, R. G. Sellar, J. A. Schaffner, M. A. Caplinger, J. M. Shamah, A. E. Johnson, H. Ansari, K. Singh, T. Litwin, R. Deen, A. Culver, N. Ruoff, D. Petrizzo, D. Kessler, C. Basset, T. Estlin, F. Alibay, A. Nelessen, and S. Algermissen, "The mars 2020 engineering cameras and microphone on the perseverance rover: A next-generation imaging system for mars exploration," *Space Science Reviews*, vol. 216, Nov. 2020. [Cited on pages 29, 30, and 34]

[62] Intel, "New views of mars thanks to intel tech." `https://www.intel.com/content/www/us/en/newsroom/news/new-views-mars-thanks-intel-tech.html`, Apr. 2021. Accessed: Sep. 29, 2024. [Cited on page 30]

[63] M. McHenry, N. Abcouwer, J. Biesiadecki, J. Chang, T. D. Sesto, A. Johnson, T. Litwin, M. Maimone, J. Morrison, R. Rieber, O. Toupet, and P. Twu, "Mars 2020 autonomous rover navigation." `https://www-robotics.jpl.nasa.gov/media/documents/AAS_2020_mobility_mmc_v10.pdf`, 2020. [Cited on pages 30 and 34]

[64] V. Verma, F. Hartman, A. Rankin, M. Maimone, T. Del Sesto, O. Toupet, E. Graser, S. Myint, K. Davis, D. Klein, J. Koch, S. Brooks, P. Bailey, H. Justice, M. Dolci, and H. Ono, "First 210 solar days of mars 2020 perseverance robotic operations - mobility, robotic arm, sampling, and helicopter," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–20, 2022. [Cited on pages 30 and 34]

[65] Garmin, "Lidar-lite v3." `https://www.garmin.com/en-US/p/557294`, 2018. Accessed: Oct. 2, 2024. [Cited on page 31]

[66] H. F. Grip, J. Lam, D. S. Bayard, D. T. Conway, G. Singh, R. Brockers, J. H. Delaune, L. H. Matthies, C. Malpica, T. L. Brown, A. Jain, A. M. S. Martin, and G. B. Merewether, *Flight Control System for NASA's Mars Helicopter*. [Cited on pages 31, 34, 57, 58, and 59]

[67] H. F. Grip, D. Conway, J. Lam, N. Williams, M. P. Golombek, R. Brockers, M. Mischna, and M. R. Cacan, "Flying a helicopter on mars: How ingenuity's flights were planned, executed, and analyzed," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–17, 2022. [Cited on page 31]

[68] K. Hinum, "Qualcomm adreno 330." `https://www.notebookcheck.net/Qualcomm-Adreno-330.110714.0.html`, 2014. Accessed: Oct. 4, 2024. [Cited on page 31]

[69] A. Cameron, "Inside the ingenuity helicopter: Teamwork on mars." https://insideunmannedsystems.com/inside-the-ingenuity-helicopter-teamwork-on-mars/, Jun. 25 2021. Accessed: Mar. 20, 2025. [Cited on page 31]

[70] B. Schilling, T. G. McGee, R. Mitch, and R. Watson, "Preliminary design of the dragonfly navigation filter." https://arxiv.org/abs/2307.13513, 2023. [Cited on pages 32, 33, and 34]

[71] T. G. McGee, D. Adams, K. Hibbard, E. Turtle, R. Lorenz, F. Amzajerdian, and J. Langelaan, *Guidance, Navigation, and Control for Exploration of Titan with the Dragonfly Rotorcraft Lander*. 2018. [Cited on pages 32 and 33]

[72] NASA, "3d lidar for autonomous landing site selection." https://technology.nasa.gov/patent/GSC-TOPS-353, 2024. Accessed: Oct. 08, 2024. [Cited on pages 32 and 33]

[73] "Askscience ama series: We're the team sending nasa's dragonfly drone mission to saturn's moon titan. ask us anything!." https://www.reddit.com/r/askscience/comments/c7r6a0/askscience_ama_series_were_the_team_sending_nasas/, 2019. Accessed: Sep. 29, 2024. [Cited on pages 33 and 34]

[74] A. Shaukat, S. Al-Milli, A. Bajpai, C. Spiteri, G. Burroughes, Y. Gao, D. Lachat, and M. Winter, "Next-generation rover gnc architectures," May. 2015. [Cited on page 33]

[75] BAE Systems, "Radiation-hardened electronics product guide." https://www.baesystems.com/en-media/uploadFile/20211206201500/1434554723601.pdf, 2022. Accessed: Oct. 1, 2024. [Cited on page 33]

[76] Nanoreview, "Qualcomm snapdragon 801: Specs and benchmarks." https://nanoreview.net/en/soc/qualcomm-snapdragon-801, 2020. Accessed: Sep. 29, 2024. [Cited on page 33]

[77] M. Vecilla, *Autonomous Navigation of Planetary Rovers*. PhD thesis, Universidad de Málaga, 2021. [Cited on page 34]

[78] NASA, "Dragonfly." https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=DRAGONFLY, Oct. 2022. Accessed: Oct. 2, 2024. [Cited on page 34]

[79] Northrop Grumman, "Navigating mars and beyond with ln-200 inertial measurement units." https://www.northropgrumman.com/space/navigating-mars-and-beyond-with-ln-200-inertial-measurement-units, 2021. Accessed: Sep. 29, 2024. [Cited on page 34]

[80] NASA, "Flight log by the numbers." `https://science.nasa.gov/mission/mars-2020-perseverance/ingenuity-mars-helicopter/`, Jun. 2024. Accessed: Oct. 10, 2024. [Cited on page 34]

[81] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Cited on page 42]

[82] H.-Y. Hsueh, A.-I. Toma, H. A. Jaafar, E. Stow, R. Murai, P. H. J. Kelly, and S. Saeedi, "Systematic comparison of path planning algorithms using pathbench," 2022. [Cited on page 42]

[83] M. Kara, "Evaluation of popular path planning algorithms," *International Journal of Electronics and Telecommunications*, pp. 13–13, 03 2024. [Cited on page 42]

[84] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011. [Cited on page 43]

[85] T. U. of Auckland, "Image filtering." `https://www.cs.auckland.ac.nz/courss/compsci373s1c/PatricesLectures/Image%20Filtering.pdf`, 2023. Accessed: Dec. 18, 2024. [Cited on pages 49 and 50]

[86] Erwin and D. R. Ningsih, "Improving retinal image quality using the contrast stretching, histogram equalization, and clahe methods with median filters," *International Journal of Image, Graphics and Signal Processing*, vol. 12, pp. 30–41, Apr 2020. [Cited on page 54]

[87] L. W. Kheng, "Feature detection and matching." `https://www.comp.nus.edu.sg/~cs4243/lecture/feature.pdf`, May. 2014. Accessed: Dec. 14, 2024. [Cited on page 56]

[88] S. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," 03 2018. [Cited on page 57]

[89] R. Rojas, "Lucas-kanade in a nutshell." `http://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/Lucas-Kanade2.pdf`, 2015. Accessed: Jan. 02, 2025. [Cited on page 58]

[90] A. Rajwade, "Optical flow cs 763, ajit rajwade contents." `https://www.cse.iitb.ac.in/~ajitvr/CS763_Spring2017/OpticalFlow.pdf`, 2024. Accessed: Jan. 02, 2025. [Cited on page 58]

[91] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," vol. 3951, 07 2006. [Cited on pages 58 and 59]

[92] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," vol. 6314, pp. 778–792, 09 2010. [Cited on page 61]

[93] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011. [Cited on pages 61 and 63]

[94] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981. [Cited on page 64]

[95] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. [Cited on page 64]

[96] K. G. Derpanis, "Overview of the ransac algorithm," 2005. Accessed: Jan. 1, 2025. [Cited on page 64]

[97] D. Scaramuzza, "Tutorial on visual odometry." `https://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf`, 2015. Accessed: Jan. 09, 2025. [Cited on page 64]

[98] C. Stachniss, "Photogrammetry & robotics lab direct solutions for computing fundamental and essential matrix." `https://www.ipb.uni-bonn.de/html/teaching/msr2-2020/sse2-18-fe-direct.pdf`, 2023. Accessed: Jan. 02, 2025. [Cited on pages 65 and 66]

[99] K. Hata and S. Savarese, "Cs231a course notes 3: Epipolar geometry." `https://web.stanford.edu/class/cs231a/course_notes/03-epipolar-geometry.pdf`, 2017. Accessed: Jan. 2, 2025. [Cited on pages 65 and 66]

[100] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003. [Cited on pages 65, 66, and 67]

[101] A. Singh and K. Venkatesh, "Monocular visual odometry." `https://avisingh599.github.io/assets/ugp2-report.pdf`, 2015. Accessed: Jan. 03, 2025. [Cited on page 67]

[102] S. Wang, M. O'toole, K. Kitani, L. Lazebnik, D. Hoeim, and S. Fidler, "Sensors ii: Cameras." `https://shenlong.web.illinois.edu/teaching/cs598fall21/assets/slides/lecture4_cameras.pdf`, 2024. Accessed: Jan. 10, 2025. [Cited on page 68]

[103] G. Roth, "Homography." `https://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/homography.pdf`, 2011. Accessed: Jan. 21, 2025. [Cited on page 69]

[104] N. Snavely, "Lecture 13 homographies and ransac." `http://6.869.csail.mit.edu/fa12/lectures/lecture13ransac/lecture13ransac.pdf`, 2006. Accessed: Jan. 21, 2025. [Cited on page 69]

[105] E. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pp. 153–158, 2000. [Cited on page 70]

[106] Northrop Grumman, "LN-200 Fiber Optic Inertial Measurement Unit," 2000. Accessed: Mar. 13, 2025. [Cited on page 70]

[107] J. Braun, T. Brito, J. Lima, P. Costa, P. Costa, and A. Nakano, "A comparison of a* and rrt* algorithms with dynamic and real time constraint scenarios for mobile robots," pp. 398–405, 01 2019. [Cited on page 95]

Appendix A

Compiler options

Table A.1: Compiler warning flags and descriptions

Warning Flag	**Description**
`-Wall`	Enable most common warnings.
`-Wextra`	Enable extra warnings.
`-pedantic`	Strict ANSI compliance.
`-Werror`	Treat warnings as errors.
`-Wconversion`	Warn on implicit type conversions.
`-Wcast-qual`	Warn when casting away const/volatile.
`-Wnon-virtual-dtor`	Warn if class with virtual functions has no virtual destructor.
`-Wpointer-arith`	Warn about invalid pointer arithmetic.
`-Wmissing-declarations`	Warn if a function is declared but not defined.
`-Wredundant-decls`	Warn about redundant declarations.
`-Wvla`	Warn about variable-length arrays.
`-Winline`	Warn if an inline function is not inlined.
`-Wformat`	Warn about format string issues.
`-Wunreachable-code`	Warn about unreachable code.
`-Wlogical-op`	Warn about confusing logical operators.
`-Wmaybe-uninitialized`	Warn about variables that may be uninitialized.
`-Wswitch-default`	Warn if switch lacks a default case.
`-Wswitch-enum`	Warn if switch doesn't cover all enum values.
`-Wduplicated-branches`	Warn about duplicated conditional branches.
`-Wmissing-attributes`	Warn if function attributes are missing.
`-Wstrict-aliasing`	Warn if strict aliasing rules are violated.
`-Wcast-align`	Warn when pointer casting may cause alignment issues.
`-Wctor-dtor-privacy`	Warn about private constructors/destructors in virtual classes.
`-Wdisabled-optimization`	Warn if optimization is disabled.
`-Winit-self`	Warn if a variable is initialized with itself.
`-Wnoexcept`	Warn about noexcept mismatch.
`-Woverloaded-virtual`	Warn about overloaded virtual functions.
`-Wstrict-null-sentinel`	Warn about null pointer as sentinel.
`-Wstrict-overflow=5`	Warn about undefined overflow with strictness level 5.
`-Wundef`	Warn about undefined preprocessor macros.
`-Wmissing-format-attribute`	Warn if function lacks format attribute.
`-Wstrict-aliasing=3`	Ensure strict aliasing rules (level 3 strictness).
`-Wunsafe-loop-optimizations`	Warn about unsafe loop optimizations.
`-Walloca`	Warn about use of `alloca`, which can be unsafe.
`-Wlogical-not-parentheses`	Warn if logical NOT causes ambiguity.
`-Wuseless-cast`	Warn about casts that do not change the type.
`-Wsuggest-override`	Suggest adding 'override' to overridden virtual functions.
`-Walloc-zero`	Warn on allocating zero-sized objects.
`-Wzero-as-null-pointer-constant`	Warn about using `0` instead of `nullptr`.
`-Wnon-template-friend`	Warn about non-template friend declarations.
`-Wdeprecated-declarations`	Warn about deprecated declarations.
`-Wconditionally-supported`	Warn about conditionally supported features.
`-Wodr`	Warn about One Definition Rule violations.

Appendix B

Mission file verification

To ensure that the mission is executed correctly, several steps are taken to verify that the mission file is properly read and populated. Figure B.1 illustrates the pipeline associated with this process.

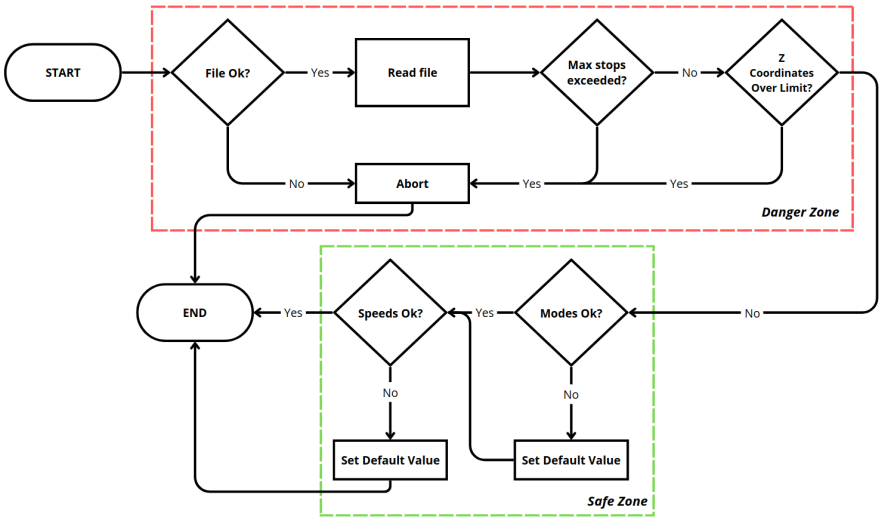


Figure B.1: Mission file verification pipeline

The core principle behind the algorithm is to proceed with the mission even if non-critical errors are detected. The "Danger Zone" refers to errors that could lead to critical failures during the mission. The z coordinate was classified as dangerous because, although the drone operates autonomously, it currently lacks the capability

to detect and avoid obstacles in real-time. Consequently, it was essential to ensure that, in the event of an invalid or missing z value, the system does not assign a default value. Instead, it must abort the mission entirely by issuing an error signal to the system, prioritizing safety above all. The "Safe Zone" encompasses issues that, when corrected, are unlikely to result in significant system failures. For instance, if the traverse speed exceeds the maximum allowable threshold, it is automatically reset to a default safe value of 2 $\mathrm{m\,s^{-1}}$. Similarly, if the mode value is numerically invalid or missing, a default operating mode of "Hover" is applied (refer to Table 5.2 for more details).

Appendix C

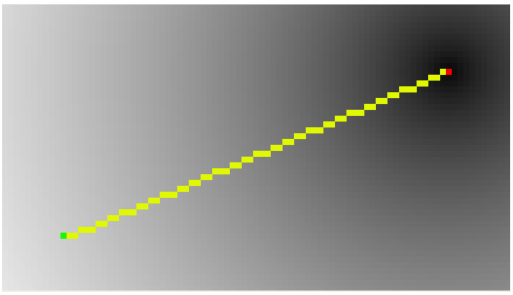
Euclidean vs Manhattan distance for heuristics

It was important to assess whether there was any notable advantage in using Euclidean or Manhattan distances for the A* heuristic. Table C.1 presents the results for trajectories with no obstacles and double resolution. Each row represents a distinct path or set of waypoints, with the "Path" column detailing the sequence of coordinates (x, y) that define the trajectory. Multiple points indicate intermediate stop points along the path, in order.

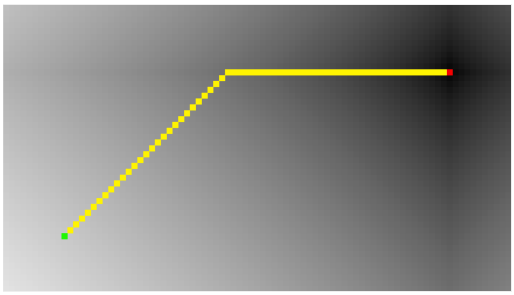
Table C.1: Metrics comparison between Euclidean and Manhattan heuristics

Heuristic	**Path**	**Time** (s)	**Path Length** (px)	**Visited Nodes (per segment)**
Euclidean	33,14	0.5678	66	1300
	5,20; 33,14	0.15	97	375; 671
	12,10; 15,7; 14,-20; 30,-40	0.0592	127	98; 7; 148; 321
Manhattan	33,14	0.0187	66	67
	5,20; 33,14	0.007	97	41; 57
	12,10; 15,7; 14,-20; 30,-40	0.0093	127	25; 7; 55; 41

Considering the results, it is clear that the Manhattan distance provides much faster results while searching fewer nodes while obtaining the same path length. Figure C.1 showcases, visually, the difference between these two methods.



(a) Trajectory using Euclidean distance heuristics



(b) Trajectory using Manhattan distance heuristics

Figure C.1: Trajectory comparison between Euclidean and Manhattan heuristics for a final point at $(x, y) = (33, 14)$ with double precision mapping

It is evident that the Manhattan distance favors straight-line movements, while the Euclidean distance allows for smoother diagonal transitions. This implies that, as stated before, even though the same number of steps toward the goal is achieved, the path generated using Manhattan distance actually covers a greater physical distance. This is because Manhattan distances place greater weight on diagonal movements, making them more costly, which causes the algorithm to prefer following straight-line paths. Notice the distinct gradients near the final point in both images. The Euclidean image exhibits a much smoother transition, whereas the Manhattan image shows a much sharper gradient along the diagonals.

Appendix D

Euclidean distance between features

The minimum distance between features has a significant impact not only on the number of features detected but also on their relative quality. Setting the threshold too low can lead to clusters with dependencies, causing RANSAC to produce unreliable solutions. On the other hand, setting the threshold too high greatly reduces the number of features detected. These extremes are illustrated in Figure D.1.

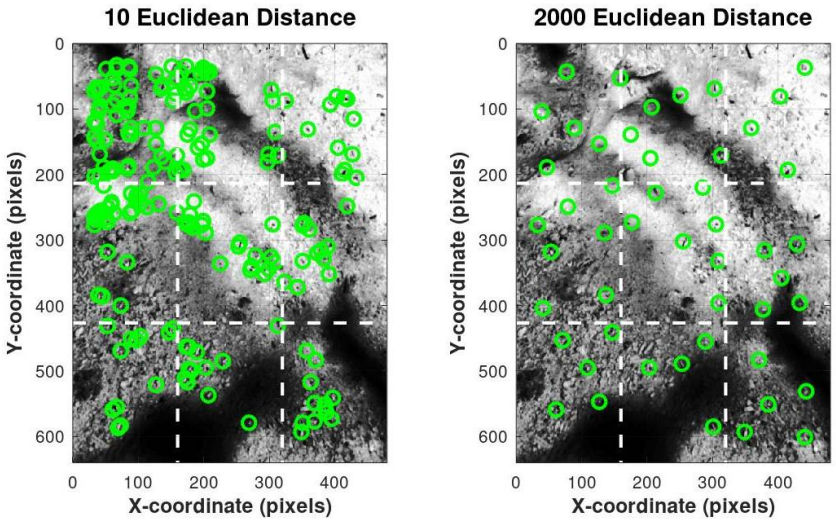


Figure D.1: Comparison of 10 and 2000 minimum Euclidean distance thresholds: 191 features detected with 10, and 49 with 2000

It is worth mentioning that the Euclidean distances depicted here are squared, as they were not subjected to the square root operation. This approach is more computationally efficient, making it better suited for systems with limited resources.

It was found that values greater than 75 and less than 1000, *i.e*, greater than 9 but less than 32 px, were acceptable. Figure D.2 shows that while issues may still arise with a threshold of 75, particularly due to colinearities, these problems are less pronounced compared to the previous example. A threshold of 1000 results in features that are highly spread out, potentially leading to too few matches for the system to perform optimally.

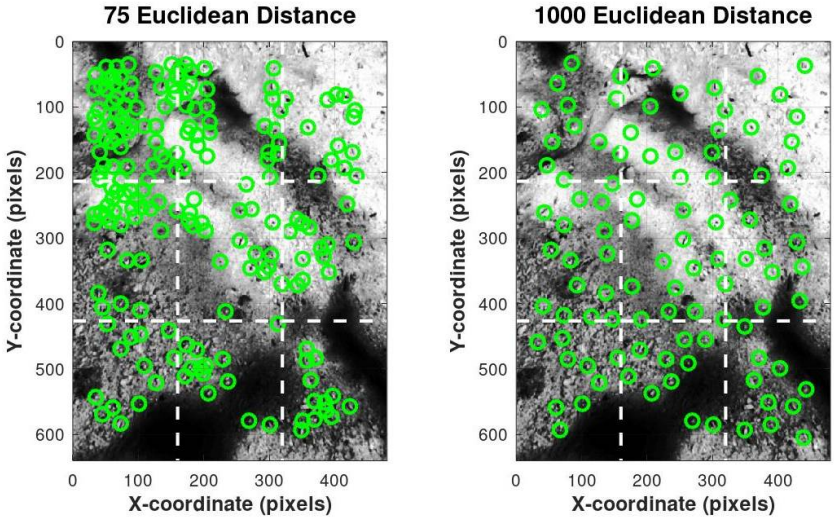


Figure D.2: Comparison of 75 and 1000 minimum Euclidean distance thresholds: 197 features detected with 75, and 101 with 1000

The optimal threshold values were found to range between 150 and 600 Euclidean distance, or 12 to 24 px. These values provided an excellent balance across all metrics, including the number of features detected and their distribution. A Euclidean distance of 275 was ultimately determined to be the most optimal, as shown in Figure D.3. Therefore, a threshold of 275, *i.e*, 17 px, was selected for inclusion in this project.

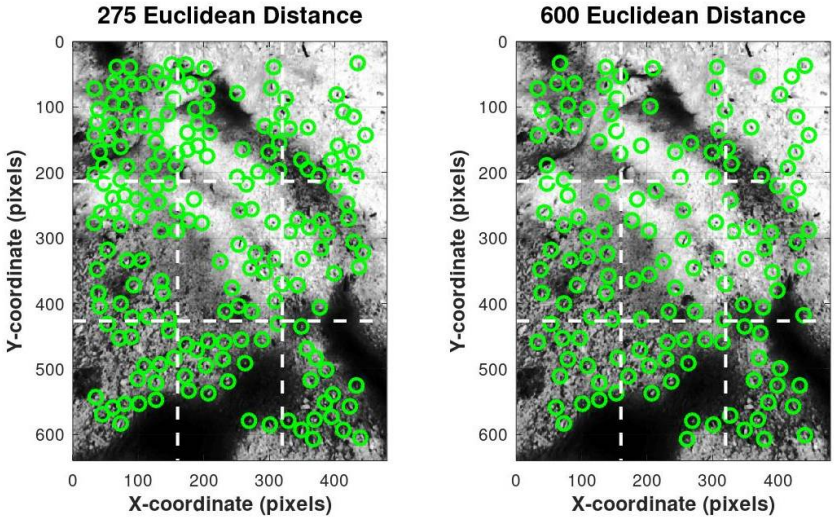


Figure D.3: Comparison of 275 and 600 minimum Euclidean distance thresholds: 189 features detected with 275, and 142 with 600

Appendix E

Pixel iteration test

The objective of this test was to evaluate the performance of the feature detection algorithms by reducing the number of px being processed. As shown in Table E.1, there is no significant advantage to iterating over every pixel in the image. In fact, both the number of features detected and, more notably, the execution time were negatively impacted, with the execution time doubling.

Table E.1: Comparison of feature detection performance: Iterating over every pixel vs. every other pixel in 20 seconds

Method	**Average execution time** (s)	**Average features detected**
Every pixel	0.0154	177
Every other pixel	0.006273	182

Figure E.1 clearly illustrates the issue described in Chapter 5, where the feature array becomes filled before reaching the end of the grid. This results in features being clustered on the left side of the grids, as observed in the first row, second and third columns of the image. The areas within these grids are highly contrast-rich, which makes it easier for the algorithm to identify interest points. However, counterproductively, this also hindered the total number of features detected and their distribution across the grid.

Figure E.2, on the other hand, shows a much more even distribution of features across the grids, while also increasing the number of features detected and reducing the computational load.

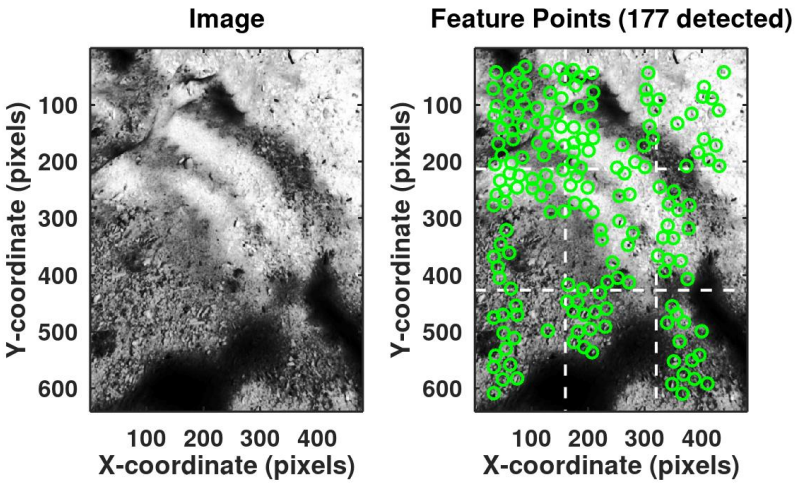


Figure E.1: Feature points after iterating over every pixel

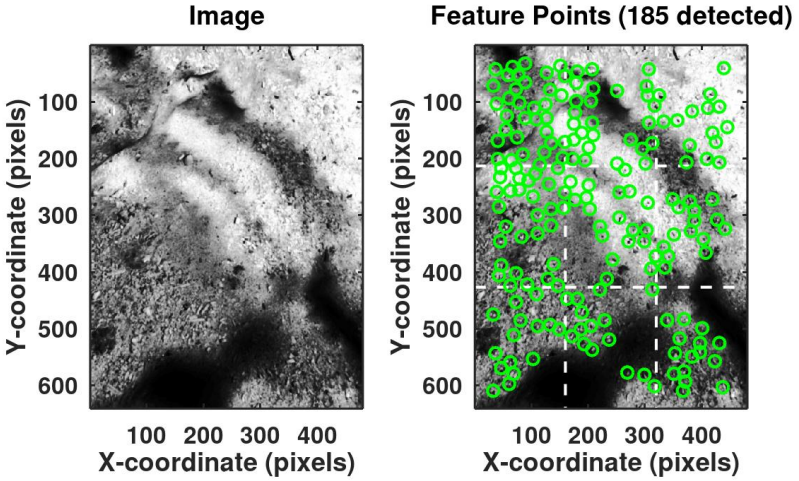


Figure E.2: Feature points after iterating over every other pixel