ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL Robotic Systems Lab
Prof. Dr. Marco Hutter

Master Thesis

Autonomous Vision-based Safe Proximity Operation of a Future Mars Rotorcraft

Spring Term 2024

Supervised by:
Roland Brockers
Robert Hewitt
Marco Hutter

Author:
Lasse Fierz

Declaration of Originality

I hereby declare that the written work I have submitted entitled

Autonomous Vision-based Safe Proximity Operation of a Future Mars Rotorcraft

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Lasse	Fierz

Student supervisor(s)

Roland	Brockers
Robert	Hewitt
Marco	Hutter

Supervising lecturer

Marco	Hutter

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Zurich, 19th June 2024

Place and date

Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Preface

The purpose of this work is to contribute a small part towards the exploration of Mars and therefore, this universe.
I'm in awe of the brilliant work achieved by the ingenious people of this field and feel fortunate to be able to work alongside them.
As this work concludes my master's degree at ETH Zurich I feel tremendous pride and gratitude to have been able to study at this phenomenal university and for all the other opportunities I was given throughout my time as a graduate student.

Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by JPL Visiting Student Research Program (JVSRP) and the National Aeronautics and Space Administration (80NM0018D0004).
I would like to extend my gratitude to the Jet Propulsion Laboratory, the Robotic Systems Lab, ETH Zurich, and the California Institute of Technology for making this collaborative work possible. I would like to thank my supervisors prof. Roland Brockers, Robert Hewitt PhD, and prof. Marco Hutter for their guidance and insights throughout these months.
It is with a feeling of overwhelming gratitude that I conclude this chapter and, with it, this thesis. The months involved with this work have been amongst the most educational of my whole life. I gathered tremendous amounts of knowledge and expertise from amazingly talented people and had uncountable memories that I will treasure forever.
A few directed words of gratitude.
To Roland
For providing me with the opportunity to contribute my small part to NASA's inspiring journey. For all the guidance, insights and discussions which helped me push my boundaries and advance farther in this work. For a motivating nature to perform the best work possible and an honest and friendly spirit which made me feel at home.
To Luca
For being an inspiring bench neighbor and making me feel welcome from the very beginning.
To Dario
For being an upbeat and honest nature and never failing to lighten a day.
To Simone
For a captivating honesty, and for showing me what is possible.
To Rob and Pedro
For their expertise and their kind, helping hand.
To Adrian and Marco
For being a home - so far away from home and for never failing to inspire me to think outside the box.
To Micha, Alex, and Leona
For fun evenings and a refreshing presence.
To Mor and Far
For giving me everything and expecting nothing in return.
To Ronja and Tove
For always listening and being there when needed.
To Jana
For never giving up and standing by my side.

Contents

Abstract

An autonomous rotorcraft literally stands or falls on its reliable landing capabilities. This procedure cannot fail even once when that same rotorcraft is on Mars. The LORNA (Long Range Navigation) project tackles this problem by introducing a Landing Site Detection (LSD) mechanism that aggregates Structure From Motion (SFM) point clouds into a multi-resolution elevation map and performs landing site segmentation on the collected depth information. In this master's thesis we incorporated this landing site detection pipeline into an autonomy framework. We implemented a behavior tree-based landing mechanism to safely and efficiently select, verify, and discard detected landing sites. Furthermore, the pipeline was enhanced using a stereo camera depth input alternative to SFM for lower altitudes to remove the necessity of lateral motion to perceive depth. The software was tested extensively in a gazebo simulation on different synthetically created as well as recorded environments, and different behaviors were considered and analyzed throughout various Monte Carlo iterations. The contributions in this work aim to enable future Mars rotorcrafts to autonomously and reliably land at safe locations, thus enabling a more daring aerial exploration of the red planet.

List of Acronyms

- **UAV:** Unmanned Aerial Vehicle
- **SFM:** Structure From Motion
- **LSD:** Landing Site Detection
- **LS:** Landing Site
- **BA:** Bundle Adjustment
- **DEM:** Dense Elevation Map
- **OMG:** Optimal Mixture of Gaussian
- **LOD:** Level Of Detail
- **HiRISE:** High Resolution Imaging Science Experiment
  (High Resolution Satellite Imagery on the Mars Reconnaissance Orbiter (MRO))
- **LRF:** Laser Range Finder
- **GT:** Ground Truth
- **LSM:** Landing Site Manager
- **FSM:** Finite State Machine
- **BT:** Behavior Tree

Chapter 1

Introduction

With the Ingenuity rotorcraft's life cycle ending, the question about future Mars rotorcrafts and their capabilities draws ever closer.
Currently, NASA is developing two different rotorcraft Mars concepts for the future.

- Mars sample return helicopter concept

  The first is associated with the Mars Sample Return mission, which has since been canceled. In this mission, NASA's Perseverance rover collects Mars rock and sand samples in test tubes. Subsequently, these tubes are returned to a sample retrieval landing platform, from which a small rocket (Mars Ascent Vehicle) launches them into Mars orbit. ESA's Earth Return Orbiter will enclose them in a highly secure containment capsule and deliver them to Earth.

  In case the Perseverance rover won't be able to collect the test samples and deliver them to the lander, a Mars Sample Return helicopter concept is envisioned. This is a rotorcraft for low altitudes equipped with a mechanical gripper, offering an alternative way of transporting Mars samples to the retrieval station should the Perseverance rover fail to do so.

- Mars Science Helicopter (MSH)

  Secondly, NASA is conceptualizing a Mars Science Helicopter (MSH) project for future exploratory large-distance missions. The aspirations for such a rotorcraft are to cover farther distances at high altitudes and explore unknown terrain in higher resolution than the HiRISE orbiter camera can. This rotorcraft will use accurate onboard state estimation to navigate through a mission trajectory and land safely, autonomously, and reliably in previously unknown terrain. These two feats allow a Mars helicopter to perform much more advanced science missions than Ingenuity.

The Long Range Navigation (LORNA) project I have been involved with is developing an approach to tackle the second project's challenges while dealing with the constraints that rotorcraft missions on Mars provide us with. These are, namely, a limitation on the size and weight of the drone, a constraint on computational power due to the deployment of limited embedded processors, and lastly, a large delay in communication, which makes adaptive remote control from Earth impossible.

1.1 Objective

The conclusive high-level objective of the LORNA science concept is the achievement of long-range safe navigation, including global localization, safe landing site

detection, and full system autonomy. The navigation endeavor is tackled using a laser-range-finder augmented visual-inertial odometry state estimator, which uses map-based localization to achieve global localization. Landing site acquisition is achieved using a structure from motion-based 3D terrain reconstruction. This output is fed into a landing site detection node creating a multi-resolution dense elevation map for landing zone segmentation. Finally, a state machine-based autonomy framework orchestrates the entire procedural workflow.
The endeavor in this thesis was to create a front-to-back landing mechanism that combines the existing vision-based landing site detection algorithm with the autonomy framework. To accomplish this, both the landing site detection algorithm and the autonomy had to be altered. Last but not least, given that the structure from motion depth generation depends on lateral movement, which is less desirable for a drone navigating at low altitudes in unfamiliar surroundings, the utilization of a stereo camera for low-altitude 3D reconstruction presents a viable solution to attain real-time depth perception without necessitating lateral displacement. Such a stereo camera was implemented in this work representing a lightweight solution that allows a drone to perceive depth statically and in vertical and lateral motion.

1.2 My Contribution

In this work, I established the interface between the vision-based landing site detection algorithm and the autonomy framework to make informed landing decisions based on detected landing sites. A safe and efficient landing mechanism was implemented in the existing autonomy framework. This mechanism utilizes a novel stereo camera 3D reconstruction procedure to avoid lateral motion at low altitudes.

- **Stereo Camera Depth**

  A stereo camera was implemented in the simulated drone model to obtain stereo camera images. A stereo camera depth node was implemented to augment SFM to supply the landing site detection algorithm with a point cloud at low altitudes without necessitating lateral motion.

  An automatic switch was inserted between the SFM node and the stereo camera depth node, utilizing the already present laser range finder sensor on board. This allows for minimal computational overhead, as only one depth creation node runs at a time.

- **Ground Truth**

  A ground truth depth node was created for two reasons. First, it allowed the validation of the stereo camera depth output. Second, having a perfect point cloud of the terrain made specific testing of the autonomous landing behavior itself possible.

- **Autonomy LSD Interface and Landing Site Handling**

  Initially, the landing site detection output only consisted of one landing site's location. This output was enhanced to utilize many more characteristics already present in the landing site detection algorithm for the autonomy to make a more informed decision regarding what spot to select. These landing site properties are

  - Terrain roughness
  - Size
  - Terrain uncertainty

  - Detection altitude
  - Obstacle height

  The autonomy framework was expanded to correctly receive and sort incoming landing sites based on their individual heuristic score. Additional landing site handling mechanisms, such as re-detection, verification, and banishment, were introduced.

- **Behavior Tree for Adaptive Decisions**

  An adaptive landing procedure consisting of both existing and novel actions was implemented using the existing behavior tree structure from the landing state within the autonomy. The implementation of the landing behavior optimizes for both safety and efficiency.

- **Simulation Setup**

  Since the switch to Gazebo Garden was made recently, the entire visual pipeline (SFM + LSD) had never run with this simulation environment before. Therefore, I implemented the changes necessary to run the landing site detection procedure on the Gazebo sensor input.

- **Analysis of Landing Site Detection Pipeline**

  Though little work was done directly on the SFM and LSD algorithms, substantial testing was performed using these nodes. For the first time, this pipeline was tested and analyzed at the desired 100 m cruise altitude.

- **Deployment of LSD Pipeline onto an Embedded Processor**

  Currently, the drone's processor is modalAI's voxl2. Both the structure from motion and the landing site detection software did not run out of the box because they had incompatible dependency handling with the voxl's AARCH architecture. Resolving these dependency issues, I was able to run the landing site detection pipeline with the structure from motion depth supply on the voxl2 using a collected rosbag of images and respective poses from the xVIO state estimator.

1.3 Organization of this Thesis

- **Related Work**

  As is custom, I will introduce the reader to what has been done in this area. The main focus will be placed on vision-based landing site detection procedures and previous work on autonomous landing.

- **System Overview**

  The existing project architecture will be outlined. Emphasis will be laid on the methods that I have heavily interacted with in this thesis. These are mainly the structure from motion depth generation, the landing site detection mechanism and the autonomy framework.

- **Methodology**

  The high-level structure of the work implemented in this thesis is conceptually laid out here. The two key contributions, stereo depth, and autonomous landing are introduced, as is the ground truth used.

I will explain why a stereo camera is necessary as a low-altitude depth alternative. Additionally, I analyze the stereo option theoretically and conclude its usage domain.

I explain the ground truth depth source used in this work both to compare stereo with and to test the autonomous pipeline without the possibility of insufficient depth information. Additionally, I analyze the ground truth's comparability to SFM to ensure adequate testing of the autonomous landing pipeline.

Lastly, I outline the prerequisites for implementing the autonomous landing behavior and introduce the methodology for the final implementation in the autonomy's behavior tree structure.

- **Stereo Camera Depth Alternative**

  This chapter elaborates on the implementation of stereo camera depth. It displays a coordinate frame overview and discusses the entire process from sensor data handling to point cloud generation. Lastly, the stereo depth output is qualitatively compared to a depth camera-based ground truth.

- **Autonomous Landing Procedure**

  Here, I will lay out the core contribution of this project, which combines the existing system with the novel contributions of this work to put together a front-to-back autonomous landing procedure. First, I describe the interface between the autonomy and the landing site detection pipeline. Then I introduce the concept of the landing procedure before I show its implementation in the form of a set of actions structured in a behavior tree. Lastly, the working pipeline is shown in a case example of a science mission flown in simulation.

- **Evaluation**

  Here, I first analyze the landing site acquisition pipeline, examining the current shortcomings of the implementations. I will then introduce the test setup, according to which I performed repeated randomized simulation flights. I then introduce the outcome-defining metrics and the results of the test flights. Lastly, these results are analyzed numerically and visually, considering the final choices of landing sites.

- **Conclusion**

  I summarize this work's novel contributions and conclusively assess the characteristics and quality of the final landing pipeline. Shortcomings of the approach are pointed out, and remedies are discussed.

- **Outlook**

  Potential enhancements to the current system are laid out, and alternatives for future iterations are discussed. Emphasis is also placed on current insufficiencies and the necessity of resolving them.

Chapter 2

Related Work

Autonomous safe landing is perhaps the most important part of a rotorcraft's mission. Therefore, it is no surprise that tremendous work has been accomplished to achieve this crucial feature.
Camera sensors are highly advantageous for navigation due to their lightweight nature and the extensive research dedicated to their development over the years. Their minimal weight makes them particularly suitable for applications where payload capacity is critical, such as rotorcraft Mars missions. Decades of intensive research have culminated in highly sophisticated algorithms and methodologies that leverage the rich data captured by visual sensors and enable the daunting task of autonomous landing.
In the following, previous work in this field is introduced and discussed with consideration of our mission goal. Most approaches split the task of autonomous landing into a visual step that analyses the terrain or the surroundings, a selection task that determines the location to land at, and finally, a navigation part that considers the selected location and approaches it.

2.1 Landing based on Visual Markers

[1, 2] and [3] use artificial landing markers as indications of valid landing sites.

- Saripalli et al. [1] lands on a fixed landing pad using GPS and a camera. The pad is marked using a distinctive, geometric visual marking. The pad is recognized by leveraging the geometrical shape of the marker. The calibration of this recognition algorithm, distinguishing the landing marker from other high-frequency objects, was done using information from previous flights. After detecting the marker, the x-y coordinates and orientation of the center point are derived, and the drone navigates towards that location. The reactive high-level landing behavior is implemented using a state machine.

- Falanga et al. [2] introduce a method to land on moving platforms. They work with visual markers on the platform and utilize onboard sensors and visual-inertial state estimation to localize and pursue this visual marker. An extended Kalman Filter estimates the platform's motion, including position, velocity, and orientation. The high-level decisions are again made using a state machine with four states: takeoff, exploration, platform tracking, and landing.

- Mu et al. [3] use an off-the-shelf commercial drone with a camera tilted 45 degrees. They added a lens to this camera to achieve a downside view and

detect the marker. The pad detection and localization are achieved by running the deep learning-based YOLOv5 algorithm on a ground station. The connection between the drone and the ground station is established using a wifi connection. The platform tracking behavior is orchestrated using a state machine. It consists of an ascent to an overview altitude, the detection and localization of the marked platform, the navigation towards the platform with reactive steps for the case when the pad is lost out of sight, and the landing mechanism.

Visual markers are a great tool for efficiently detecting a valid landing area. However, they are not usable for our endeavor, as we are flying on Mars' uncharted terrain. Once valid landing sites are detected, the predominantly used approach of pursuing the chosen landing zone using visual sensors orchestrated by a state machine and using a vision-based on-board state estimator is a perfectly valid strategy for our mission as well.

2.2 Homography-based Autonomous Landing

[4, 5, 6] and [7] pursue autonomous landing implementations based on homography assumptions:

- Bosch et al. [4] use homography estimates and feature correspondences to continuously update a model of the surroundings. This way, they can determine horizontal, obstacle-free areas.

- Brockers et al. [5] exploit a planarity assumption of the scene to achieve landing on elevated platforms and ingress through rectangular openings. No visual markers are used. Using SURF-feature point correspondences, homography estimates are created. To refine these estimates, further surfaces are used to apply a multiple-homography-alignment-algorithm. Lastly, the calculated plane normal and the estimated vehicle motion parameters are used to find a point located on the plane. This visual algorithm is split into four phases: takeoff, plane detection, refinement, and descent.

Though a valid procedure to find adequate, flat landing areas using homography-based approaches is unfeasible for our purposes, as a homography assumption can't generally be made on Mars' rough and obstacle-rich terrain.

2.3 Landing Mechanisms based on Range Sensors

- Johnson et al. [8] use a LiDAR sensor for 3D reconstruction of surrounding terrain for a larger Mars lander vehicle. The LiDAR data is aggregated into an elevation map. On that map, landing areas are determined based on an assessment of their slope and roughness. In a subsequent step, the landing guidance algorithm navigates toward the currently chosen landing sites. This guidance module is designed to redirect the current pathing to another landing site should incoming measurements change the optimal landing site candidate.

- Scherer et al. [9] introduce a landing pipeline for a full-scale helicopter. LiDAR measurements are analyzed in a coarse-to-fine manner. This way, they can pre-determine promising landing areas which can subsequently be assessed in more detail. The defining characteristics of a landing site are the slope of a fitted plane and the roughness.

Weight is a limiting constraint for our endeavour, as a rotorcraft has to fly on Mars' 1% air density. Thus, a LiDAR sensor that easily weighs half a kilogram is not a valid choice. Cameras, on the other hand, can weigh as little as a few grams.

2.4 Mars Lander Approach

In [10], NASA pursued a vision-based strategy using a predefined map of Mars' surface and a downwards-facing monocular camera to orient a Mars lander in the surrounding terrain. However, rotorcrafts need to consider much smaller hazards compared to a lander. At 25 cm/pixel for images and 1 m/pixel for the DEM, the available Mars footage from the HiRISE camera on the Mars Reconnaissance Orbiter is not sufficient in resolution to supply prior information to the landing process of a UAV. Rover images and Ingenuity footage could be used. However, the usage of this data would significantly limit possible flight areas.

2.5 Monocular Stereo-based Autonomous Landing

- Desaraju et al. [6] use monocular images and the drone's lateral movement to reconstruct the terrain. Thereafter, all the terrain information farther away than a certain threshold is disregarded, and safe landing areas are detected on the elevation information. The criteria for selecting landing sites include the flatness and levelness of the area, ample space for the UAV to land, and the absence of obstacles.
- Brockers et al. [7] use two vision-based approaches for state estimation. First, the onboard camera is used as a velocity sensor by applying an inertial optical flow algorithm. This allows the drone to very quickly stabilize when thrown into the air. Secondly, visual self-localization and mapping are used to estimate the pose of the drone over a longer duration with significantly less accumulated drift. Both of these measurements and IMU readings are merged into an Extended Kalman Filter. All of this is processed on an onboard processing unit. For autonomous landing, the same approach as in [6] is applied using monocular camera images to reconstruct the surroundings and choose a high-vantage landing plane without obstacles and of sufficient size.

Both [6] and [7] use monocular stereo to reconstruct the terrain and detect landing sites on it. The problem of detecting landing sites on the direct stereo depth output is the generated depth errors. A possible remedy for this is elevation maps.

2.6 Elevation Mapping Approaches

Other modern approaches make use of 2.5D representations of the surroundings. While [11] uses a fixed surface elevation map, [12] and [13] use a robot-centric 2.5D terrain representation similar to the setup used in this project.

- Johnson et al. [11] uses monocular camera images and a structure from motion algorithm to create point clouds. These point clouds are converted into a fixed surface frame and aggregated in a dense elevation map (DEM). Hazard segmentation is performed using the common strategy of determining the slope on the map by fitting planes and considering smaller obstacles like rocks. A binary landing map is created based on these two metrics and a distance transform to ensure the minimally necessary size of a valid spot. Lastly, the site with the largest circular valid patch is selected. If two sites compete at the

same level, the one closer to the pre-determined emergency landing location is chosen.

- Fankhauser et al. [12] use a forward-looking range sensor and onboard state estimation to create a local, robot-centric elevation map used for intermediate path planning within a global mission. The map is implemented as a 2D grid where each cell stores a Gaussian cell state containing a height estimate and a variance. To account for the drift accumulated in the robot's pose, the uncertainty in the local map's uncertainty is grown based on the covered distance and orientation changes. This approach does not feed information from the created local map back to state estimation.

- Daftry et al. [13] use thermal-infrared cameras and a visual-inertial odometry algorithm to localize the UAV in its surroundings during nighttime. They apply a monocular depth estimation approach matching pixel intensities directly. These are aggregated in the 2.5D elevation map outlined in [12]. As in previous approaches, landing sites are detected on the created DEM according to slope, roughness, and size of a landing spot.

2.7 Learning based Methods

Other novel approaches use learning-based methods as did [14, 15] and [16].

- Abdollahzadeh et al. [15] uses a deep regression approach in the shape of a u-net to solve the challenge of detecting and scoring landing sites. The input to the network is an RGB image which is projected into a lower-dimensional latent space. From there, the intermediate representation is up-sampled again to finally output a landing site score map revolving around three main levels with smooth transitions in between - low risk, medium risk, and high risk.

- Neves et al. [14] counteract traditional sensor limitations for detecting marked landing sites by implementing a deep learning-based multimodal sensor fusion network capable of merging 3D LiDAR data, thermal, and ordinary camera images. The network consists of a backbone containing a block of CNNs for feature extraction, a transformer encoder, and two multi-layer perceptions, which output the two outputs which are a confidence score that a landing site indicated by a landing marker is present in a certain bounding box and the coordinates of said bounding box. Secondly, they introduce a reinforcement learning-based decision-making lander. This lander consists of multiple layers that, in the end, output the correct landing action. The action space is limited to moving forward, backward, left, right, and vertical descent. The reward function consists of a weighted consideration of each dimension's distance to the landing platform. It was trained on simulated data and transferred into the real world.

As deep learning methods have significantly outperformed classical computer vision methods in other applications, they are certainly promising and, in the long run, definitely a pathway to consider. In addition to great general performance, deep learning methods provide significant robustness to a change in sensor information, such as camera images taken at different times throughout the day.
However, learning-based methods come with significant drawbacks in the context of the pursued task. First, considering the limitations present in Mars missions, the probable additional computational overhead from learning-based methods can not be neglected. Furthermore, neural network-based solutions give up simplicity and interpretability for the benefit of precision. This is not to be underestimated

in a hostile environment such as Mars' rough terrain, where perfect accuracy and as much information for mission planning purposes as possible are required. Additionally, learning-based methods require substantial training data, which is not available in large quantities in the context of autonomous UAV landing. Especially when the training data needs to come from another planet. Lastly, missions flown on Mars pose unique challenges compared to conventional flights.
Note that the previously laid out Mu et al. [3] uses YOLOv5, a deep-learning-based detection approach for the detection of the landing marker. It is no surprise that in their setup this was run on a ground station as opposed to the processing unit on the drone.

Chapter 3

System Overview

fig. 3.1 shows the high-level architecture of the LORNA concept.
The xVIO state estimator uses the camera, IMU, and laser range finder sensor onboard the vehicle to estimate the rotorcraft's current pose at any given time. Map-based localization (MBL) using HiRISE maps allows for global localization. The flight controller receives these poses, and the output from its internal estimator is fed to the autonomy and landing site detection nodes.
As mentioned above, the rotorcraft is flown using a PX4 flight controller. It communicates with the autonomy framework using the MAVROS ROS wrapper for the MAVLink rotorcraft protocol. This connection is used for transmitting waypoint information and setting parameters.
The sensor images and their respective associated drone pose are given to the landing site pipeline. First, the 3D terrain is reconstructed using structure from motion. The reconstructed terrain is then given to the landing site detection algorithm(LSD) which first aggregates the information in a robot-centric multi-resolution elevation map and then segments valid landing sites on that. Detected landing sites are fed into the autonomy to make the required adaptive landing decisions.
The ground station QGroundControl is used for the creation of mission waypoints which are transferred to the autonomy using its mission planner support node introduced in section 3.3.1. Therefore, it is used more as a preparatory tool for the autonomy framework than a central part of the pipeline. In the developmental stages of the landing site pipeline, however, the drone can be flown without the autonomy. In such cases QGC serves as a user interface allowing the mission planning and parameter setting.

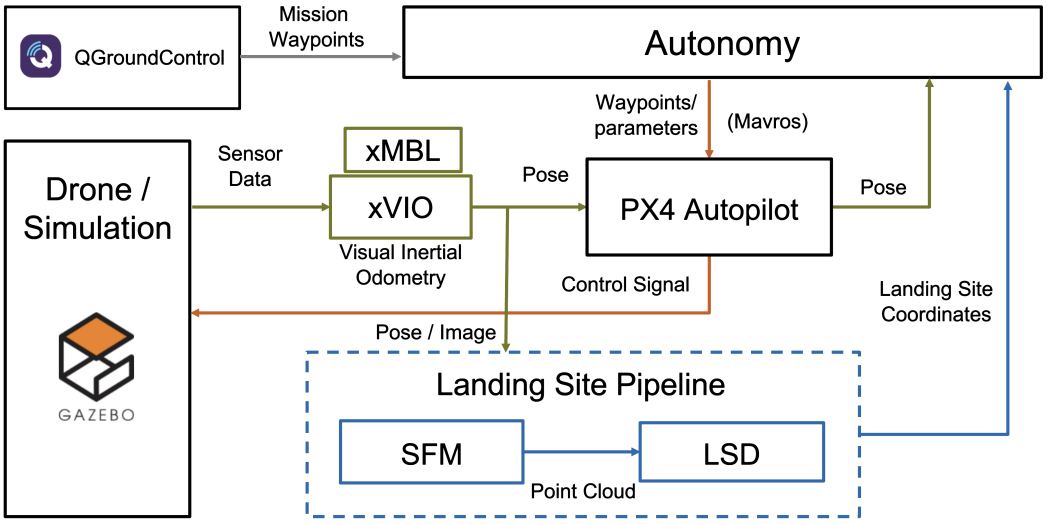


Figure 3.1: LORNA Project Setup

As the state estimator and the map-based localization were not fully implemented at the time of this work, ground truth pose information from the simulated sensors was used instead. This is shown in fig. 3.2

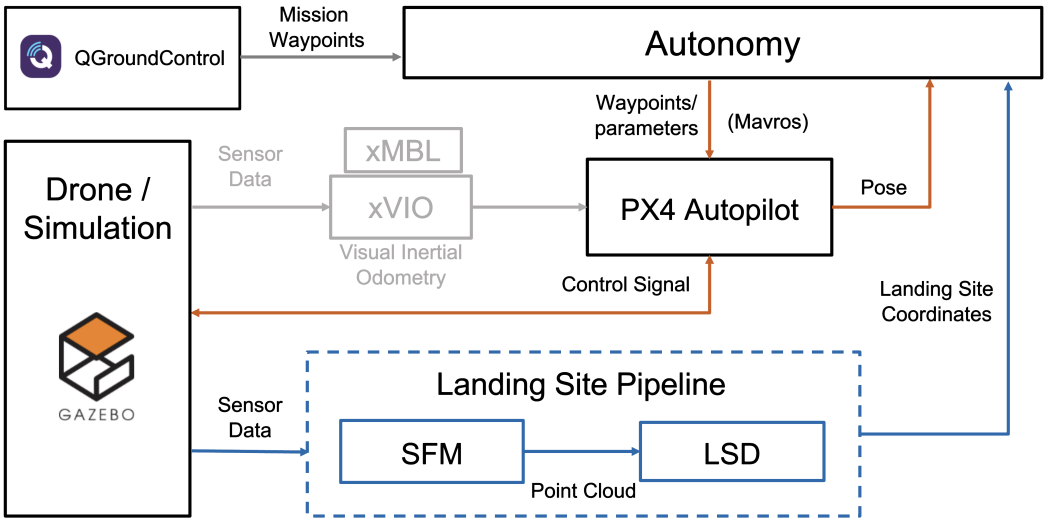


Figure 3.2: LORNA Project Setup with GT pose

Since this thesis focuses on integrating existing software instances, it is essential to delve into each component and elucidate their operational mechanisms.

3.1 Simulation

I almost exclusively used a simulated drone to test the algorithms in this work. This allowed for a facilitated development with respect to speed and safety. The choice of simulator, Gazebo Garden, was made prior to this work during the development of the autonomy.

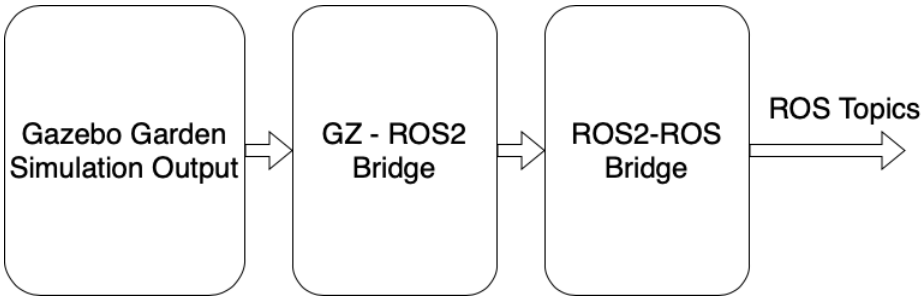


Figure 3.3: Gazebo ROS Bridge

As the entire software stack of the LORNA project depends on ROS instead of ROS2, a bridge was used to convert the sensor information from Gazebo Garden to ROS2 and from ROS2 to a ROS topic.

Most development during this work was performed on the Arroyo map shown in fig. 3.4. The Arroyo Seco is the area outside the Jet Propulsion Laboratory in Pasadena, California. Though not strikingly similar to Mars' landscape at first glance, the Arroyo contains varying terrain, including small rocks and uneven ground, which makes it a good test ground for UAV Mars landing applications.

Figure 3.4: Gazebo map of the Arroyo Seco area outside JPL at 4 cm per pixel

3.2 Landing Site Detection Pipeline

The landing site detection pipeline consists of two nodes. A structure from motion node [17] which creates a point cloud using a keyframe-based stereo depth approach on monocular images and a landing site detector algorithm [18, 19] which aggregates the depth measurements into a rolling buffer-based multi-resolution elevation map and segments landing sites on the created DEM. These found landing sites should then be supplied to the autonomy.

3.2.1 Structure From Motion (SFM)

The structure from motion approach retains images and their associated pose priors as keyframes in a sliding window buffer. These keyframes are used for a bundle adjustment pose refinement, and subsequently, using a previously stored keyframe and the incoming pair, a stereo depth algorithm is applied to achieve a depth image. In the following, the three main parts are explained in more detail:

Key Frame Handling

At the end of an iteration, the keyframes are updated. Naturally, in the beginning, before the rolling buffer is filled, each incoming frame is simply added to the back of the buffer queue. Upon reaching the queue's maximum capacity, incoming frames are analyzed regarding the two following factors:

- **Frame to frame parallax**

  To ensure that enough distance between the last keyframe and the incoming frame has been covered, the root-mean-square value for the parallax distance of the matched features is calculated and compared with a minimum threshold.

- **Information retention and contribution**

  New information should be added to advance the buffer over time. Therefore, new features should be detected on an incoming image. However, to match the incoming image with the existing keyframes, sufficient features must also be shared with the existing ones.

Therefore, if these two conditions are fulfilled, a new frame is accepted in the buffer. To determine the quality of the current keyframes, the oldest keyframe is used to determine the following metrics:

- **Number of matched features between the two frames**

- **Overlap of the two image footprints**

  The areas' overlap is calculated using the simple pinhole model formula for an image footprint and the calculated baseline.

- **Feature area ratio**

  The maximum rectangle spanning all features is derived for both frames. Due to parallax from lateral motion, they don't coincide, allowing us to determine the feature span overlap area ratio.

If these three metrics lie above a certain threshold, the keyframes are considered good, and only the newest key frame is exchanged, retaining all others. If not, the new frame is added to the back, pushing the rolling buffer further and losing the oldest previous keyframe.

Bundle Adjustment

Before 3D reconstruction, the poses are refined using a bundle adjustment algorithm. This is beneficial as the state estimator always has a certain error that drifts over time.

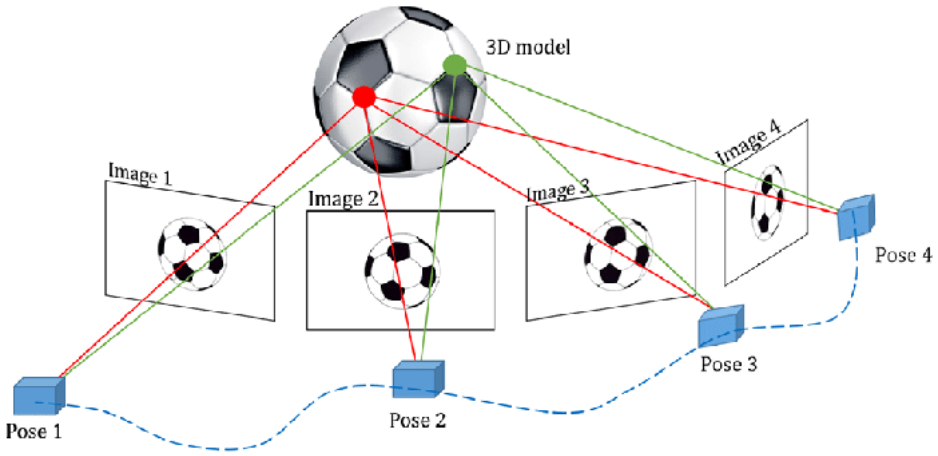


Figure 3.5: Bundle Adjustment Schematic as shown in Zhao et al. [20]

fig. 3.5 shows the change in reprojection, which the bundle adjustment optimization technique endeavors to diminish. Generally, it does so by considering a number of keyframes and optimizing their poses and the camera parameters used to acquire the images. In this case, only the pose refinement is considered for the keyframes in question.

Monocular Stereo Depth

This crucial step of 3D reconstruction can be performed after choosing adequate keyframes and refining their poses. For this, the keyframes are checked to have sufficient but not fully coinciding image overlap with the new image and a small enough baseline inclination so that assuming the same altitude above ground is reasonable. If the checks are successful, the keyframe with the largest baseline to the new frame is chosen for the stereo procedure.
Note that if all keyframes comply with these checks, then the oldest keyframe is always used for the stereo depth creation. Therefore, in practice, the reference view for the depth image is the oldest keyframe for a few iterations until the overlap with the new image is too small, and the oldest keyframe is switched out. This results in a small jump in the depth image's reference location.
fig. 3.6 shows a selection of such keyframes to perform stereo on. Note the visible rectification in the left reference image, which has been rotated slightly in order to enable horizontal feature matching.
The disparity values can be derived using the two frames and the parallax distance between the detected features. With the camera's intrinsic parameters and the disparity values, a depth image of the terrain can be created. A color-coded example thereof is shown in fig. 3.7 where red points are closer and blue ones are further away from the camera sensor.
Then, using these depth values of the individual and the camera parameters, the 3D coordinates of each detected point can be calculated, and thus, the depth image is converted into a point cloud which is the format required for LSD. The RViz visualization of such a point cloud is shown in fig. 3.8.

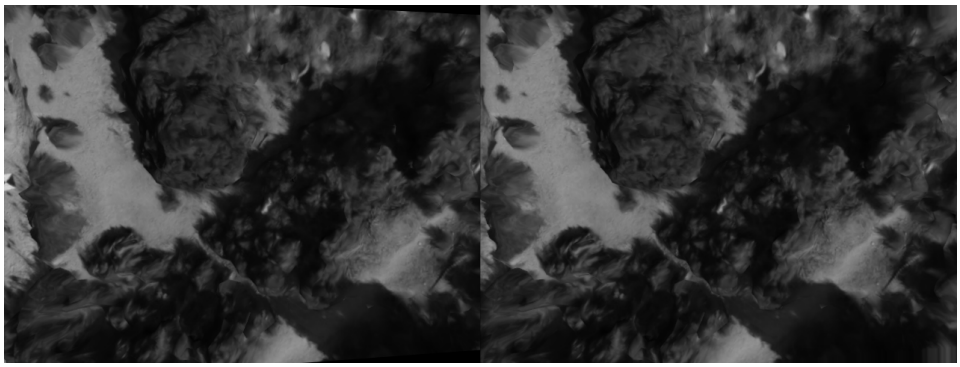


Figure 3.6: Left and right keyframe image selected for SFM depth

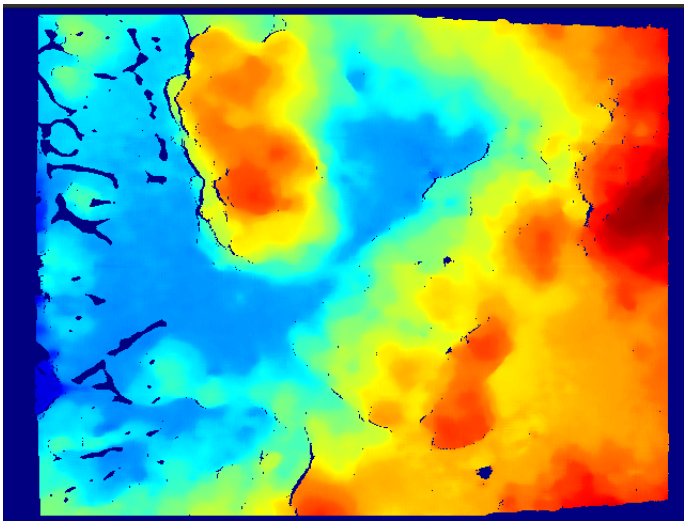


Figure 3.7: SFM depth image created from the stereo approach

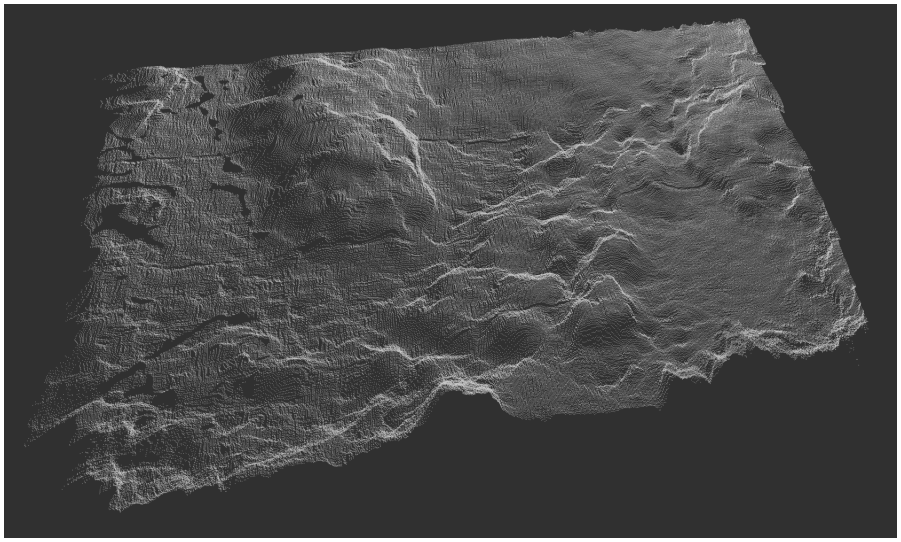


Figure 3.8: SFM point cloud visualization in RViz

3.2.2 Landing Site Detection (LSD)

Depth Aggregation

The foundation of the landing site detection mechanism is a rolling buffer-based multi-resolution elevation map as indicated in fig. 3.9. Each base layer cell is represented with 4 cells at a higher resolution layer.

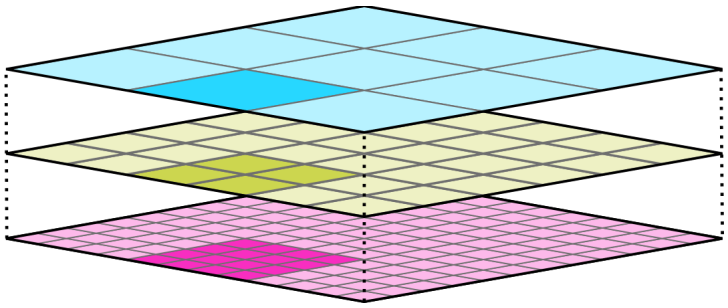


Figure 3.9: Multi resolution elevation map as displayed in [18]

As introduced in Proenca et al. [19], each cell in this dense elevation map (DEM) is composed of an optimal mixture of Gaussian (OMG) state described by the following three variables:

- μ - The mean of the depth at this cell
- σ^2 - The variance of the depth measurements
- S - An auxiliary variable to keep track of past measurement's uncertainties

Each input iteration, a depth measurement, and the associated stereo depth uncertainty are received. The measurements are placed in the respective cells based on the perceived points' level of detail (LoD). The LoD is $\frac{z}{f}$, where z is the depth measurement, and f is the camera's focal length. Specifically, an incoming measurement is placed in the highest resolution layer of all layers, which is coarser than the incoming measurement. If the measurement has a larger pixel footprint than all of the layers, it is entered in the coarsest layer which is at the top.
In a subsequent step the measurements are pooled up to the top layer and down again to make the DEM more consistent and fill in missing values. The same result could be achieved by entering each measurement individually and directly pooling it up and down the cells that cover its footprint. However, splitting the procedure into a step that acquires all measurements and a second task that collectively pools all cells achieves the same with fewer updates.
Like Kalman filters, the OMG cells' uncertainties improve over time as more measurements are entered. Because of this, the DEM's terrain estimate will converge over time. This is beneficial for the acquisition of landing sites on well-known terrain. However, convergence on known terrain makes it harder for future inputs to override the existing DEM. To counteract this Proenca et al. [19] proposed a time inflation factor which was however not implemented. This would reduce the weight of older data and make it thus easier for incoming points to alter the cells.
When an incoming point cloud comes from a camera pose which yields less than 95% image overlap with the previous location, a map shift of the rolling buffer is performed. The rows and columns no longer necessary are cleared, and the starting index is shifted for further incoming point clouds. This implementation allows the map to move without the need to reassign filled cells. In other words,

the required parallax effect is achieved by merely reassigning the cell values of the DEM container.
This shift is visualized in fig. 3.10. Note the images[1] are only used for ease of demonstration. They represent incoming point clouds. The dashed orange arrow indicates the drone's movement relative to the instantaneous DEM. The new image is entered at the new starting index and points exceeding the map boundaries are wrapped around to the other side. The deleted part of the DEM is filled with the incoming points and new measurements update the previous information at the unchanged locations. Note that fig. 3.10 shows a very radical map movement. Smaller camera movements between iterations would lead to most of the DEM to be retained.

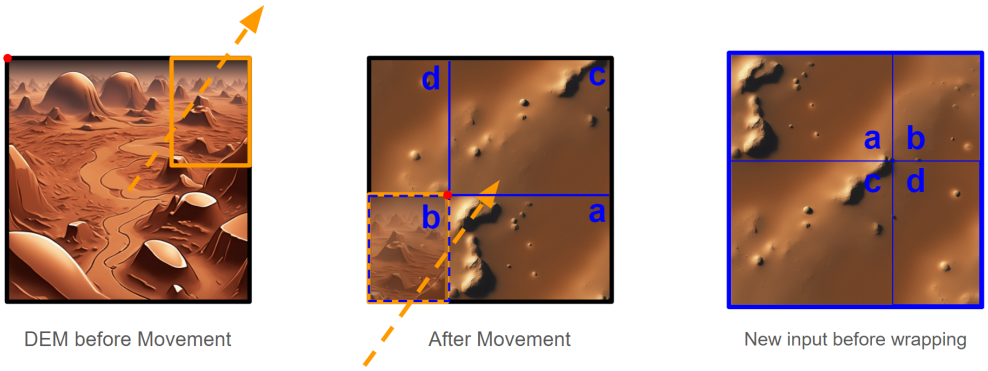


Figure 3.10: Schematic LSD map movement procedure: Outlined with orange is the retained part of the DEM (in the second frame, it is overlaid by part of the new input and thus outlined by a dashed line). The blue image represents the next input point cloud, and the red dot indicates the starting index where the new point cloud will start to be stored. Both images were created using AI.

Hazard Segmentation

Landing sites are then detected on this created DEM. This is done using a roughness and slope assessment of the perceived terrain. Roughness defines the maximum absolute altitude difference around a cell for a given resolution layer, and slope is determined by fitting a plane to the vicinity of a considered point.
The roughness check is done for each resolution layer of the DEM. To assess the roughness around a cell, all surrounding cells within a predefined proximity threshold are considered, and the maximum value is stored. The computational cost of the hazard segmentation is dominated by the roughness checks at the lower layers. Therefore, to make it more efficient, a minimum and maximum buffer were introduced, respectively. This allows the storage of the extreme values at a given location, and then for the roughness assessment of a neighboring cell, the knowledge about that mostly overlapping area can be used. With this, only the new margin of cells has to be considered for minimum/maximum candidates of subsequent cells. Finally, the roughness of a cell is the difference between the maximum and minimum value in that cell's vicinity.
A visualization of this procedure is shown in fig. 3.11:

[1]Images were created using DeepAI's image generator (https://deepai.org/machine-learning-model/text2img)

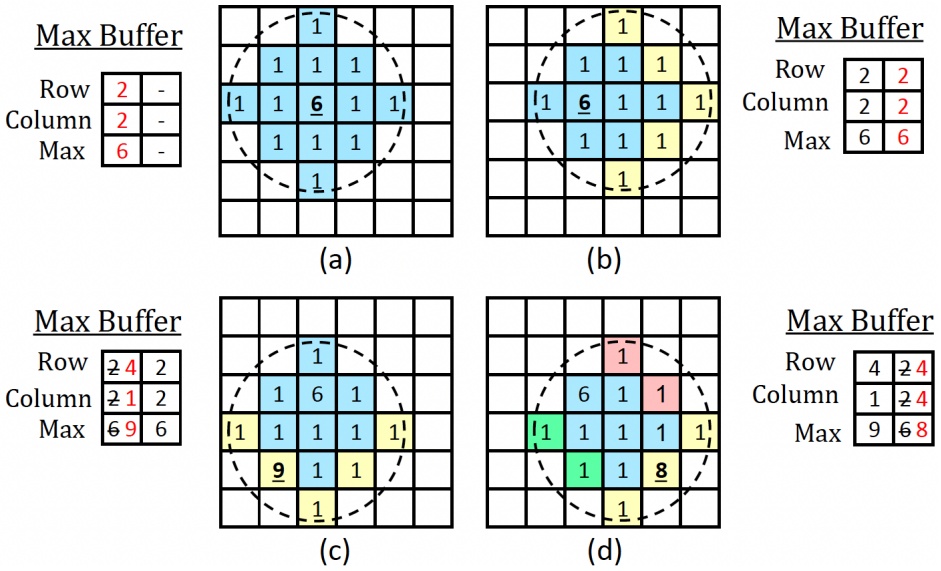


Figure 3.11: Optimized roughness assessment as shown in Proenca et al. [19]

As the slope of a location depends on a larger encompassing area around a cell, the assessment of the slope at the highest (coarsest) layer is sufficient. For each cell in the top layer and the surrounding area, a plane is fitted using the linear least squares equation $Ax = b$ where A is a matrix containing the x and y coordinates and a homogeneous coordinate for the plane's offset in each row. The solution vector x contains the plane's properties (N_x, N_y, d) where N_x and N_y define the plane's normal vector's orientation and d is the plane's offset. Note that N_z was fixed to be 1. The maximum inclination can then be calculated as the angle between the normalized normal vector and the z-axis $\theta = \arccos\left(\frac{N \cdot e_z}{||N||}\right)$
If the roughness and inclination values lie within the acceptance threshold, the spot is recognized as a landing site and marked as such in a binary landing map.(3.12)

Landing Site Selection

After applying a distance transform on the landing site map, which yields the minimum pixel distance to a non-landing site, non-maximum suppression is performed. In this maximum peak detection, only N landing sites are selected, which have a size that is at least bigger than the largest landing site multiplied by a certain diminutive factor and are sufficiently far away from another peak to be considered an actually distinct landing site. The non-maximum suppression on the landing site sizes is shown in fig. 3.13.

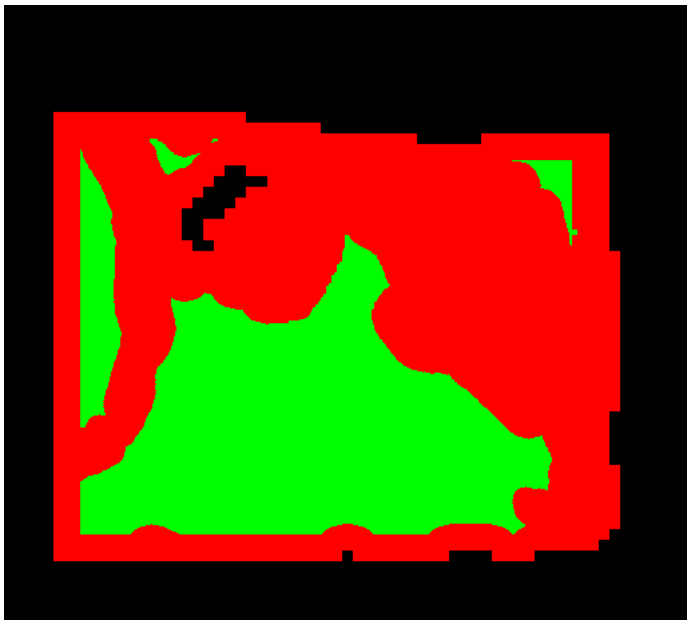


Figure 3.12: Binary landing site map from the LSD debug image. Black is terrain not yet covered, red is invalid landing sites, and green is detected valid landing sites.

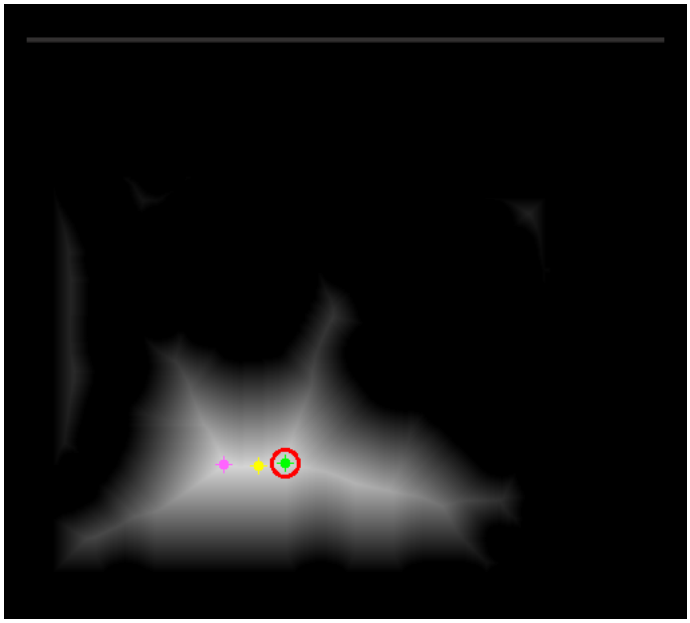


Figure 3.13: Non-maximum suppression: the three best landing sites at any given iteration are chosen, of which the very best is encircled.

Lastly, their positions are refined by applying a mean shift algorithm for a few iterations. This algorithm uses a Gaussian kernel, which considers the roughness, normalized distance transform, and the uncertainty of the cells in the sampled region. The exact implementation is shown in Proenca et al. [19]. Selecting multiple peaks at any given time is useful, as the mean shift algorithm is prone to local minima. So having multiple candidates and applying the mean shift algorithm to each one individually makes selecting a good landing site more probable.

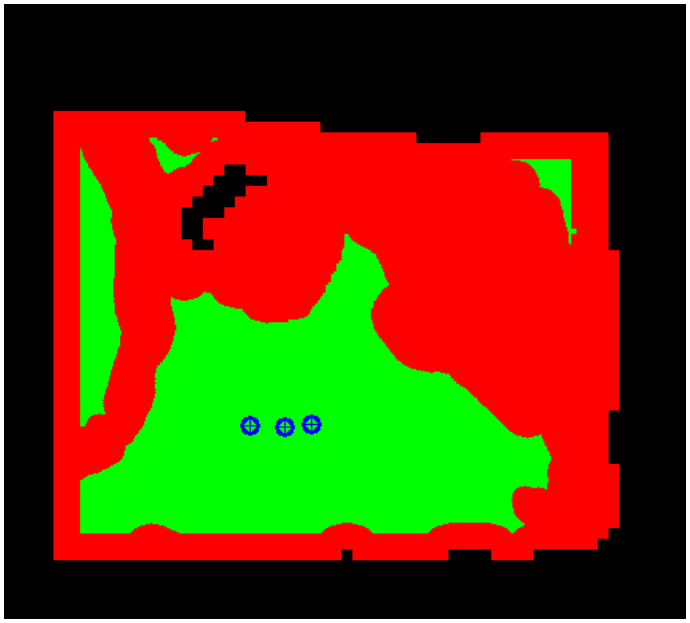


Figure 3.14: Binary landing site map after non-maximum suppression

fig. 3.14 shows the landing map with the final shifted maximum area landing sites. These optimized landing site candidates are indicated with the blue crosses.

Debug Images

The landing site detection debug image is a good comprehensive visualization of the landing site detection algorithm.
fig. 3.15 shows such a debug image. On the left, the multi-resolution map displays the same terrain area in different resolutions. Red pixels are closer, blue pixels are further away. In the middle, one can see the uncertainties of the detected points. Blue signals a low uncertainty, while red denotes a high uncertainty. On the right is the binary landing site map mentioned above. Green indicates valid landing sites, and the blue crosses indicate the chosen non-maximum suppressed and mean-shifted landing sites.
fig. 3.16 shows the monocular images used by SFM to create the point cloud that SLD built the DEM with.

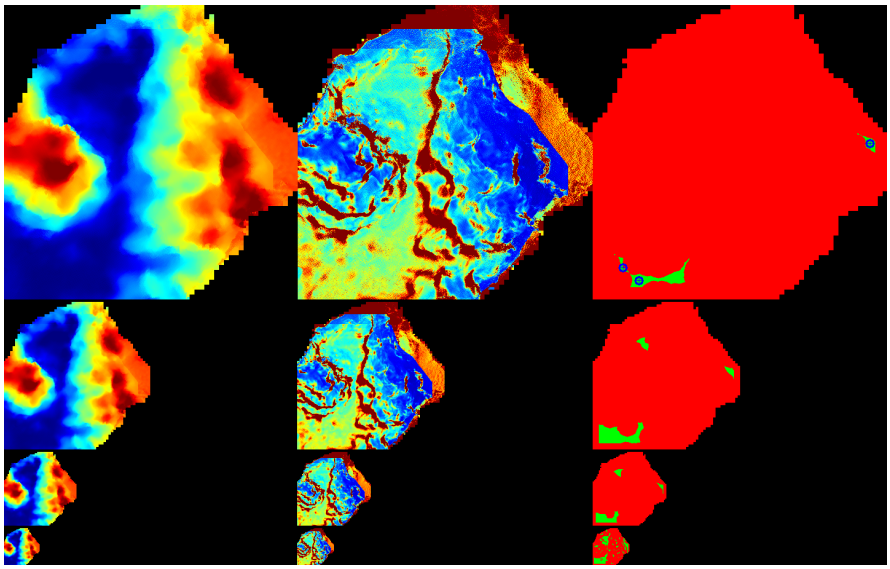


Figure 3.15: LSD Debug Image - Left: DEM, Middle: Uncertainties, Right: LS Map

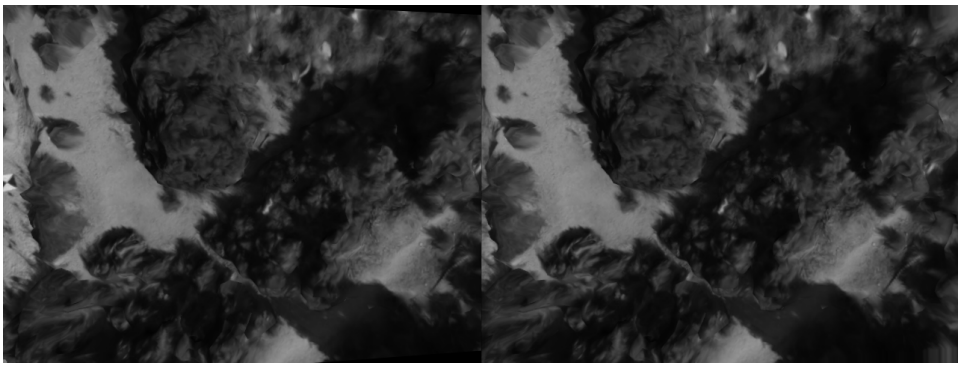


Figure 3.16: Monocular images used for stereo as reference

3.3 Autonomy

The autonomy framework was developed within the LORNA project as a master's thesis by my friend and colleague Luca Di Pierno ([21]). It is the overarching instance governing all the necessary behaviors during a mission and constituting the interface between all the nodes of the navigation pipeline.
A visualization of the setup is shown in fig. 3.17.

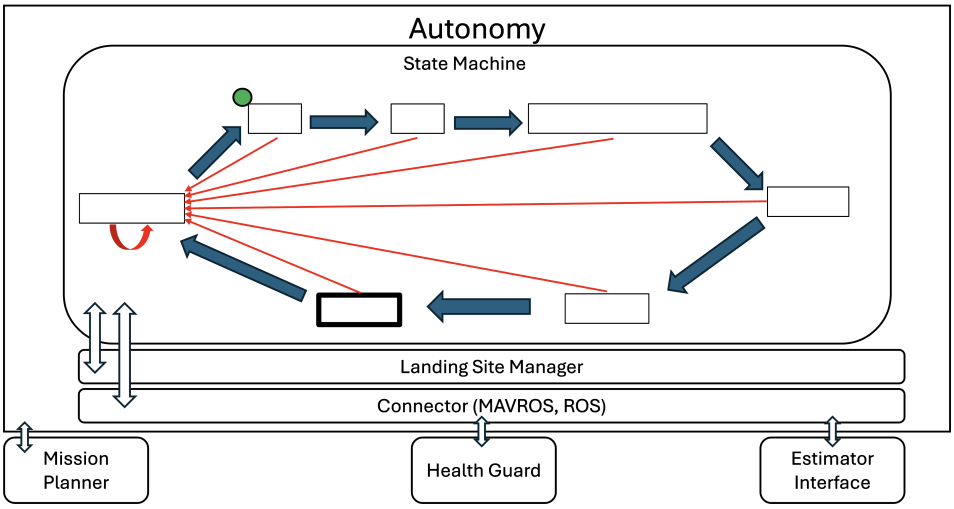


Figure 3.17: Simplified overview of the autonomy

As shown in fig. 3.17, the core of the autonomy framework is a finite-state machine that reactively orchestrates task execution, ensuring that the individual actions are carried out in an organized and structured manner. Alongside the state machine, there are two separate entities. A connector and the landing site manager. Lastly, a blackboard module serves to transfer information between states of the state machine.

3.3.1 Support Nodes

Three support nodes aid the autonomy framework. These are shown at the bottom of fig. 3.17:

- Mission planner

  As the name suggests, the mission planner enables the easy packaging of a mission's waypoints into a format that the autonomy can parse and understand. For this, it merely needs a mission plan file from a ground station such as QGroundControl used in this work. From this, the mission CSV file is created with the plain navigation information required to fly the desired course. Furthermore, the mission planner allows users to add tasks to a mission waypoint. This allows for the very fast creation of missions with the potential modular enhancement of waypoints for science tasks.

- Health-Guard

  This node is a notification tool that alerts the autonomy in case of a health issue in the system. This could be a range of events, such as an actuator failure or simply a low battery level. The connection to the autonomy is established using a ROS service in the ROS connector interface (3.3.2). At

each autonomy tick, the health state is queried using the ROS service. In case of an issue, the autonomy assesses the gravity of the issue and, in case of an emergency, initiates the reactive behavior.

- Estimator Interface

  This is the auxiliary node that handles the localization inputs. It is designed to be able to handle incoming pose estimates from different sources such as GPS, Vicon (a motion capture-based localization system) and LORNA's xVIO estimator. In this work, apart from occasional autonomy tests on the hardware using the Vicon system and a field test using GPS, the estimator interface was not used, and the flights were performed on the flight controller's pose, which is the simulation's ground truth pose.

3.3.2 Autonomy Interface

Connector

The connector consists of two parts and constitutes the interface with the other software instances of the project:
The MAVROS connector establishes the off-board mode in the PX4 autopilot to control the parameters and waypoints through the autonomy directly, effectively replacing the flight controller's decision-making behavior. It performs the arming and disarming procedure and executes actuator checks, among other tasks, to enable a rotorcraft flight.
The ROS connector receives health events about the system's status from the health guard node, the drone's current pose from the flight controller's state estimator, and crucial for this work, the detected landing sites from the LSD algorithm.

Landing Site Manager

The autonomy contains another separate entity in addition to both the state machine and the connector, the landing site manager.
It consistently processes the incoming landing sites from the LSD algorithm, which were received by the ROS connector. During this processing, each landing site's distance to the drone's current position is calculated, and each landing site is assigned a heuristic value and ordered according to that value in the landing site buffer. This buffer is implemented as a min-heap. This allows replacing and ordering a new landing site with comparatively small overhead.
Prior to this work, the heuristic used for comparison was simply the distance to the drone's current position resulting in the consistent choice of the closest landing site.
The landing site manager's site processing is depicted in fig. 3.18
Upon reaching the landing state and executing the landing behavior, the landing site buffer is ordered in descending order, and the best landing site is returned, which can then be used as the next navigation target to finally land at.
Both the landing site manager and the connector are implemented as singleton nodes for facilitated use throughout the entire autonomy framework. This means that they are accessible in every action performed by any state of the state machine without the need to instantiate a landing site manager object instance. This is achieved by removing the class's copy constructor and assignment operator and creating an instance acquisition function, which returns a pointer to the class's single static instantiation.
This thesis expanded the landing site manager significantly, both regarding the heuristics considered and the general handling of detected landing sites.

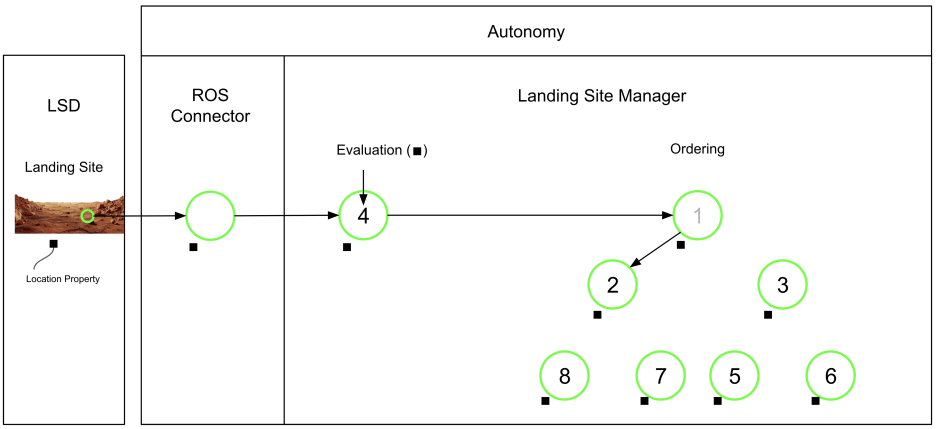


Figure 3.18: Landing site handling from detection to sorted buffer index in the landing site manager: A landing site's location, indicated with a green circle, is detected by LSD and given to the autonomy. The landing site manager assigns the site a heuristic value based on the location and sorts it in the buffer array.

Blackboard

The blackboard is yet another singleton entity within the autonomy that allows information to be easily shared between individual modules of the state machine. It is implemented using a template-based approach so that any variable can be directly stored and retrieved dynamically without changing the structure of the blackboard.

3.3.3 Finite-State Machine (FSM)

Before going into the state machine implementation of the autonomy framework, the finite-state machine's working principle is explained:

Working Principle

A finite-state machine is a mathematical model of computation that allows the reactive implementation of a system with a finite number of states. It is defined by a finite set of possible states with an initial state, the transition rules in between these states, the events that trigger these transitions, and the actions which are performed when the system is in a given state. It provides a very clearly structured implementation of the various states that a system can be in and is, therefore, a matching choice for implementing a rotorcraft mission.
A state machine continuously executes a tick. An executed tick carries out the action associated with the state in which the state machine currently resides. Depending on the output of that action, a transition event is triggered, moving the state machine from one state to another. This way, the decision flow is propagated throughout the state machine until the behavior is concluded. This conclusive state, in turn, also has a transition that allows the state machine to start over again.

The Finite-State Machine in the Autonomy Framework

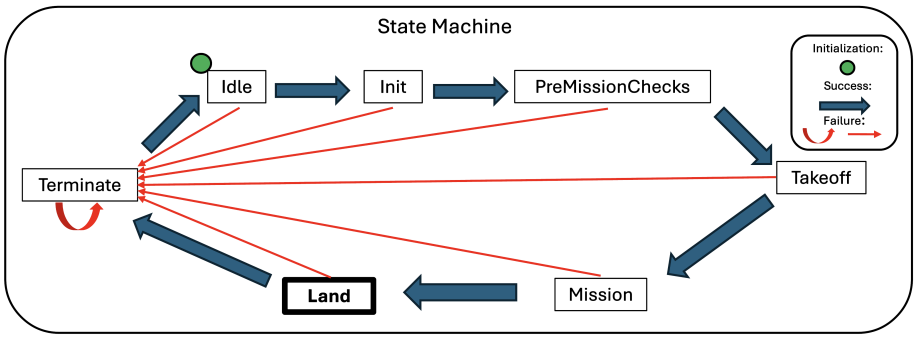


Figure 3.19: State machine of the autonomy framework

In our case, the autonomy itself is a pseudo-state machine, continuously executing a tick. During nominal conditions, this simply denotes a tick in the finite-state machine. The autonomy's state machine consists of the following states shown in fig. 3.19:

- Idle

  The state machine's startup state.

- Init - initialization

  The autonomy's global parameter server is started and the landing site manager, and the connector are initialized to start processing incoming information.

- PreMissionChecks

  The autonomy performs the last checks before the mission is started. These include a pre-arm check, a battery check and an actuator check.

- Takeoff

  The drone takes off using the MAVLink connection to the flight controller and the given takeoff altitude.

- Mission

  The drone navigates to each waypoint given in the mission plan.

- Landing

  The reactive landing sequence is initiated which uses detected landing sites and the retained takeoff location if needed.

- Terminate

  This is the end state of an executed autonomous flight. If any node executed within a state fails, the transition to the Terminate state is made as well.

In the case of a health event, however, the state machine handles the health event based on the state in which the FMS is currently. For instance, if it arises during initiation, the state machine simply transitions to the termination state, aborting the flight. In contrast, if the FSM's current state is the mission state, the rotorcraft immediately returns to the home position to land.

In each state during a state machine tick, the respective behavior node is executed. Most often, this is a single action node. A behavior tree is used for more complicated procedures. This is, for instance, the case for the landing state. For this thesis, the landing node indicated in bold in fig. 3.17 is the most crucial part of the state machine. This is where the adaptive landing decisions are made to choose an adequate landing zone and orchestrate safe traversal to that location before finally landing on the surface.

3.3.4 Behavior Tree

A behavior tree is a tool for systematic hierarchical plan execution. It enables the creation of complex tasks comprised of various simple tasks without having to worry about the implementation details of such a small modular task. In that sense, it is similar to a finite-state machine.
However, unlike a state machine, nodes in a behavior tree are always executed in a hierarchical order. A node in a behavior tree is in one of three states at all times:

- RUNNING
- SUCCESS
- FAILURE

Being restricted to three states doesn't implicitly advocate for the system's reactivity. The more transition possibilities an entity has, the more reactive the system containing the entity is. However, the hierarchical execution flow of the nodes makes behavior trees more easily scalable than finite-state machines. This is because implementing many transitions in a state machine quickly increases the system's complexity. On the other hand, action nodes in a behavior tree can be organized in higher-level control flow nodes, allowing scalability of the system without a quick increase in complexity.
This is a matching choice of framework for the creation of complex modular flight behaviors, especially since a rotorcraft mission can be easily split into small modular maneuvers such as ascension, lateral motion, and rotation. Additionally, behavior trees allow anyone to draw up complex missions without the necessity of understanding the individual action node implementations in depth.

Working Principle

As the name suggests, a behavior tree can be visualized as a directed tree, with the root node as the starting point.
After the root node, three categories of tree nodes can follow:

- Control flow nodes
- Action nodes
- Decorator nodes

Control flow nodes are comparatively higher-level nodes that manage the execution flow of the behavior tree.
The action nodes define the actual low-level tasks performed in the broader behavior tree setup.
Lastly, decorator nodes are auxiliary nodes that directly alter the workflow of action nodes to achieve smoother and more complex tree structures.
Preceding nodes in a tree are called parent nodes, while subsequently executed nodes are called child nodes of a node. A root node has no parent node and exactly

one child node, while control nodes have one parent node and potentially several child nodes. Action nodes are leaf nodes with a parent node but no child nodes, and decorator nodes have one parent and one child node.
Similarly to a state machine, the workflow of a behavior tree starts with the root node continuously sending a tick signal to its child node.

Autonomy Control Flow Nodes

The following control flow nodes are implemented:

- Fallback Node

  It attempts to execute the first child node, and if successful, it returns a success state. Otherwise, it continues to the next child node. If no child node was successful, it returns the failure state. A boolean operator analogy for this would be the logical OR, stopping at the first successful entity.

- Sequence Node

  Executes one child after another. It only returns the SUCCESS state if all child nodes run successfully. Otherwise, it returns false. Here an analogy would be the logical AND boolean operator.

- Parallel Node

  In this control flow node, the child nodes are ticked in the hierarchical fashion as usual. However, instead of waiting for a given child node's SUCCESS or FAILURE result, its successor is already ticked. The parallel node returns SUCCESS if a predefined number of child nodes return SUCCESS or FAILURE in case a predefined number or all child nodes return FAILURE.

Autonomy Decorator Nodes

The most important decorator nodes are:

- Inverter Node

  Inverts the output of an action node. A boolean analogy would be the "!" (not) operator.

- Repeat Node

  Repeats a node a number of times until that child node fails or a timeout is reached.

- Retry Node

  It is similar to the repeat node loop, but instead of repeating a successful child node, it retries a failing child node a number of times until it succeeds or a defined timeout is reached.

- Timeout Node

  Adding a timeout to an action node that otherwise wouldn't be temporally limited.

Autonomy Action Nodes

Many actions are required for the various subtask a rotorcraft has to perform during a mission. The most important ones for the landing behavior in this work are the following:

- ChangeAltitudeAction

  Changes the drone's altitude to the given value. The ascend/descend velocity diminishes upon reaching proximity to the desired waypoint.

- HoldPoseAction

  Self-explanatory: the drone holds the current pose for a given time.

- LandingAction

  Similar to the ChangeAltitudeAction, it descends to a waypoint, which in this case is simply the ground vertically below the drone's current position. Upon reaching a certain proximity to the landing point, the descent velocity is reduced to a minimum to accomplish a smooth landing.

- NavigateToWaypointAction

  Moves the drone laterally to a given waypoint by continuously interacting with the flight controller. It uses the same proximity-based slow-down mechanism as ChangeAltitudeAction and LandingAction.

- RotateTowardsWaypointAction

  Rotates the drone to face the given waypoint.

Behavior Tree Example

An example of a simple behavior tree is shown in fig. 3.20.

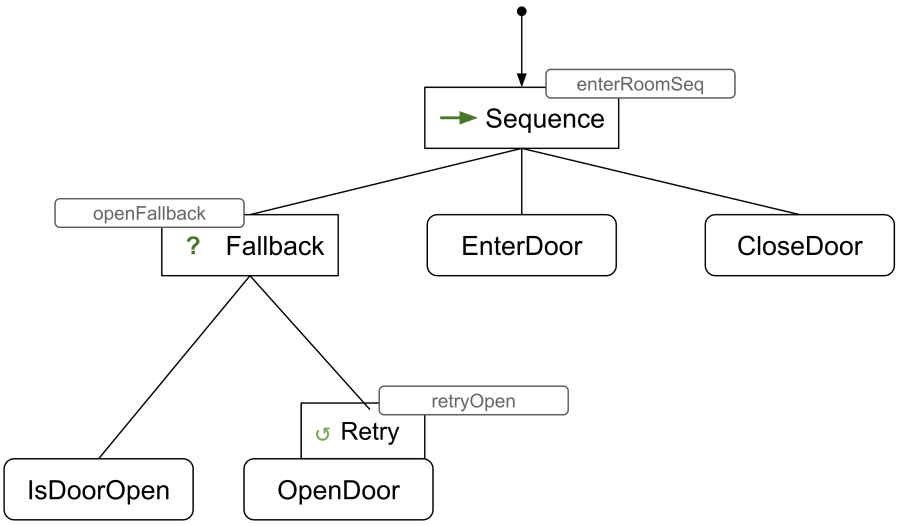


Figure 3.20: Simple, often used behavior tree example of entering a room

Big, rectangular boxes indicate control flow nodes, boxes with round edges are action nodes, and small rectangles attached to action nodes are decorator nodes. The names of the individual control flow nodes are attached to distinguish them from each other.

In this example, the root node, depicted as a black circle, initiates a sequence node. The sequence node executes all of its child nodes one by one. The first is a fallback node which checks whether the door is already open using the IsDoorOpen action. If it returns successfully, the fallback node concludes the state success, and the rest of the sequence node can be executed. If not, the tree repeatedly attempts to open the door using the Retry decorator node. Either the door opens or the fallback node and, therefore also, the sequence node fails. Lastly, if the door eventually opens, the other actions of entering and closing the door can be performed.

Chapter 4

Methodology

As this thesis's endeavor was to merge and enhance various aspects of the LORNA project, the complexity lay rather in understanding the existing work and its interfaces as opposed to the challenging methodology pursued in the novel contributions. Therefore, the implementation aspect of this work carries more weight than the theoretical decision-making associated with it.
The autonomy framework[21] allows us to fly independent missions at cruise altitudes of 100+ m. The structure from motion approach captures 3D information during traversal as its adaptive baseline allows it to perceive high-quality depth information at such high altitudes. This information can be used by LSD to detect landing sites during missions.
At low altitudes, SFM works as well, but surrounded by obstacles, the need for lateral motion poses significant risk. A local state estimator is, by nature, prone to accumulating an estimation error, and the same holds for the structure from motion algorithm. Our terrain knowledge can thus become void.
To overcome this issue, a range sensor can be used. An obvious consideration is LiDAR. However, a LiDAR sensor produces rather sparse point clouds unless a newer sensor, like Ouster's OS1-128[1] sensor, is used. In that case, there remains the weight issue with the sensor's 495g. As the drone is going to fly in Mar's very thin atmosphere, this isn't feasible.
Staying with the project's theme of visual sensors, a stereo camera poses a solution as it offers in-place triangulation with a very low weight ($\approx 10g$). Therefore, in this thesis, I present a stereo camera range node implementation to remedy the shortcomings at low altitudes.

4.1 Stereo Camera

The implementation of the stereo camera sensor itself is very straightforward as simply duplicating an existing camera, offsetting it an adequate distance to resemble the real model, and setting the parameters to equal the hardware, results in the desired outcome.
The simulated camera sensors used for this had the following properties:

- Image width: 640 pixels
- Image height: 480 pixels
- fx: 275.42px

[1] https://ouster.com/products/hardware/os1-lidar-sensor

- fy: 276.27px
- cx: 342.22
- cy: 234.66

The input to the stereo camera depth node is the two camera images and the drone's base link pose. Processing this information is different from the existing SFM algorithm, so a new depth generation node was implemented.
As mentioned in chapter 2, state-of-the-art deep learning-based stereo depth methods have considerably higher computational overhead. Due to the embedded CPU's computation limitations, this restricts us to using classical algorithms such as OpenCV's widely used implementations of Heiko Hirschmüller's approach ([22]).
The initial goal of the stereo camera implementation of this work was to show proof of concept for this approach of depth detection without lateral motion. Due to personal experience with the OpenCV library, the initial choice of stereo depth method was OpenCV's StereoSGBM algorithm. This algorithm is introduced in section 5.2.3.
JPL has its own visual library, JPLV, containing a stereo-matching method already in use for SFM, that will likely be a better choice for the future of the stereo depth node as well. This is because of three reasons:

- JPLV has a faster implementation which is beneficial for Mars missions flown with a limited processor.
- JPLV supports the CAHVORE camera model JPL uses for its visual applications. OpenCV, however, does not.
- Lastly, JPLV was created and is maintained in-house, which is always preferable to open source solutions.

However, for the proof of concept of the contributions in this work, especially using camera images without distortion, there was no specific reason to switch away from the preliminary StereoSGBM implementation. Therefore, the continuation thereof was pursued.

4.1.1 Stereo Camera Advantages

The specific advantage of a stereo camera implementation when compared to SFM can be summarized in the following points:

- No necessity of lateral motion
- Hardware depth perception
- DEM conversion
- Efficiency

Lateral Motion

As already mentioned above, lateral motion is an undesirable necessity for a rotorcraft in unknown terrain.
In this setup, the structure from motion approach is based on a keyframe buffer, which needs to be filled with image-pose pairs at different horizontal positions in order to start acquiring depth information. The current setting in the implementation Domnik et al. [17] uses 6 keyframes. Therefore, for a single point cloud, it is necessary to move laterally 6 times to start perceiving depth. With a necessary

baseline of 0.23[2] m, that would be over a meter before SFM start collecting data. Note that, As described in section 3.2.2, LSD needs to converge for some time before high quality landing sites can be detected. Therefore, even more terrain would need to be traversed. We would like to avoid this necessity when flying the rotorcraft at such low altitudes.

Software vs Hardware Depth Perception

Structure from Motion, a software node that relies on camera poses supplied by a state estimator, is, by design, subject to inaccuracies. A depth node based on a stereo camera, on the other hand, works with a fixed, rigid baseline between the camera views. Thus, for low-altitude flights that bear the danger of collision, a more robust hardware approach is preferred.

DEM Conversion

At low altitudes the image footprint of the tracking camera is only a few meters in each dimension. Therefore, to repeatedly observe the same terrain and let the measurements converge, the drone, using SFM, would have to fly forth and back over the same area. With a stereo camera implementation the drone can simply hover above the potential sites to gather this information.

Efficiency

All in all, the SFM setup allows us to detect a landing site at course altitude and after traversing horizontally to that location, we can simply descend to a stereo camera friendly altitude to hover for the verification. Therefore, simply descending to a low altitude and hovering for a short duration is significantly more efficient than flying lateral patterns at different altitudes above a landing site.

4.2 Ground Truth Depth

For evaluating the stereo depth node as well as proof of concept aspirations of the autonomous landing behavior introduced hereafter, a ground truth is required.
Additionally, GT was important because, at the time of this work, the structure from the motion node showed frequent signs of unreliability. See section 7.1 for the evaluation thereof.

4.2.1 Ground Truth Implementation

The simulation already supplied the ground truth pose of the drone's base link through the ROS bridges. When applying the static camera transform to it, this yielded the ground truth camera pose. Using Gazebo's depth camera sensor[3], a ground truth point cloud could be created.
The depth camera creates the image using traced rays, which fill a pixel with the centermost range value that a ray detected.
Looking at fig. 4.1, we can see that the ground truth point clouds yield very clean and easily interpretable LSD DEMs. fig. 4.2 shows the reference view of the terrain in the Gazebo simulation.

[2] Calculated by using the formula introduced in eq. (5.4) at 2.5 m altitude, a focal length of 275.42, a subpixel disparity error of 0.5 and an acceptable depth error of 10 cm

[3] As there was a bug in Gazebo's source code, the depth camera couldn't be used out of the box. More on this in chapter 13.

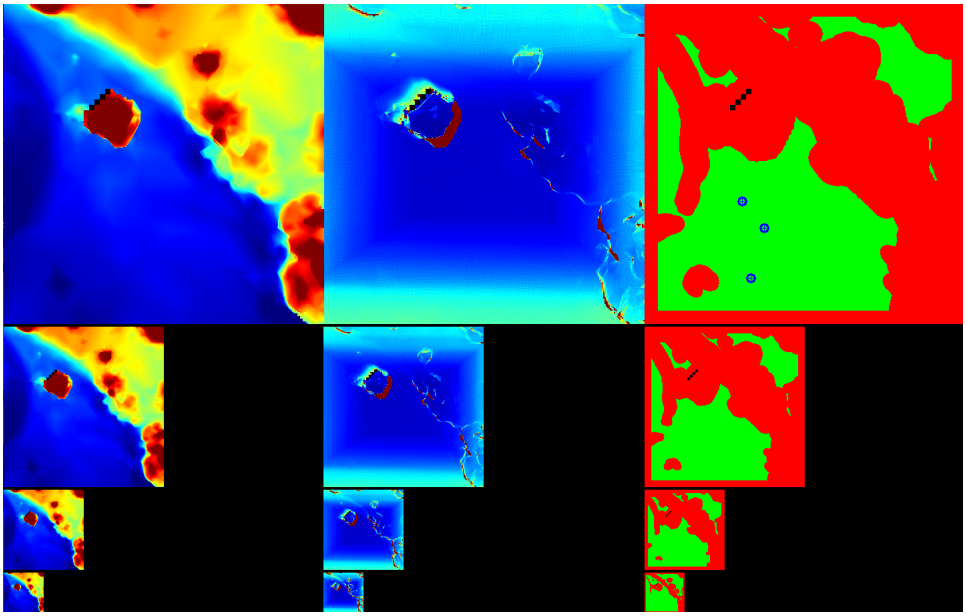


Figure 4.1: LSD debug image using a GT point cloud

Figure 4.2: LSD debug image simulation terrain reference

4.2.2 Comparability to SFM

This is sufficient for a qualitative analysis of the stereo-depth node. However, to use the ground truth as an alternative for SFM, one must ensure that the GT quality is not too good. For instance, SFM, like the stereo camera depth, has a depth error associated with its point cloud creation. At high altitudes, this can lead to the neglect of small (but mission-threatening) rocks. So, in order to test the autonomous landing pipeline with ground truth, I had to make sure that small rocks of about 10 cm in diameter were not seen by the GT at a cruise altitude of 100 m.
For this, the test setup shown in fig. 4.3 and fig. 4.4 was used.
On the Arroyo map, three cylinders of different sizes were placed. On each cylinder a small rock of 10 cm diameter was placed. The drone was then flown over the test setup at an altitude up to 100 m to detect the scene once with SFM and once using the GT depth. The goal was to find out, whether the ground truth depth quality would have to be artificially decreased, using Gaussian or median filtering for instance, in order to make it more comparable to SFM.

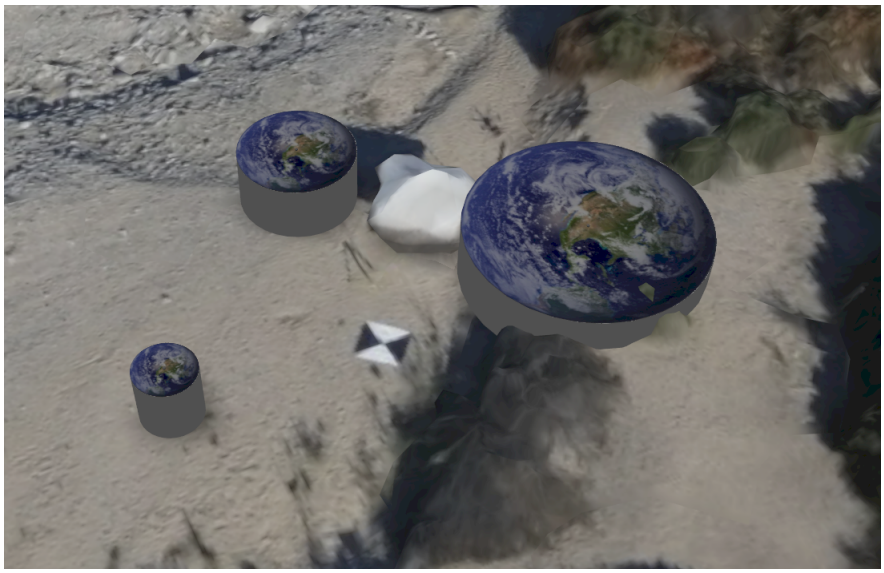


Figure 4.3: GT test setup1: Three cylinders of different sizes (4, 2, and 1 m diameter) were placed on the map

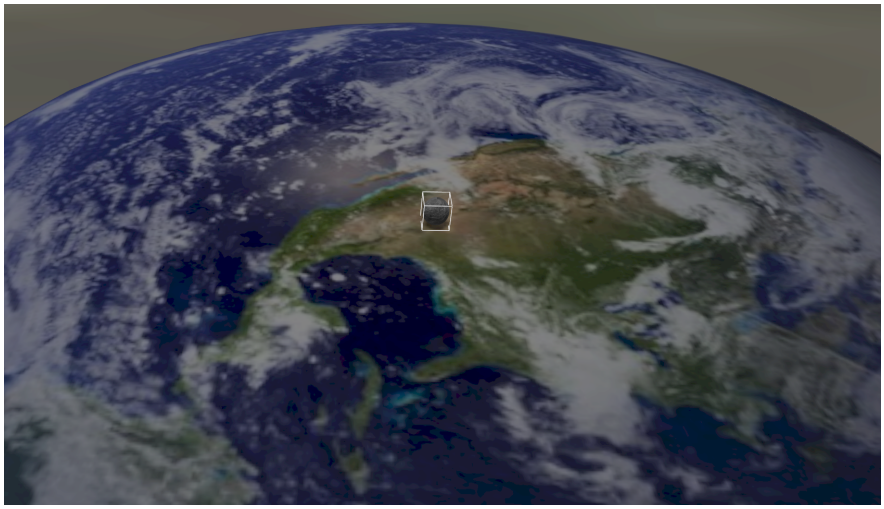


Figure 4.4: In the middle of each cylinder, a rock of 10 cm diameter was placed

Looking at fig. 4.5 and fig. 4.6, one can see that, though the SFM quality is visibly worse, neither SFM nor GT detected the small rocks on the platforms. This can be seen because in both Debug images, the center of the platforms was considered a landing site even though there was the rock, which should definitely prevent the detection of a valid landing site in the very center.
Therefore, the ground truth depth could be used to test the autonomous landing pipeline without making a landing site verification step at low altitudes redundant.

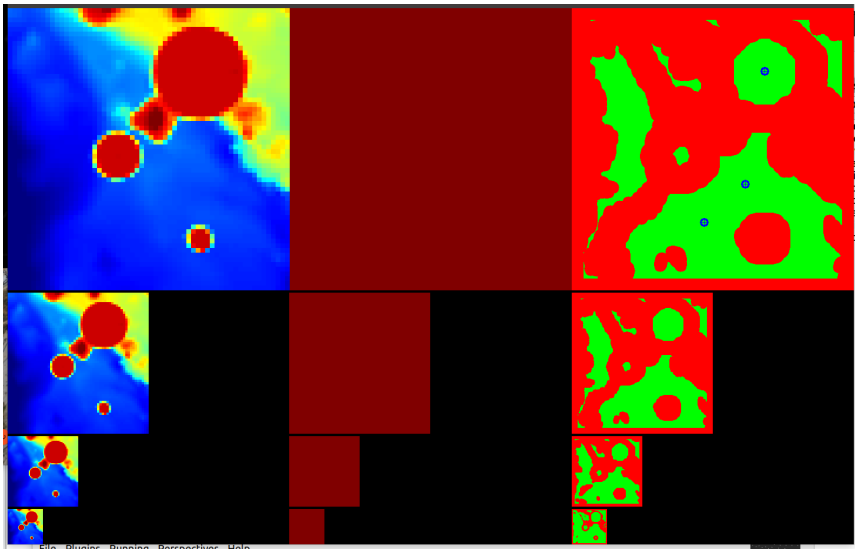


Figure 4.5: Ground truth result of the GT test

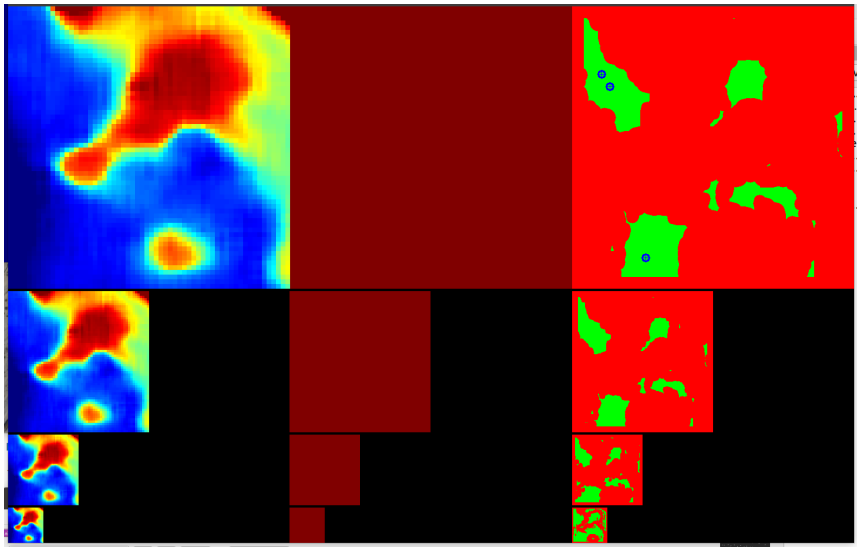


Figure 4.6: SFM result of the GT test

4.3 Autonomous Landing Procedure

After implementing a stereo camera as a low-altitude alternative to SFM and ensuring a correct ground truth comparison, the main contribution of this work could be tackled: bringing the visual landing site pipeline together with the autonomy framework to achieve reliable autonomous landing in unknown terrain.
The manner of implementation for the autonomous landing sequence was more or less given by the system architectures that this work combines. The two crucial points of the methodology were the interface between the autonomy framework and the landing site detector and the implementation of the decision-making in the behavior tree of the landing node as well as the landing site manager.

4.3.1 LSD - Autonomy Interface

Both the autonomy and the landing site detection node had to be altered for the interface with autonomy.

- Landing site detection output

  Prior to this work, LSD only published the location of one landing site. To give the autonomy more information to make adequate decisions and to avoid stagnation on a single overconfident landing site, LSD was changed to first publish three landing sites each iteration and secondly yield additional characteristics with each output landing site.

- Landing site interface of the autonomy

  The autonomy framework was changed to correctly receive and handle the incoming landing sites with all the newly associated properties. Furthermore, novel landing site handling concepts like re-detection and banishment were introduced.

4.3.2 Autonomous Landing Behavior

As previously explained, the landing behavior in the autonomy framework is implemented using a behavior tree consisting of small modular and expandable actions. The existing BT framework and the available actions prior to this work are shown in section 3.3.4.
The two paramount qualities in pursuit of which the landing behavior was designed are safety and efficiency. Naturally, safety is the first-most concern. We want a reliable autonomous landing procedure for the drone to repeatedly land safely. However, the more efficient it is, the more daring science missions can be designed and the faster the red planet can be explored in detail.
For the conceptual design of the autonomous landing behavior, a fail-safe dogma was adopted, meaning that the drone never moved both horizontally and vertically. Instead, any required traversal was preceded by an adequate ascent to a safe height and followed by a vertical descent to the desired altitude at the new location. Secondly, the available landing site characteristics were used to enhance the overall efficiency.

Chapter 5

Stereo Camera Depth Node Implementation

5.1 Theoretical Analysis

The obvious drawback of a stereo camera when it comes to depth perception is its limited baseline. It only perceives depth accurately for objects within a certain proximity to the lens.
Even assuming a perfectly calibrated and rectified camera, the depth estimation is always inaccurate because of disparity errors.
The formula to derive a depth value from a calculated disparity is:

$$z = \frac{f \cdot b}{d} \tag{5.1}$$

Where b is the z is the depth estimate, b is the baseline, f is the focal length, and d is the disparity value.
Taking the derivative of z w.r.t. d, we get

$$\frac{\partial z}{\partial d} = -\frac{f \cdot b}{z^2} \tag{5.2}$$

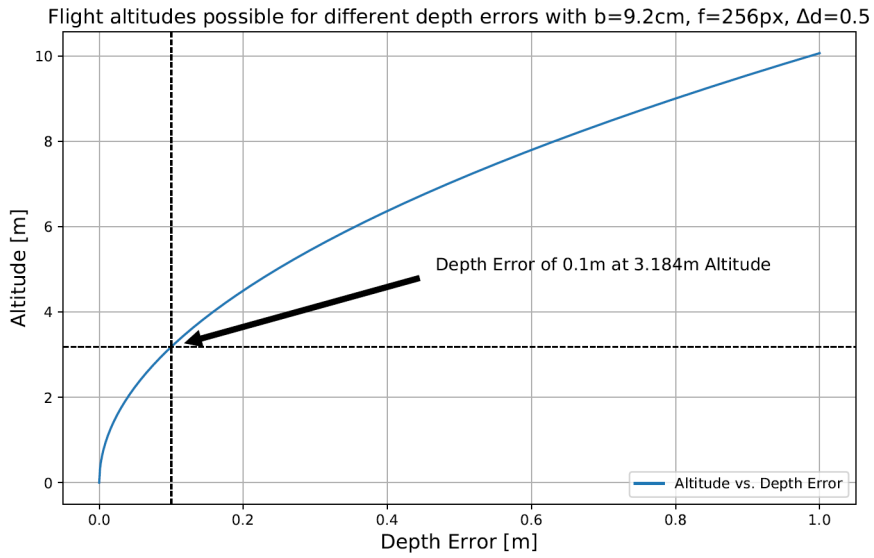
And substituting (eq. (5.1)) we get:

$$\partial z = \frac{z^2}{f \cdot b} \partial d \tag{5.3}$$

Where the sign was left away, as for our application, there lies equal danger in a point being perceived as too close and too far away.
For the maximum altitude given a maximum allowable depth error, this yields:

$$z_{\max} = \sqrt{\frac{\Delta z_{\max} \cdot b \cdot f}{\Delta d}} \tag{5.4}$$

Δz is the depth error and Δd is the disparity error.
The simulated stereo camera in Gazebo had a focal length of 275.42 pixels, and according to the physical stereo camera used on the drone at JPL, a baseline of 0.092 m was used.
With these properties and estimating a subpixel-precision disparity error of 0.5 pixels, the depth error at varying altitudes looks as follows:
Let's assume we allow a maximum depth error of 10 cm. Considering this constraint, we can fly at a maximum altitude of 3.184 m as indicated in section 5.1.



This limitation must be considered. However, it is neither too surprising nor restrictive, as the stereo camera is simply a depth alternative for low-altitude flight maneuvers. In the context of an entire science mission, it is almost exclusively used for landing site verification purposes.

5.2 Implementation

Like Structure from Motion, the stereo depth instance is a ROS node supplied with images and image poses from the xVIO state estimator. As the state estimator was in its final development stages during my thesis, camera images and a ground truth camera pose from the simulation were used instead as input for the stereo algorithm. Note that only one camera pose is given as the second one is derived in a straightforward manner, given the fixed hardware baseline.

5.2.1 Stereo Setup Overview

fig. 5.1 shows the drone setup in the simulation with the stereo camera. The stereo camera pair is indicated with the opaque boxes. The large distance to the drone's core is necessary to avoid capturing the landing feet in the image due to the simulation model's discrepancy with the physical drone. As presented in chapter 12 the drone hardware has landing skids that are spread significantly farther apart than for the simulated model. That is why, when using the stereo camera mounted at the core of the physical drone, the mainstays are not visible in the images detected. In the simulation, they would be detected unless the stereo cameras were positioned further from the rotorcraft's center as shown in fig. 5.1.

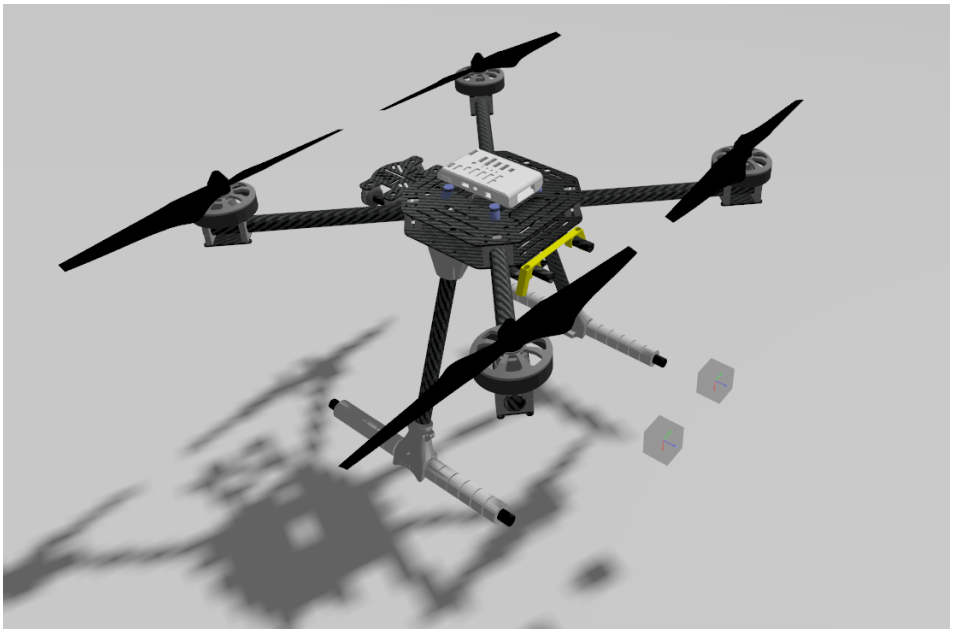


Figure 5.1: Stereo camera on drone indicated by opaque boxes

Frames

A critical part of navigation is always the consistency of the coordinate systems in which quantities are represented, hereafter in fig. 5.2 the present coordinate systems of the stereo camera setup are displayed.
Notably, there are three important frames:

- The reference world frame W

  This is the global frame relative to which the drone flies, and the point clouds are created.

- The drone (base link frame) D

  This is the pose of the moving drone throughout a mission. It is constantly published by the simulation.

- The camera pose C

  The camera pose is the frame in which the point cloud is created and aggregated. The relative pose of the cameras is set in the simulated drone's setup file. This transformation is static and is applied to each incoming pose message directly.

Hereafter, the following notation is used:

- t_{dc}^{W}: Transformation from drone to camera, represented in the World frame
- R_{DC}: Active Rotation from the Drone to the Camera frame
- r_{wp}^{W}: Position vector from the world frame to the detected point, represented in the World frame.

One more thing to note in fig. 5.1 is that in Gazebo's camera convention, the optical axis of a camera points along the x-axis. Therefore, the base link pose was

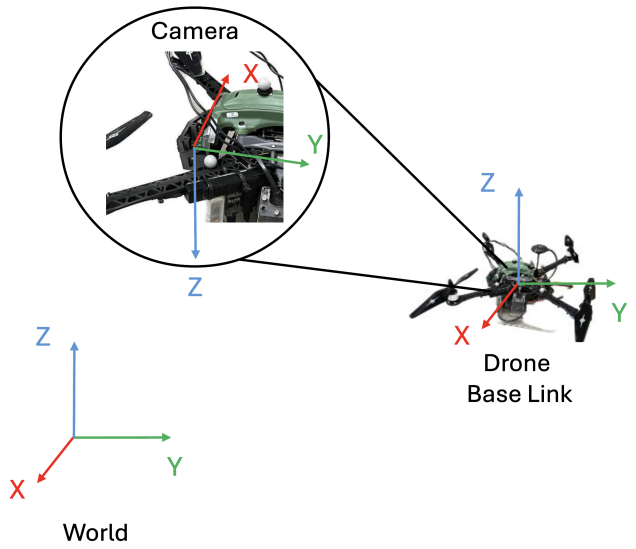


Figure 5.2: Coordinate frames in stereo camera depth setup

subscribed to and converted to the camera pose using the respective transformation. Neither Gazebo's base link nor camera coordinate conventions are relevant as long as we correctly track the pose of a downwards-facing camera with the correct convention.

5.2.2 Input Handling

The input of the images and the base link pose are received using ROS subscribers. Despite publishing these simulated Gazebo topics at equal rates, they did not arrive simultaneously. In the completed setup using SFM this is handled by supplying SFM with both the image and the pose from the xVIO state estimator. As stereo needs two images from the stereo camera, this will have to be handled manually either by making xVIO also publish the stereo images or having some other way to ensure a synchronized delivery of the stereo images and the xVIO pose.
The pose is only required for transferring the created points from the camera frame to the world frame. Therefore, the two input images were processed into a depth image in a single step, and only then were all three inputs used together to create the point cloud.
fig. 5.3 schematically shows how the stereo camera depth node handled this shortcoming of asynchronous sensor messages by manually picking the pose which temporally closest corresponds to the image's timestamp. This was possible as the image processing step dominates the computation time of the input handling, and the pose can thus be updated in the meantime to best fit the image timestamps.

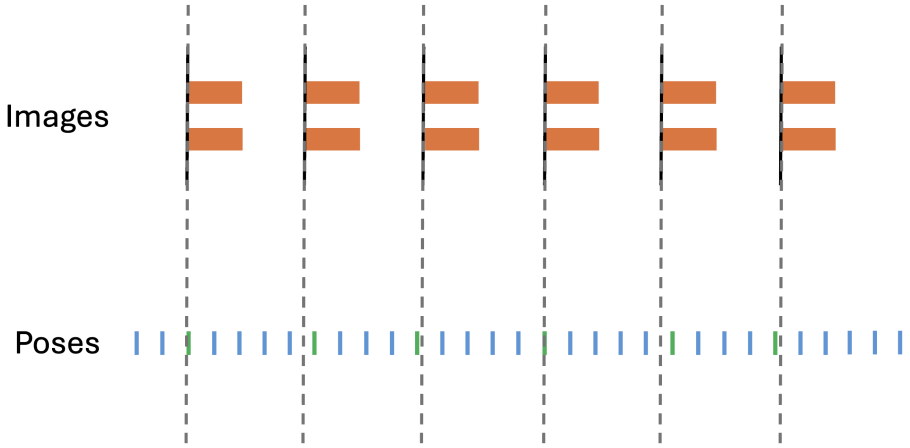


Figure 5.3: Schematic visualization of input synchronization used in the stereo depth node

5.2.3 Disparity Creation

The chosen StereoSGBM (Semi-Global Block Matching) algorithm creates a disparity map from two horizontally aligned images using an approach based on Heiko Hirschmüller's stereo approach introduced in [22].
The resulting disparity image is shown in fig. 5.4.
The StereoSGBM algorithm cuts off the left boundary, as can be seen in fig. 5.4. This is due to the maximum disparity parameter, which prevents the pixels on the left border of the left image from being matched. As the left image is the reference image, the output removes the border. Additionally, artifacts could be seen on the right-hand side of the image. Both of these border artifacts were filtered out using a mask.

5.2.4 Point Cloud Creation

Having created a disparity image using the approach laid out in section 5.2.3, the disparity pixels are first converted to the depth values using the classic disparity depth relation 5.1 shown in section 5.1.
The 3D locations of each detected point can be derived from the created depth image, pose, and camera parameters. This is done by simply tracing the projection line of the detected point as indicated in fig. 5.6 and using the similar triangles as shown in fig. 5.7
The formulas for the point coordinates in the camera frame are shown in eq. (5.5).

$$x_{cp}^{C} = \text{depth} * \frac{(u - o_x)}{f_x} \tag{5.5}$$

$$y_{cp}^{C} = \text{depth} * \frac{(v - o_y)}{f_y} \tag{5.6}$$

$$z_{cp}^{C} = \text{depth} \quad \text{Depth value is the same} \tag{5.7}$$

Lastly, to get the point cloud in the world frame, the coordinate transform is applied:

Figure 5.4: Disparity image from the stereo camera using StereoSGBM.

$$r_{wp}^{W} = r_{wc}^{W} + R_{WC} \cdot r_{cp}^{C} \tag{5.8}$$
$$= r_{wd}^{W} + R_{WD} \cdot r_{dc}^{D} + R_{WC} \cdot r_{cp}^{C} \tag{5.9}$$

Where

$$r_{cp}^{C} = \begin{pmatrix} x_{cp}^{C} \\ y_{cp}^{C} \\ z_{cp}^{C} \end{pmatrix} \tag{5.10}$$

The final point cloud created using this process is shown in fig. 5.8.
The node's final output is a point cloud generated in the world frame, together with two poses representing the camera locations of the generated point cloud.

Figure 5.5: Stereo depth image

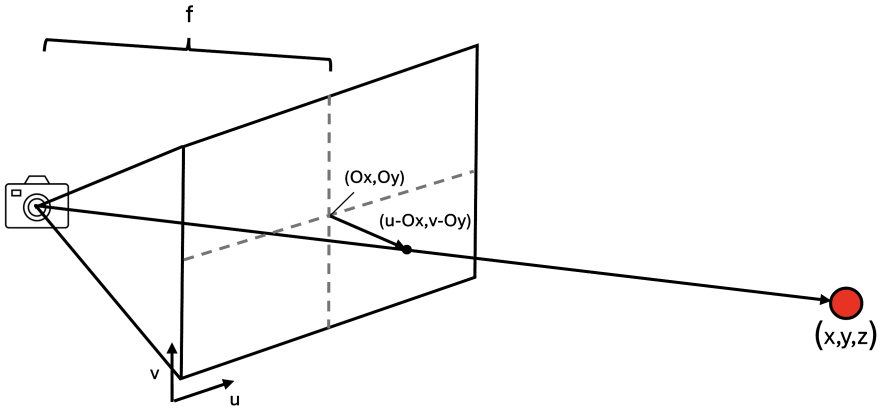


Figure 5.6: Schematic of the line projection procedure to derive the 3D location of detected points

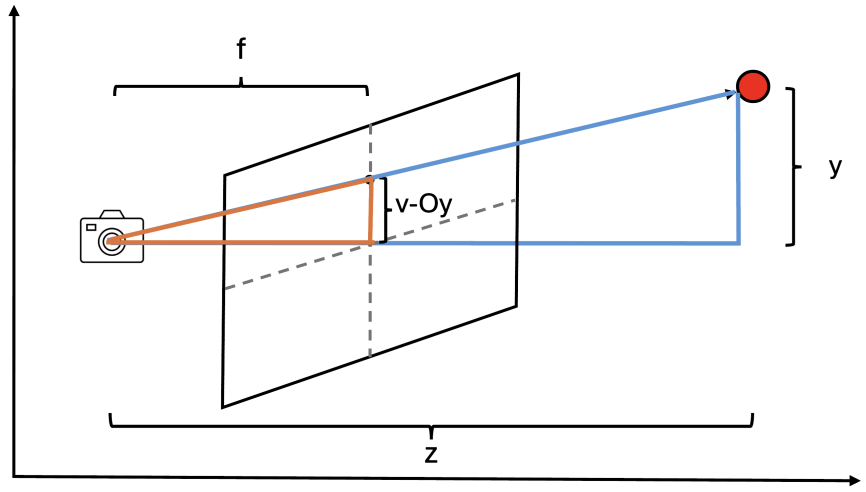


Figure 5.7: Schematic of the line projection procedure to derive the 3D location of detected points

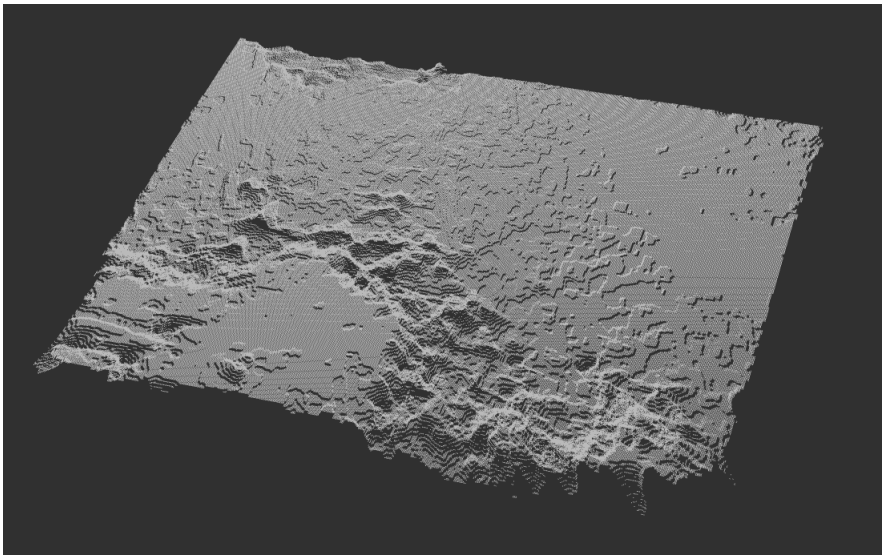


Figure 5.8: RViz visualization of stereo camera point cloud

5.2.5 Switching

One needs to switch between the two alternatives to achieve the final desired perception mechanism of flying laterally with SFM and using a stereo camera depth node at low altitudes.
The drone's current altitude above ground is the obvious decision boundary to implement in the switching mechanism. This could be achieved by analyzing the generated point cloud at a given iteration to determine the median altitude, which indicates the altitude above ground. However, this is an avoidable computational overhead.
As mentioned in chapter 3, the drone has a laser range finder on board. This allows us to estimate the altitude above ground at any given moment without needing image processing.
Therefore, the switching is performed using a separate ROS subscriber, which continuously checks the LRF's measurement and activates or deactivates the SFM node and stereo node, respectively. This is schematically shown in fig. 5.9.[1]

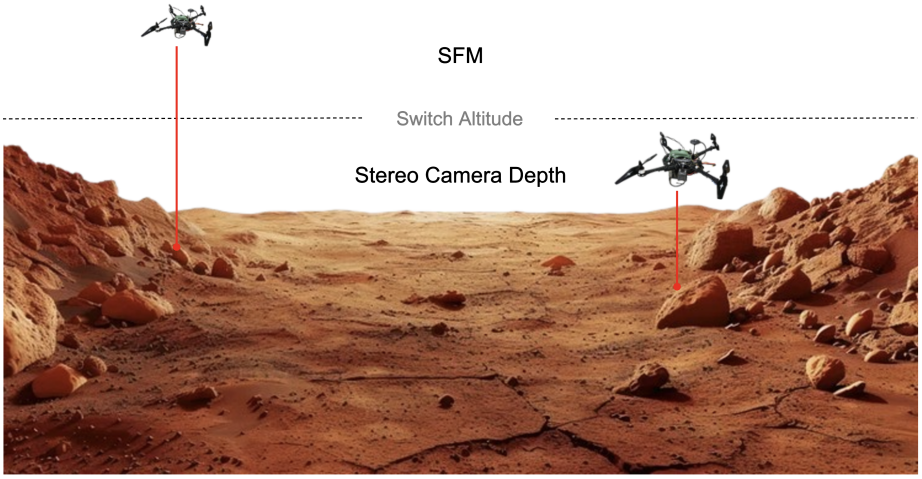


Figure 5.9: Laser Ranger Finder Based Switch between Depth Sources

A second application of this switching is to disable the stereo node until the drone reaches approximately 50 cm altitude. The reason for this is that the camera images don't have sufficient overlap to detect matches for the closest points with the largest disparities below that altitude. This is because the StereoSGBM algorithm is implemented with a maximum disparity of 80 pixels. This is shown in fig. 5.10.

[1] BG image source: Adobe Stock images

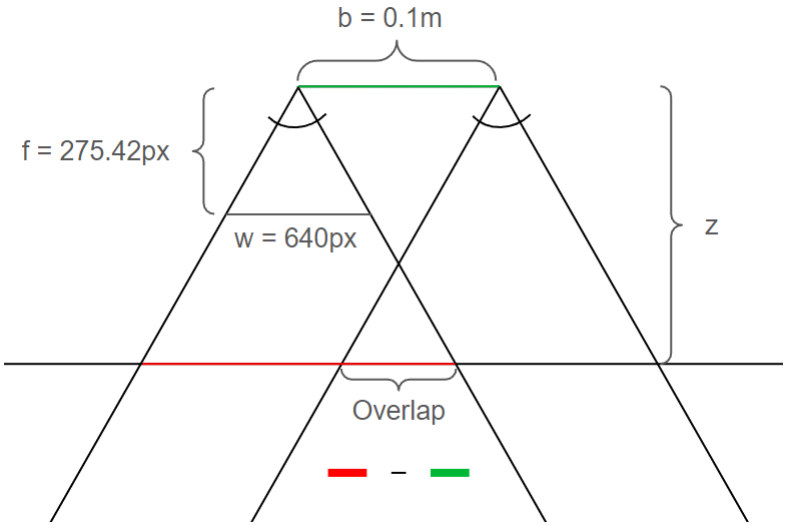


Figure 5.10: Overlap problem at low altitudes: The baseline is indicated in green, and the image footprint in red. The overlap of the two camera images is the reference camera's image footprint minus the baseline.

With the given camera parameters shown in section 4.1, the metric image overlap is calculated to be.

$$overlap = 2 \cdot \frac{w}{2 \cdot f} \cdot z - b = \frac{w \cdot z}{f} - b = \frac{640 \cdot z}{275.42} - 0.1m \tag{5.11}$$

$$overlap = \frac{d \cdot z}{f} = \frac{80 \cdot z}{275.42} \tag{5.12}$$

eq. (5.12) shows the minimum image overlap required for a possible maximum disparity of 80 pixels. Plugging this into eq. (5.11) and solving for z, we get:

$$z = \frac{b \cdot f}{w - d} = \frac{0.1m \cdot 275.42}{640 - 80} = 0.04918m \tag{5.13}$$

Therefore, the drone must be at least 50 cm in the air to perceive depth well. Note that this is rather a high minimum altitude. This is because the simulated cameras have rather small focal lengths. The trade-off with smaller focal lengths is the ability to use stereo higher above the ground.
For instance, a camera pair with a focal length of 512 pixels and the otherwise same setup would only need a minimum altitude of 9.14 cm. However, the altitude at which a depth error of 10 cm is already at 2.347 m as opposed to the derived 3.184 m shown in section 5.1.

5.2.6 Landing Site Detection without Lateral Motion

Taking off vertically with the drone in the simulation, the first landing site in the LORNA project without lateral motion was found. fig. 5.11, fig. 5.12, and fig. 5.13 show the drone in the simulation, the generated stereo point cloud and the landing site detected in the LSD debug image respectively.

Figure 5.11: Drone during vertical ascent in simulation

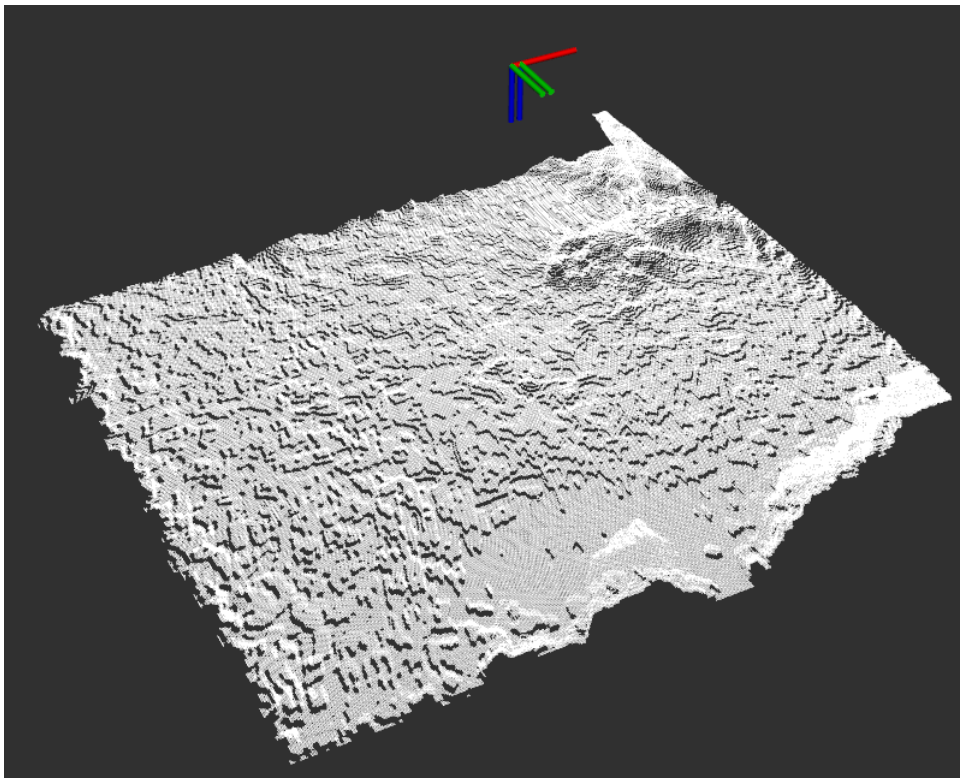


Figure 5.12: RViz visualization of created point cloud from stereo camera

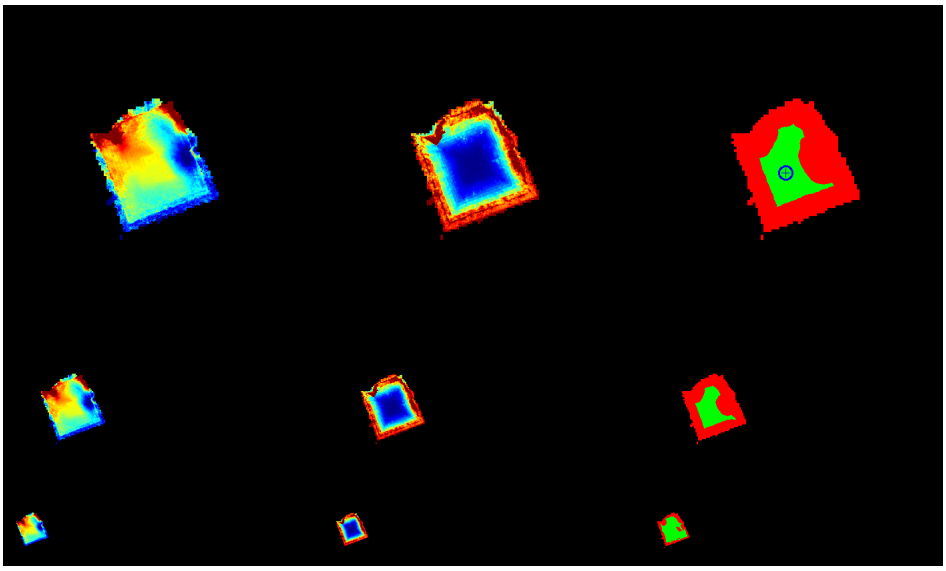


Figure 5.13: LSD Debug output displaying LORNA's first detected landing site during vertical motion

5.3 Qualitative Practical Analysis

Once implemented, the landing site detection instance could be supplied by the stereo depth node. The result thereof can be seen below:

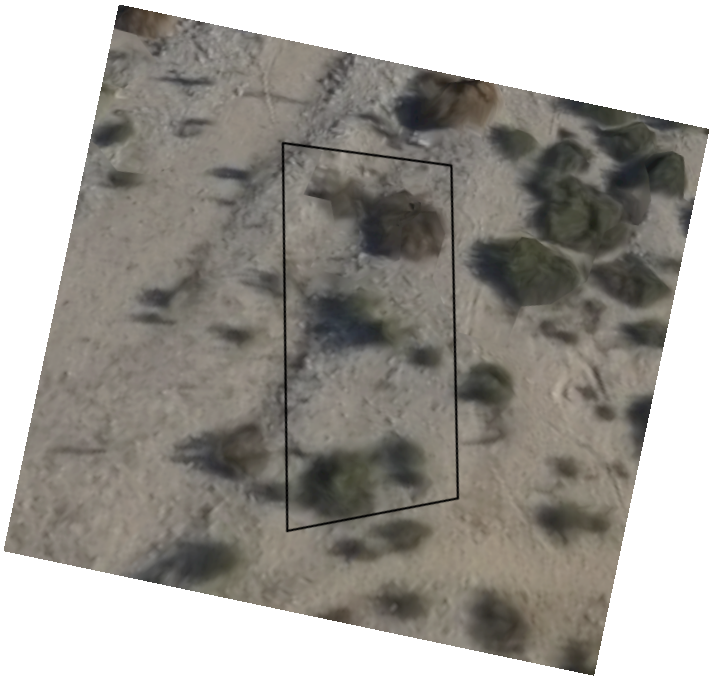


Figure 5.14: Considered terrain patch in Gazebo simulation

When comparing the result to the ground-truth LSD output, it can be seen that LSD creates a very accurate DEM from the stereo camera depth input. The landing sites detected are reasonable compared to the terrain reference.

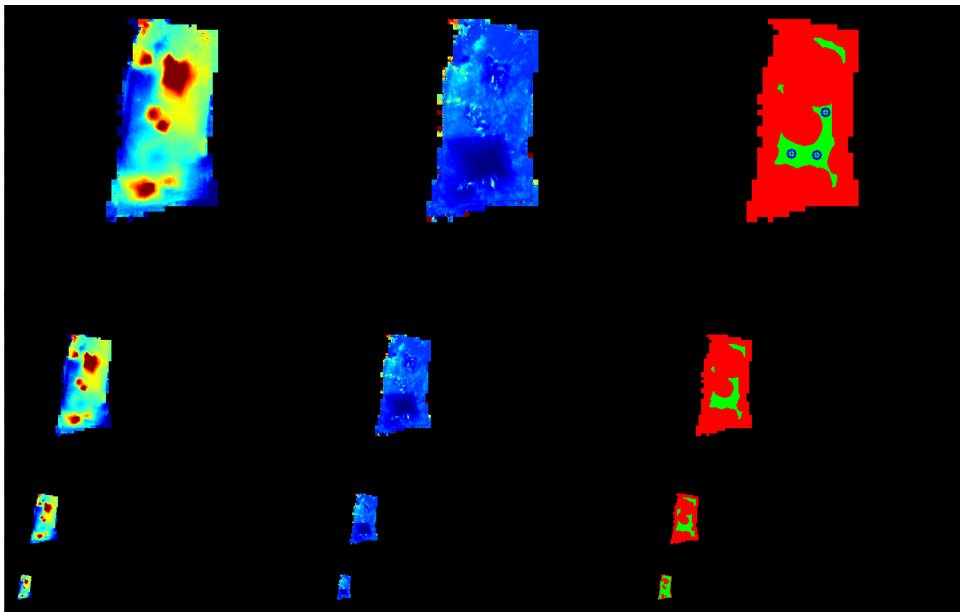


Figure 5.15: Stereo camera depth supplied LSD debug image at 2.5 m altitude

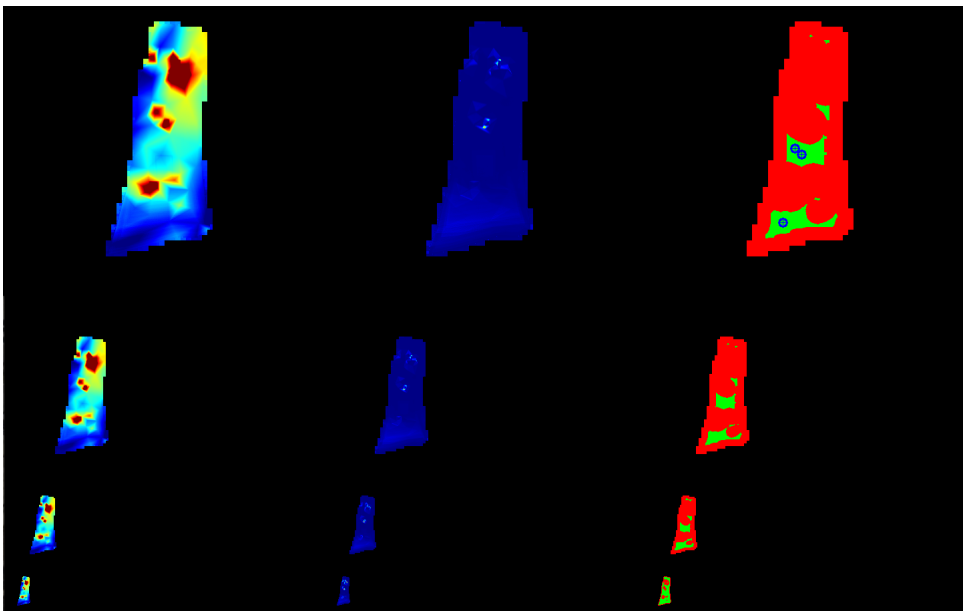


Figure 5.16: GT depth supplied LSD debug image at 2.5 m altitude

The stereo camera has a relatively small, fixed baseline, so its usage domain is restricted to low altitudes. Therefore, the residual part of a mission is still flown using SFM.

Chapter 6

Autonomous Landing Procedure

The implementation of the autonomous landing procedure consisted of two parts. First, a sound interface between the autonomy and LSD was established, ensuring the autonomy's supply of quality information from LSD and the adequate processing thereof.
Secondly, the adaptive landing behavior itself was implemented.

6.1 LSD - Autonomy Interface

In order to make high-quality landing decisions in the autonomy framework, the landing site detection algorithm must send the system high-quality information.

6.1.1 LSD Properties

Before this work, the output of the landing site detection algorithm was merely the location of a found landing site. However, as described in section 3.2.2, the landing site detection algorithm segments hazards based on roughness and slope. Thereafter, it considers the size of a landing site and the uncertainty associated with a certain selected location.
Therefore, simply outputting the location of a landing site is a waste of information when so many characteristics are available to make an informed selection.
I decided on the following properties to be the content of the LSD output:

- Location

  The location of the landing site in the world frame.

- Uncertainty

  The uncertainty value is also a product of the landing site detection algorithm. It denotes the normalized uncertainty across the area around a given landing site.

- Roughness

  The roughness value is the exact value already used for the hazard segmentation step in the landing site detection. See section 3.2.2 for an explanation of this property.

- Size

  To determine the size of a landing site, the landing site detection algorithm performs a distance transform on the created landing site map to find the closest non-landing site for any found landing site. This returns the radius of the largest valid landing circle around a landing site. When the physical value is calculated, the metric radius is returned as the size of a landing site.

- Obstacle Altitude

  The obstacle altitude was newly introduced in this work. It defines the current highest point of the aggregated DEM's highest resolution layer. As the LSD landing site segmentation uses the terrain more so to detect areas absent of obstacles, no actual object detection is performed, and no hazard information is retained in this visual pipeline; this value serves the autonomy as an indication of the obstacle heights to avoid in the vicinity of a certain landing site. For more on this, see section 6.3.1.

The final landing site detection output shown in fig. 6.1 is a custom landing site ROS message containing the above-mentioned characteristics of the detected spot. For more detailed explanations of these properties, see section 3.2.2 and section 3.2.2.

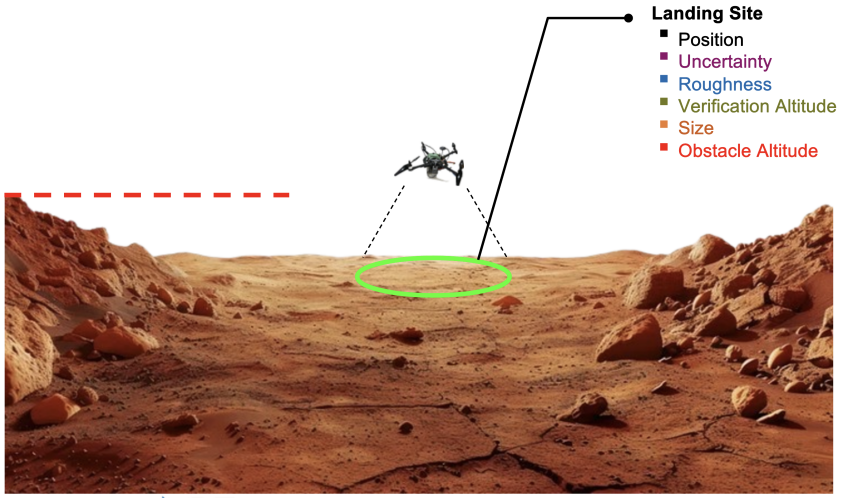


Figure 6.1: Schematic of new properties

Note: The supplied landing site from LSD already underwent a filtering procedure utilizing the roughness, slope, size, and uncertainty properties. The value of reconsidering them again in the autonomy is two-fold.
First, we have to remember that LSD does not use the current distance to the drone in its landing site segmentation. Therefore, for autonomy to be able to weigh the quality of a landing site (roughness, slope, size, uncertainty) against its convenience (distance to the drone), we need the complete set of properties at all times.
Secondly, a bigger handle in the weighting decision of these properties against each other is desirable. For instance, as mentioned in Proenca et al. [19], the uncertainty values rise significantly in monotonous areas. Thus, a reweighing of the properties is required for optimal performance.

6.1.2 Landing Site Heuristic

The autonomy processes the in section 6.1.1 listed values in order to arrive at the following properties:

- Current Distance to Drone L_{dist}

  Well-likely the single most important characteristic of a landing site[2]. Each iteration calculates the current distance to the drone's position for each retained landing site. The distance is then normalized by dividing it by the cruise altitude, which is 100 m. In practice, there were easily enough landing sites found while moving to allow landing sites to fall off when farther away than 100 m.

- Roughness L_{rough}

  The roughness property is the unaltered roughness value received from LSD. It is already normalized and enters the loss function as it is.

- Uncertainty L_{var}

  The same holds for the uncertainty. It is already normalized by design and enters the loss function unaltered.

- Size L_{size}

  Analogous to the roughness and uncertainty properties, size comes directly from the landing site detection. However, unlike the two preceding properties, it is not normalized but simply denotes the metric radius of the largest circle of valid landing area that can be fit around a given landing site. This is achieved in LSD by performing a distance transform on the created landing site image.

  To normalize this value, the maximum landing site size is retained, and each landing site's size is divided by it to achieve normalized size information.

  Also, as can be seen in eq. (6.1), the size contribution enters the loss function with a negative sign. This is because, compared to all other characteristics, the size defines a property we want to maximize.

- Verification Altitude L_{verAlt}

  A site's verification altitude is the smallest vertical distance between the drone and the landing site at which that site was (re-) detected.

  The verification altitude is a useful property because of numerous reasons.

  – Further Indication of Certainty
    First, similar to the uncertainty metric, the verification altitude indicates how certain we can be about a detected landing site, as spots detected at lower flight altitudes are more likely correct due to the reduced depth error. Even though it might seem to overlap with the uncertainty property in this regard, these two characteristics are quite complementary, as the uncertainty takes OMG convergence and camera specifics into consideration, while the verification altitude is a purely location-based metric.

[2]Each received landing site has already undergone a threshold filtering regarding slope and roughness.

- Landing Site Property Updates

  As the verification altitude yields a simple and good estimation of the trustworthiness of an incoming landing site, it can be used as a flag to know when a landing site's properties should be updated. When a landing site is re-detected with a verification altitude lower than the previously stored one, the algorithm trusts it more and alters the previously stored properties to the new ones received.

- Verification

  Continuously updating the verification altitude upon re-detection allows us to determine the lowest altitude at which a landing site was re-detected. This information can be used to verify that a given site was considered a valid landing spot even at low altitudes.

The final heuristic defining the quality of a landing site is, a simple square loss function:

$$L_{LS} = w_{dist}L_{dist} + w_{rough}L_{rough} + w_{var}L_{var} + w_{size}L_{size} + w_{verAlt}L_{verAlt} \quad (6.1)$$

6.1.3 Landing Site Manager

Before this thesis, the landing site manager received artificially generated landing sites from a dummy landing site node. This work connected the LSM to the actual landing site detection output. See section 3.3.2 for an introduction to the landing site manager.
With the new landing site properties introduced in section 6.1.2 the metric for evaluating a landing site is no longer a maximizing heuristic but a loss function to be minimized. Therefore, the simple switch of a min-heap to a max-heap was done to efficiently and consistently switch the worst landing site and order the incoming one.
Apart from the new heuristics and the handling thereof, the following important new concepts were introduced for the landing site manager:

Re-Detection

In the LSM's state before this thesis, each landing site was considered individually and processed. The worst landing site, according to the heuristic, was filtered out, and the new landing site was put in its place and then ordered into the min heap.
Re-detection defines the mechanism of assessing the proximity of an incoming landing site to the already considered landing sites in the buffer. Incoming landing sites close enough to previously detected sites are considered re-detections. In that case, the new landing site is not entered into the buffer; instead, the previously detected landing site has its properties updated.
However, this property update is only performed when the incoming landing site is detected at a lower flight altitude. Expressed in specific terms, this means that the landing site is only updated if the incoming landing site's verification altitude is lower than the previously detected one. This is because more trust is placed in landing sites detected closer to the ground.
The benefit of updating the characteristics lies in refining the site's information and, therefore, retaining a higher-quality landing site candidate.
The proximity within which an incoming site is considered a re-detection of another is determined by linear interpolation. At 100 m altitude, any landing site within 1 m of another is considered the same site. At the verification altitude, which was

set to 2.5 m during this work, the re-detection distance is 0.1 m. This threshold change implicitly accounts for the larger terrain errors made at higher altitudes. The schematic re-detection procedure is shown in fig. 6.2.

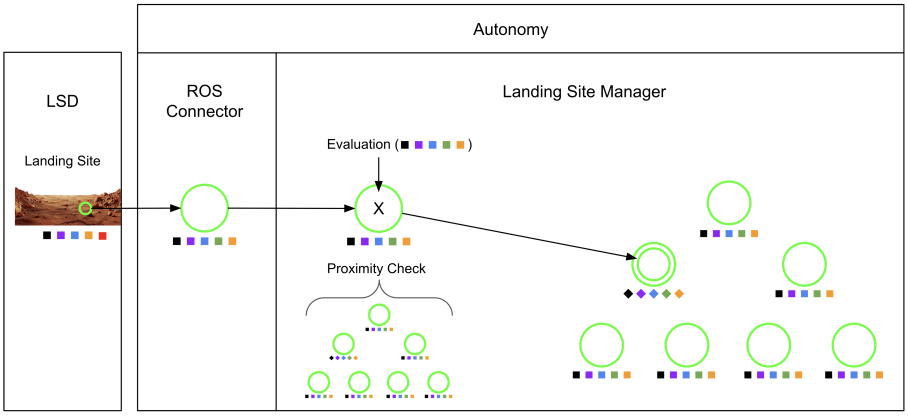


Figure 6.2: LS re-detection: When the proximity check is successful, the incoming landing site updates an existing landing site instead of entering the buffer itself.

The properties are indicated with the colored squares according to their introduction in fig. 6.1. Note that the obstacle altitude property is part of the LSD output, but as it is not part of the loss function, it is not considered in the LSM's ordering. Secondly, the verification altitude and the distance to the drone are calculated in the ROS connector.

Selection

In the already existing autonomy framework, the handling of landing sites was very similar. Landing sites were ordered, and upon entering the landing state, the best one was shared with the navigation nodes through the Blackboard variable interface. The selection happened, therefore, through the landing site manager signaling to the other nodes which landing site to navigate towards.
The landing site manager was changed in this regard to actively choose a landing site and update it consistently. This way, a handle on the current candidate exists. This handle is used for numerous benefits, including consistently updating the landing site upon receiving new information, verifying that exact landing site, and banning it in case of a verification failure.

Verification

As explained in section 6.1.3, when a landing site is re-detected, all properties are updated if and only if the verification altitude is lower than previously perceived. This results in a landing site always retaining the lowest altitude at which it was detected.
This promotes the choice of a landing site detected at low altitudes as mentioned in section 6.1.2. What's more, we can use this as a verification tool.
When pursuing a landing site that was detected at 100 m altitude, we need to validate it at a low altitude before committing to it and landing. We can do so safely as, yes, the initial landing site estimate from 100 m altitude might not be a good choice, however as can be seen in the depth error formula shown in eq. (5.4), given an approximate baseline of 15 m at 100 m altitude with a focal length of 256 and an assumed subpixel disparity error of 0.5, the structure from motion algorithm

yields depth measurements with an approximate depth error of 1.3 m. Therefore, when verifying a landing site at about 2.5 m altitude, the drone has sufficient buffer to potential terrain.
Thus, the verification consists of the low altitude hovering above a landing site, consistently scanning the terrain using the stereo camera, which constantly sends landing sites to the autonomy. After a verification timeout duration is reached, the verification altitude of the chosen landing site is compared to the hover altitude, and if they are within a small error threshold of each other, the site is considered verified, and landing is initiated.
If the verification fails, the chosen landing site has to be banned. This includes not only removing the landing site from the buffer, as it might be re-detected in the future. Instead, the landing site has to be entered into a ban list against which new landing sites are compared. If a re-detection of an incoming landing site with a banned one is triggered, the incoming landing site is ignored.
This mechanism excludes the possibility of repeatedly detecting and pursuing a promising false candidate.
The final procedure is shown below in fig. 6.3. A received landing site is checked for proximity to the selected landing site, the banned landing sites, and the current landing sites in the buffer. If it is close enough to one of these sites, it is either discarded in the case of the ban list or otherwise used for re-detection. If not, the landing site is entered into the LS buffer.

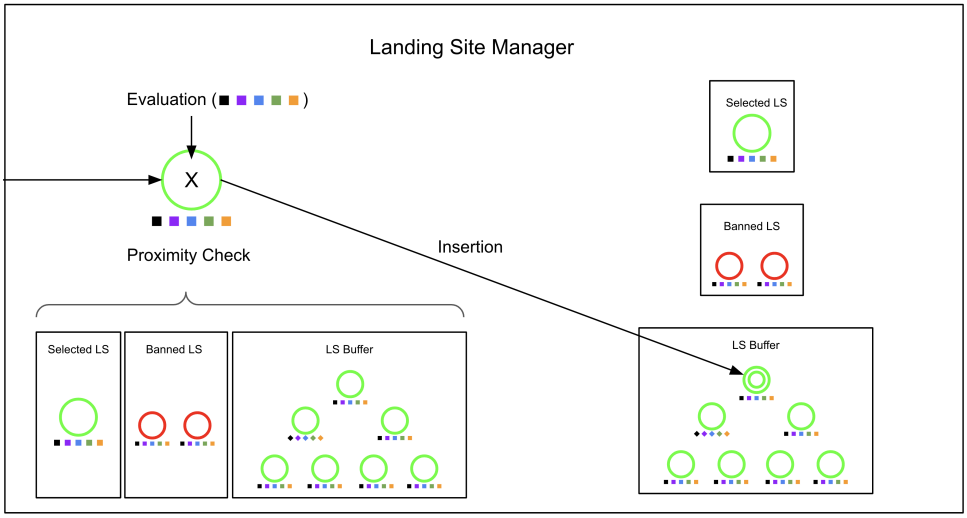


Figure 6.3: Complete LSM landing site handling - this example shows the case of a site in the buffer being re-detected.

6.2 Conceptual Behavior

To understand the landing behavior, the mission as a whole has to be considered.

6.2.1 General Mission

Looking at a mission performed by the autonomy framework at a high level and laying emphasis on the landing pipeline, the procedure looks like this:

- Preparation
  - Beforehand, a mission is created manually by exporting a QGC plan and converting it to a format readable by the autonomy.
  - The configuration parameters are set to the adequate values for the flight at hand.
  - The autonomy is started.
  - The necessary connectors with the ROS nodes, the flight controller, and further auxiliary nodes are initialized.
  - An actuator check is performed.
  - Finally, the final starting signal is awaited.
- Takeoff

  The drone takes off vertically until the desired course altitude of the first entered waypoint is reached. During this vertical flight, stereo starts off detecting point clouds, which are given to LSD for map aggregation and landing site segmentation. After reaching the switching threshold mentioned in section 5.2.5, stereo is deactivated, and SFM starts up. However, as the rotorcraft is still in mere vertical motion, no point clouds are created by SFM.
- Mission

  Upon reaching the first waypoint's altitude, the drone starts lateral motion. This is where SFM starts supplying LSD with the first non-trivial landing sites, which are transferred to and processed by the autonomy. The mission may have numerous waypoints, and the landing sequence is initiated upon reaching the last one.
- Landing

  The landing behavior uses the perceived landing sites to reactively and safely determine, validate, and approach a landing site. When the landing decisions have concluded, the drone lands or, in other words, descends until ground contact is established. The decision-making is detailed in section 6.2.2.
- Termination

  Upon successfully landing, the drone enters the termination state, which cleans up the autonomy pipeline and disarms it. If the drone's battery has sufficient residual voltage, a new mission can be started.

6.2.2 Landing Behavior

The core of the landing behavior is what happens between selecting a landing site and, finally, vertically landing. As previously mentioned, the landing behavior was designed to be safe. Additionally, an effort was made to derive an efficient behavior implementation within this safe behavior space.
The conceptual landing behavior, which is started at the end of the mission state and repeated a predetermined number of times, is the following:

1. A check is made to whether any landing sites have been registered yet
   (a) If no landing site was detected, go to a new location and fly a pattern until a potential site has been found.
   This procedure is repeated a predetermined amount of times. If they fail consistently, the drone returns home by ascending to a safe altitude, traversing to the home location, and descending until touchdown, slowing down upon reaching a certain proximity above ground.
   (b) If there are landing sites, order them in ascending order (regarding their loss scores) and pick the best one.
2. Using the drone's current position, the landing site's location, and the site's obstacle altitude parameter, determine a minimum safe clearing altitude up to which to ascend.
3. Traverse to the considered landing site.
4. Descend to a predetermined verification altitude above ground (during this descent, the stereo camera takes over at some point).
5. Hover for a given duration. In practice, 15 s seemed an adequate amount of time to give LSD a chance to converge.
6. Perform the verification check introduced in section 6.1.3.
   (a) If verification is successful, initiate landing.
   (b) If verification fails, ban the landing site and return to point 1.

6.3 Software Implementation

The final landing behavior is implemented as a behavior tree, which allows the adaptive and scalable implementation of simple tasks to create a larger and more complex decision-making entity.

6.3.1 Action Definition

The autonomy framework already defined the core control flow nodes and some action nodes required for the landing behavior. These are introduced in section 3.3.4. Hereafter displayed is a list of additional required actions for the landing behavior. It should be noted that they define individual, independent actions and should not be considered a description of the entire behavior. For this consider section 6.3.2

- CheckLandingSiteAction
  This is a simple utility action that checks whether any landing sites have been found by querying the LSM's landing site buffer length.
- ChangeAltitudeLSAction
  This action was implemented to allow us to move vertically to a certain fixed altitude above a chosen landing site. This allows the drone to easily reach a predefined safe altitude at which it traverses to the landing site.

- GetLandingSiteAction

  Upon leaving the mission state and entering the landing state, the autonomy attempts to select the currently best landing site according to the in section 6.1.2 introduced loss function.

  This is done by using the landing site manager to rank the landing sites in an ascending sorted list (as opposed to a max-heap) and selecting the first entry with the lowest loss.

  The selected landing site is then stored in a blackboard interface which allows easy data sharing between all the actions within the autonomy.

  Once a landing site has been chosen, the fail-safe mechanism to go to that landing site is the following:

  1. Ascend to a safe altitude.
  2. Traverse laterally to the landing site's position.
  3. Descend to the landing site or verification altitude.

  The question remains, however, what the adequate clearing altitude is. The goal is to find an altitude high enough to fly safely without the risk of collision yet low enough not to waste energy for the increased ascent/descent distances.

  This is where the aforementioned obstacle altitude from section 6.1.1 comes in. The obstacle altitude gives us an indication of the height to clear around the landing site. Therefore, we can take this as the start altitude for the derivation.

  As the DEM created by LSD covers only a limited area, the obstacle altitude is only an indication of the highest terrain present at the chosen landing site. The worst-case scenario would be detecting the landing site at the very edge of the DEM shortly before significantly higher terrain starts. Therefore, one has to anticipate this case.

  A safe altitude buffer is implemented by assuming the worst-case 45-degree terrain incline starting immediately at the landing site. Thus, the necessary clearing altitude can be derived by linearly interpolating the final value from the initial obstacle altitude and the distance between the drone and the landing site that must be covered.

  In the end, to not ascend to excessive heights, the clearing altitude is capped at a pre-determined, terrain-based fail-safe altitude

$$z_{\text{clear}} = z_{\text{obst.}} + z_{\text{buffer}} + d_{\text{drone-LS}} \tag{6.2}$$

$$z_{\text{clear}} = \min\left(z_{\text{clear}}, z_{\text{fail-safe}}\right) \tag{6.3}$$

  Above, we can see the derivation of the clearing altitude where z_{clear} defines the clearing altitude, z_{buffer} a safety buffer on top of the obstacle altitude, $d_{\text{drone-LS}}$ the drone's current distance to the site and $z_{\text{fail-safe}}$ the fail-safe altitude at which the clearing altitude is capped. A schematic visualization of this is depicted in fig. 6.4.

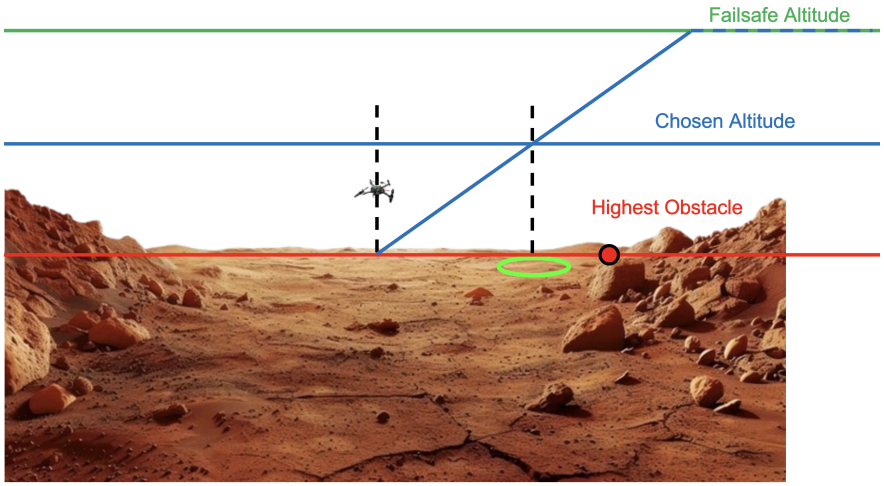


Figure 6.4: Schematic of the clearing altitude decision procedure

- LandingSiteVerificationAction

  Once a landing site is chosen, we must ensure it's a good spot to land before attempting to do so. The LandingSiteVerificationAction does this by using the verification altitude property of a given landing site. When a landing site is detected using SFM at 100 m altitude it will be overwritten when re-detected at 2.5 m above the ground.

  This is the mechanism exploited to verify a landing site. The drone hovers above the landing site for a pre-determined duration in place and attempts to re-detect the chosen landing site. This means that the LSM continuously processes incoming landing sites, and if one is close enough to an existing one, it is considered a re-detection. The landing site is verified in that case, and the landing action can be triggered.

  In case of verification failure, the landing site is removed and actively banned to prevent future false positives at high altitudes.

  Additionally, as previously mentioned, the verification hover period at low altitudes most probably leads to the detection of nearby landing sites. This is almost as important as verifying a previous landing site, as it yields a good candidate in close proximity.

- LandingSiteSearchAction

  The previously described actions are core implementations of the nominal behavior in the landing sequence. However, what happens when no landing sites are found, either due to a fault in the landing site detection setup or really unfavorable terrain?

  The most intuitive answer seems a good idea—simply look further for a site. The technical implementation of this task at high altitudes requires detection by SFM and, therefore, lateral motion. So, an easy solution is to fly a pattern at a new location with the exact same landing site handling procedure as in the nominal case.

  The LandingSiteSearchAction implements this through a predefined rectangle of waypoints which are flown through. This action module is canceled upon detecting a single landing site. In that case the usual landing procedure is continued.

- GetNextPatternCenterWPAction

  If the drone fails to find additional landing sites, it simply moves to a new location and tries again. This behavior is repeated several times or as long as sufficient battery voltage is available.

  The drone picks a random position around the final mission waypoint, flies there, and moves laterally in a rectangular shape in the hope of detecting landing sites.

  Note: If this action and the subsequent landing site search fail to detect any landing sites, they are repeated as often as the battery state permits, and the drone has to return to its takeoff position.

6.3.2 Behavior Tree Implementation

In the following, the behavior tree visualizations follow the scheme introduced in section 3.3.4.

High Level Behavior

To improve the readability, first, the high-level landing behavior is shown in fig. 6.5:

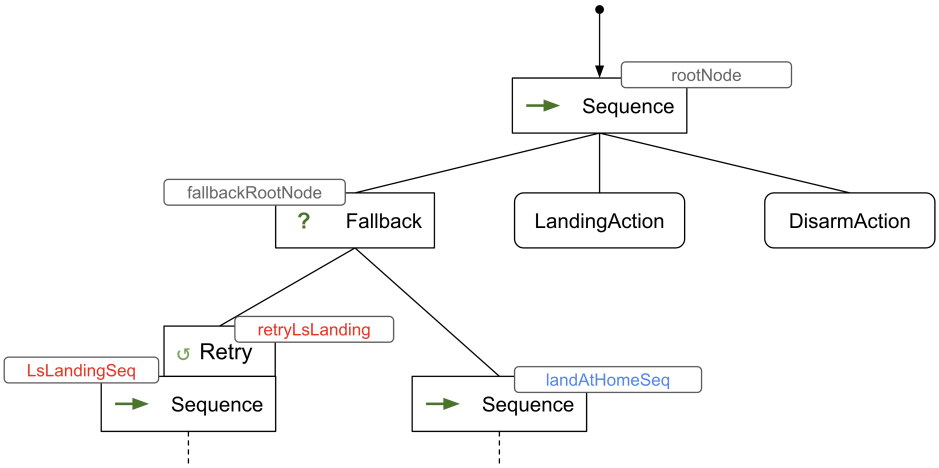


Figure 6.5: High-level landing behavior represented as BT

When the landing state is entered, the root node is initiated. First, the core landing behavior outlined in section 6.2.2 is attempted repeatedly using the retry decorator node. If it is successful, the fallbackRootNode returns success, and the landing Action and the disarmAction are executed. If the node fails several times, it returns the failure state, and the sequence node to land at the home position is initiated.

Core Landing Behavior utilizing Landing Sites

The conceptual implementation of section 6.2 is shown in fig. 6.6. As for the behavior tree example from the system overview in section 3.3.4, node names are attached when a node type is used more than once. Due to a lack of space and to improve readability, the action nodes in the VerBehSeq (verification behavior sequence) were placed onto two lines. Their order is to be read according to the line connections from the sequence node from left to right.

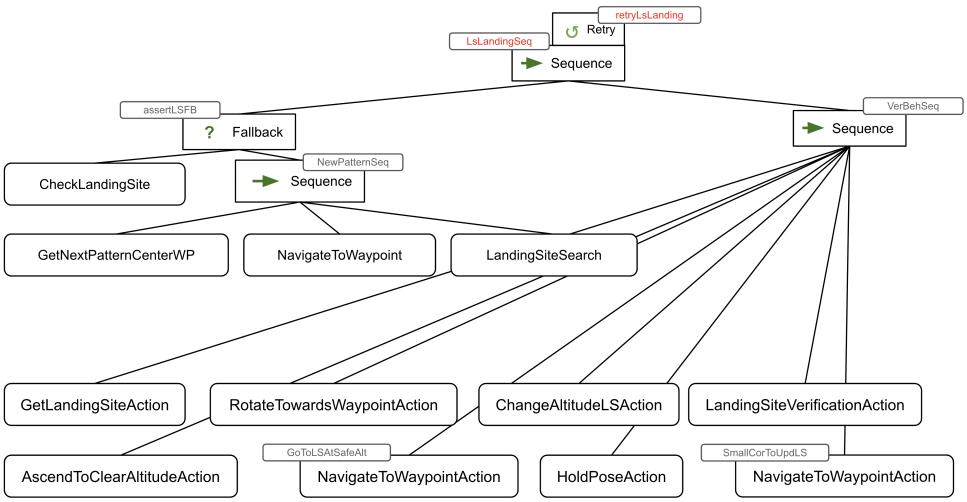


Figure 6.6: Landing procedure implementation as behavior tree

The left part, summarized by the assertLSFB (assert landing site fallback), checks whether landing sites were detected. If they were, the fallback yields success directly. Otherwise, a landing site detection pattern sequence is executed. This site is covered by numbers 1. a) and 1. b) of the conceptual behavior in section 6.2.
The right side contains the behavior to be executed when a landing site has been found. These are exactly the steps laid out in numbers 2 - 6. b).

Behavior to Land at Home Position

The last behavior to define, though simpler than the previous, is the process of returning to and landing at the home position.
The implementation thereof is shown in fig. 6.7.

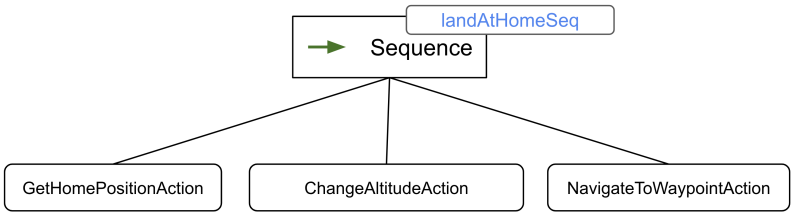


Figure 6.7: Behavior tree of the implementation when landing at the home position

The behavior when landing at the home position is very similar to the landing site-based landing; however, it is a bit simpler. The landing location is queried, following which the drone ascends to a safe altitude and traverses to the home position. There, the landing action is then initiated.

6.3.3 Full Pipeline in Action

In the following, a simulated flight demonstrating this landing behavior is shown. fig. 6.8 shows the setup of the demonstration flight on the Arroyo map introduced in section 3.1. The drone takes off until 100 m altitude and traverses to the indicated location where the landing behavior is initiated. Note that fig. 6.11 and the following images of the demo were taken when flying with ground truth depth above the stereo altitude. This is because SFM had often occurring issues at the time of this work. For more on this subject see section 7.1.

Figure 6.8: Demonstration flight setup: Left: Arroyo Seco Gazebo map, Right: QGroundControl mission to be flown at 100 m altitude

Takeoff

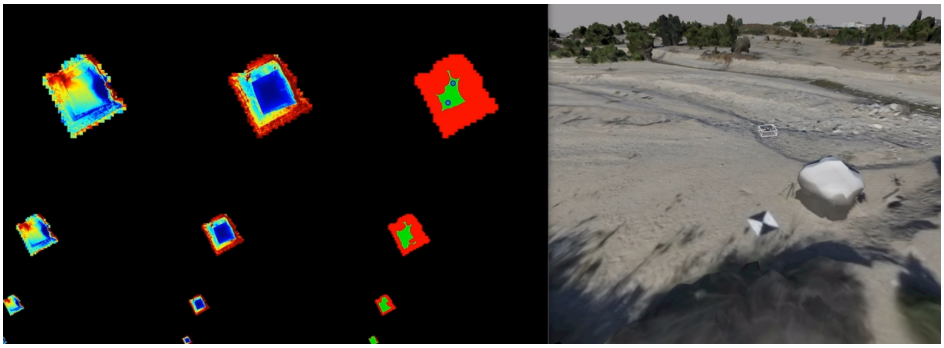


Figure 6.9: Takeoff of the drone: Left: LSD debug image, Right: Ascending drone in the simulation

fig. 6.9 shows the drone's takeoff. During this time, LSD is supplied with stereo camera depth point clouds. Note the increasing uncertainty of the outer edges due to the increasing altitude and the fixed baseline.
fig. 6.10 shows the same takeoff at a later point in time. Note how the LSD debug image stayed the same despite the drone rising to a higher altitude. This is because

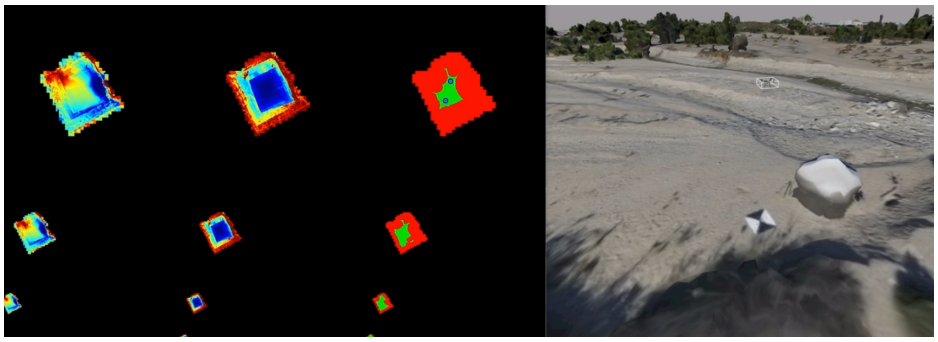


Figure 6.10: Later point of the drone's takeoff

the stereo camera depth node has been switched off by the laser range finder readings, and SFM has taken over. As SFM does not perceive depth during vertical ascent, however, LSD does not register any input.

Mission

Upon reaching the cruise altitude, the first mission waypoint is pursued. The lateral motion allows SFM to slowly merge its information into the LSD DEM. After a few seconds, the SFM information converges sufficiently for the landing site to be detected on them. This is shown in fig. 6.11.

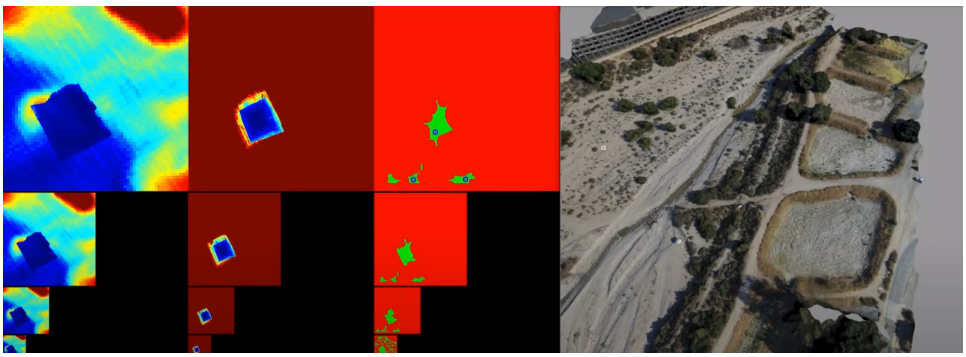


Figure 6.11: Initial traversal in the mission state upon reaching cruise altitude.

From this point onwards, the whole mission is flown with the SFM pipeline supplying LSD with depth information and LSD sending detected landing sites to the autonomy.
When reaching the last waypoint of the mission, the landing state is initiated.

Landing Behavior

Following the landing procedure visualized in fig. 6.6, a landing site is selected.
Following the outlined behavior, the drone ascends to a clear altitude and traverses to the landing site location, where it descends to a verification altitude above the ground. This is shown in fig. 6.14.
After verifying the landing site in question, the last small re-detection correction of the landing site's position is considered, and the drone adjusts accordingly as shown in fig. 6.15.

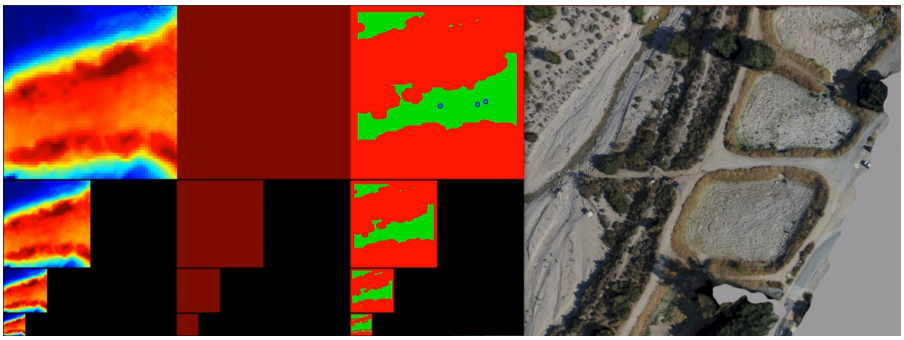


Figure 6.12: End of mission state

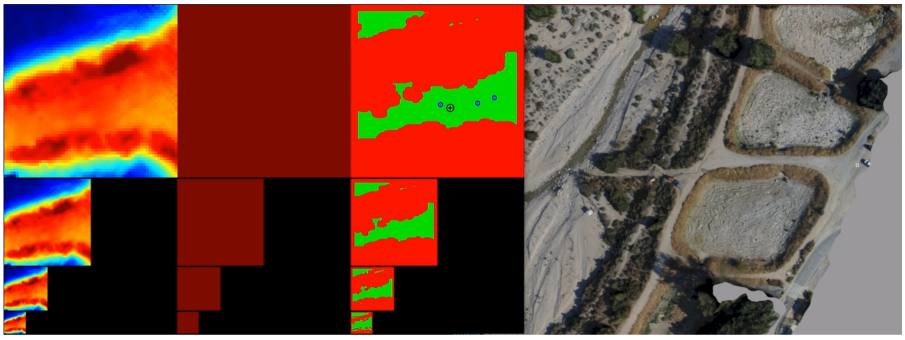


Figure 6.13: Start of landing sequence: the best landing site is selected and indicated in the LSD debug window with the black encircled cross.

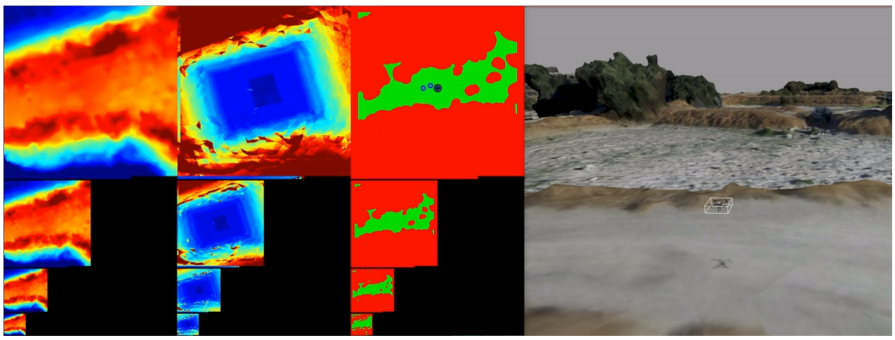


Figure 6.14: Rotorcraft hovering on top of landing site and trying to re-detect chosen landing site

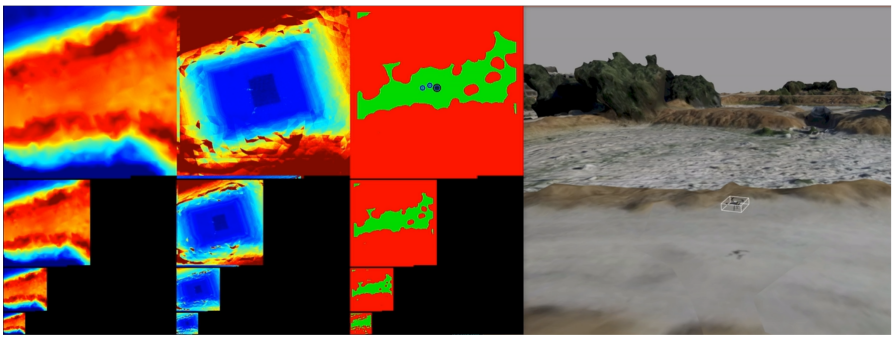


Figure 6.15: Rotorcraft adjusts position to refined landing site location

Final Landing Action

Finally, the drone can land safely. It descends with a constant velocity that decreases upon entering a minimum proximity to the landing altitude. fig. 6.16 shows the final location of the rotorcraft after the landing procedure.

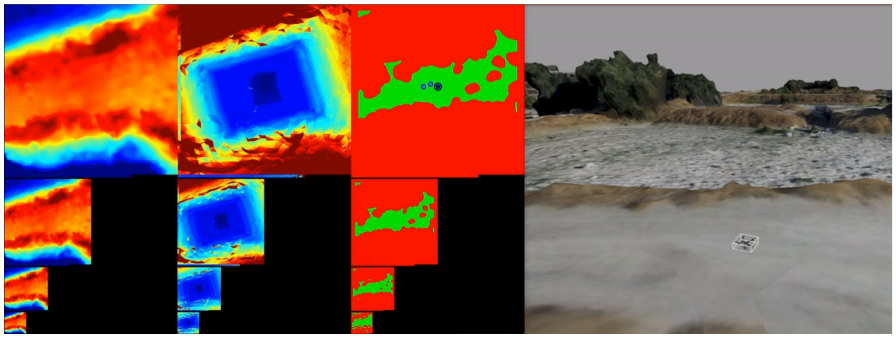


Figure 6.16: Final landing of the drone chosen by LSD

Chapter 7

Evaluation

Before analyzing the work done in this thesis, the depth acquisition pipeline, consisting of the SFM and LSD algorithms, is discussed.

7.1 SFM Insufficiencies

SFM is part of the LORNA pipeline and promises to be a valid option for the tackled endeavor. Before this work however, it was never tested extensively, especially at high altitudes (100 m).

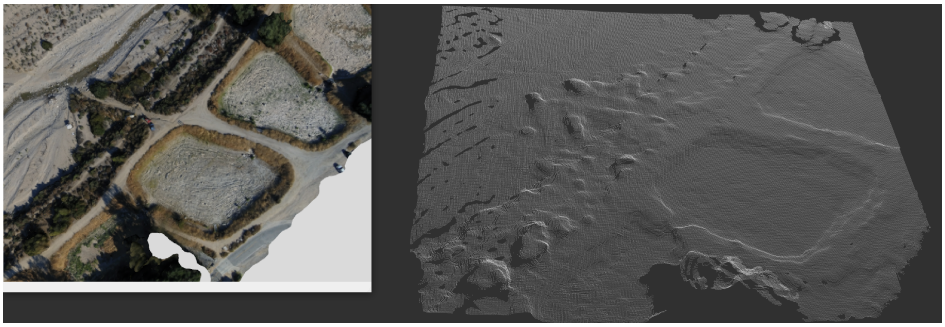


Figure 7.1: SFM point cloud when initialized correctly - left: SFM's input image, right: SFM's point cloud output visualized in rviz

Looking at fig. 7.1, it is fair to say that SFM is capable of creating accurate point clouds at high altitudes. Though performing well when initialized successfully, SFM often had issues during startup and showed frequent insufficiencies during rotations or, in general, high-altitude flights.
Unfortunately, a lack of time prevented me from investigating these issues in detail. Nevertheless, the shortcomings of the current standing are laid out to the best of my knowledge in the following to hopefully be resolved in future work.
SFM's main issues occurred in the following three scenarios:

- During startup

  Frequently, when starting SFM's keyframe recording, it would not start or output corrupted data, especially when there was slight rotation in the drone's movement. When making sure that SFM is initialized during perfect lateral motion, the initialization succeeded most often. This leads to a strong suspicion that SFM handles rotations poorly during initialization.

- When changing direction

  Most often, the drone is in vertical or straight, lateral movement. However, it has to turn between waypoints or landing site attempts. Currently, SFM isn't able to handle these cases well. As for the initialization, it either stops or outputs faulty data. A demonstration of SFM outputting point clouds and stopping upon changing direction is shown in fig. 7.2. An example of corrupted input data is shown in fig. 7.5 and fig. 7.3. Note how in fig. 7.3 SFM started producing invalid data after a map move. Furthermore, fig. 7.6 and fig. 7.7 depict two instances of faulty point clouds created by SFM.

- During high-altitude flights

  As mentioned in section 3.2.1, SFM often creates point clouds using the same keyframe until it can't establish sufficient image overlap with incoming images anymore and the oldest keyframe is switched. At high altitudes, this leads to frequent, large map movements in LSD as outlined in section 3.2.2. As a result of this, LSD's elevation map has a harder time converging, and fewer landing sites are detected. This phenomenon is shown in fig. 7.4, section 7.2 and chapter 11.

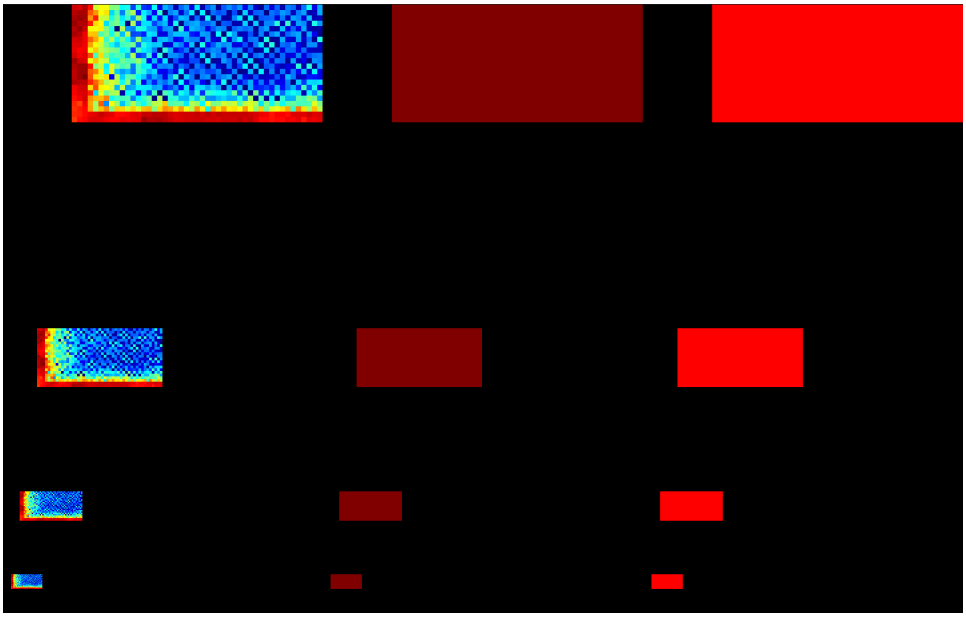


Figure 7.2: LSD debug image after changing direction: SFM stopped to generate stereo points so that the LSD map is moved without new information coming in.

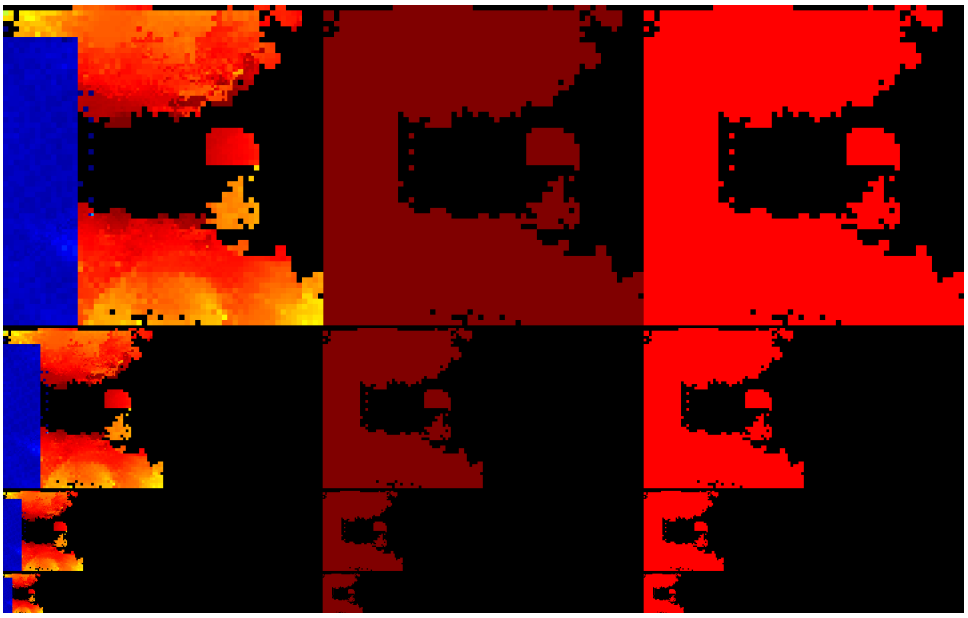


Figure 7.3: One example of the LSD output when supplied with suboptimally performing SFM

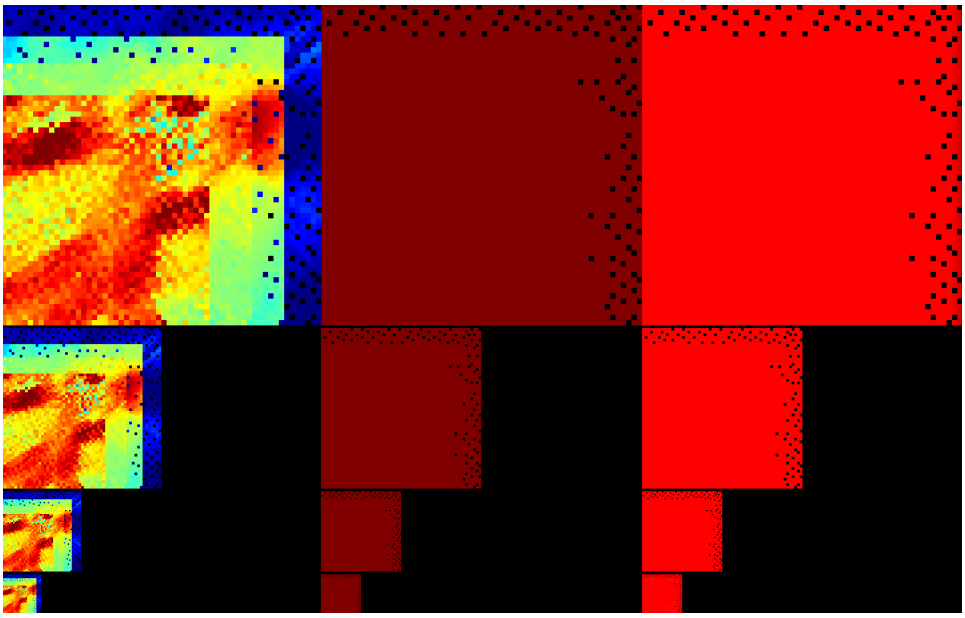


Figure 7.4: Intermediate LSD state after map movement

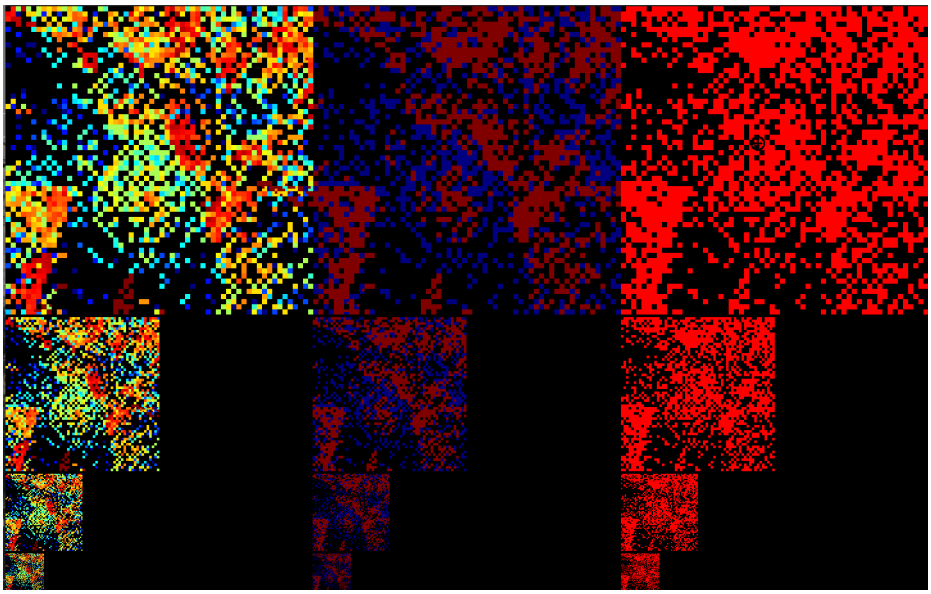


Figure 7.5: Faulty SFM data points generated during flight

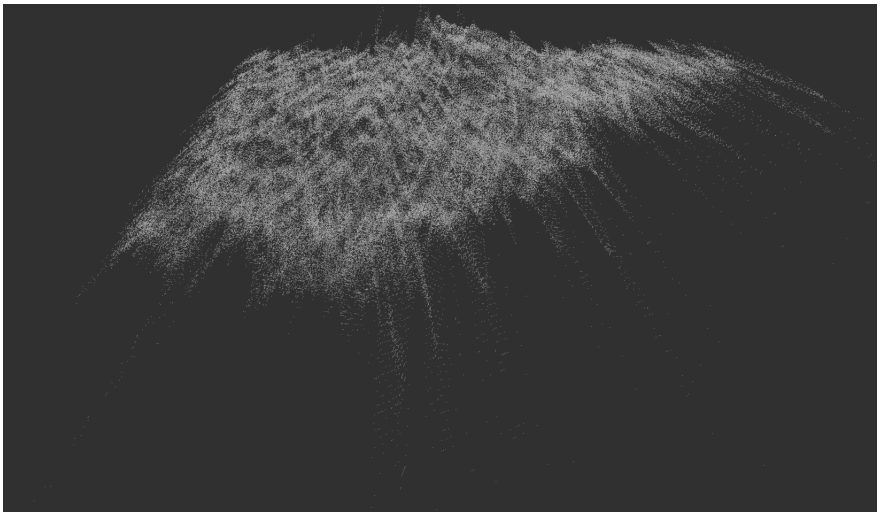


Figure 7.6: Visualization of faulty SFM point cloud

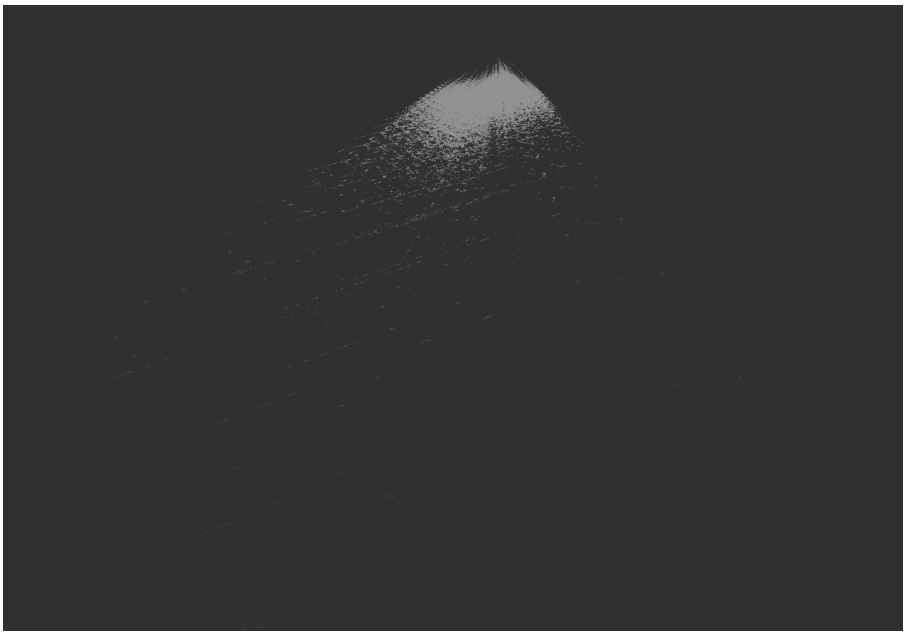


Figure 7.7: Additional visualization of faulty SFM point cloud

7.2 LSD Analysis

7.2.1 Map Coverage

Given high-quality depth clouds, the LSD algorithm can accurately detect valid landing sites without fail.
LSD's consideration of the same terrain at different resolutions leads to a very consistent elevation map. However, when flying at high altitudes where the image footprint is big, this same characteristic prevents the usage of most of the supplied camera image as only the centermost area coinciding with the other layers of the image is used. This is shown in fig. 7.8.

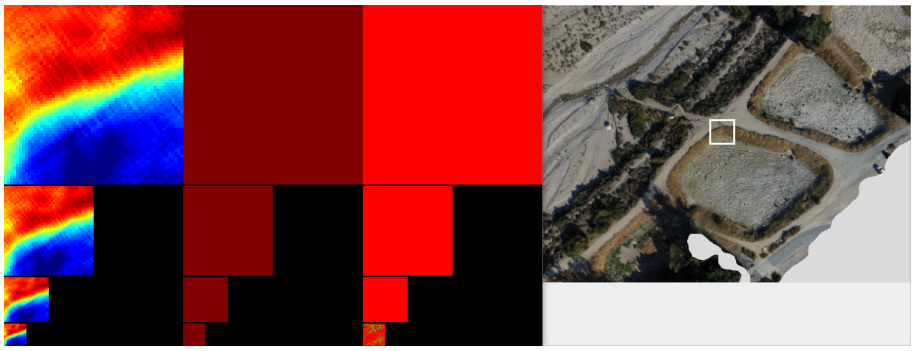


Figure 7.8: Comparison of the tracker camera image and the region of interest used by LSD

7.2.2 Number of Layers

Due to a fixed number of layers, the image points detected at high altitudes have a pixel footprint that exceeds that of a cell from the coarsest layer. Therefore, the point cloud input to LSD is sparse and leaves many cells empty. Empty cells prevent the detection of landing sites in the neighborhood. The smaller the coarsest layer's cell footprints, the stronger the effect. fig. 7.9 shows the LSD DEM before convergence when using 2 layers at 100 m altitude.
The lowest resolution layer has a fixed resolution of 0.05 m / cell. Considering the 4-fold division of a cell when moving to a higher resolution layer, the resolution at the top level is 0.2 m / cell. On the other hand, at 100 m altitude with the camera parameters introduced in section 4.1, the tracker camera's pixel footprint is 0.363 m/px.
After a while, the map is filled with further measurements as shown in fig. 7.10.

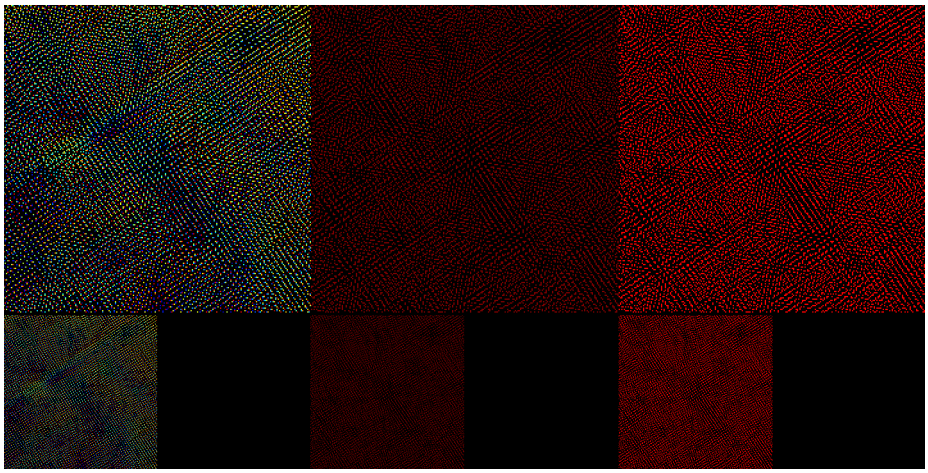


Figure 7.9: LSD DEM at 100 m right after initialization with only 2 layers

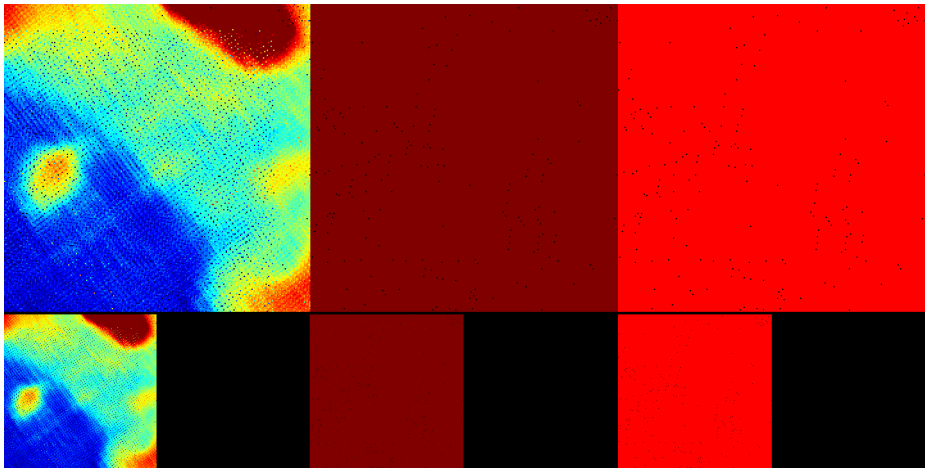


Figure 7.10: DEM at 100 m with 2 layers after some conversion time

However, due to the map movement mentioned in section 7.1, most of the DEM is erased before the DEM converges sufficiently.This is shown in fig. 7.11.

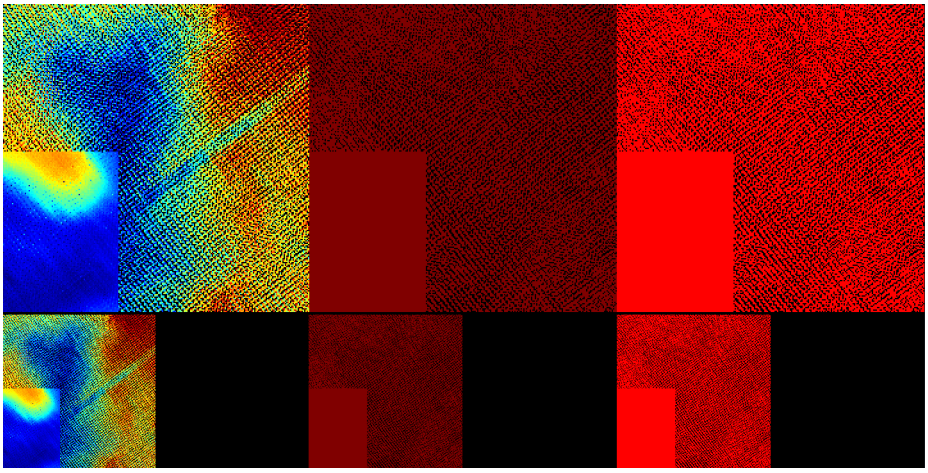


Figure 7.11: DEM at 100 m with 2 layers after map movement

A remedy for this is the usage of more layers, resulting in a decreased resolution at the coarsest layer. This way, the discrepancy between the footprints of the incoming points and the cells available is not as big, and as a result, the DEM is less sparse. It has to be noted, however, that using more layers leads to more computational overhead. An example LSD image shortly after initialization with 4 layers is shown in fig. 7.12.

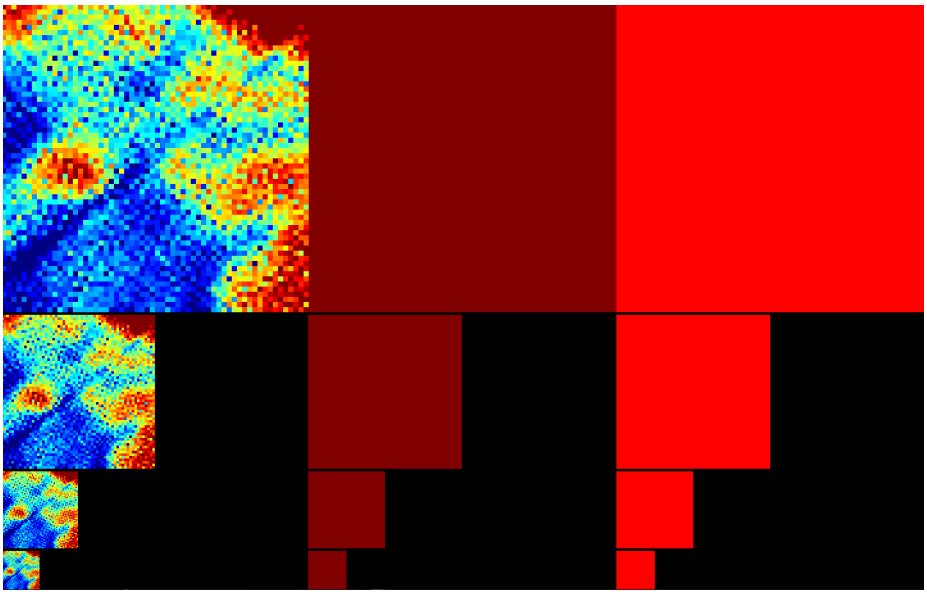


Figure 7.12: LSD DEM at 100 m with 4 layers

Note that more layers lead to a stronger pooling effect at the highest resolution layers. In other words, the sparsity of the DEM when using LSD with fewer layers is counteracted by the enhanced pooling of further layers, leading to a high-resolution layer filled with the same coarse information present at lower layers. Due to the incoming point cloud's large point footprint, there is no higher-resolution informa-

tion to be had. Therefore, resolving the sparse maps with coarse information does not denote a loss of information. At high altitudes, we want to find areas with promising landing sites. These don't have to be perfect landing sites themselves yet and serve more as a signifier for later refinement. Thus, using lower-resolution information is perfectly nominal.
For a visual side by side comparison when using 2, 3, or 4 layers, see chapter 11.

7.3 Experimental Setup

The pipeline was tested by repeatedly flying a mission with randomized initial conditions. On each setup, 100 flights were performed, and each flight was given a maximum of 10 minutes before the next iteration commenced.

7.3.1 Simulated Terrain

The performed experiments were flown on the following two maps:

- Arroyo Map - Map from the Arroyo Seco area outside the East entrance of the Jet Propulsion Laboratory. Predominantly used map during development

Figure 7.13: Map of the Arroyo Seco area outside the Jet Propulsion Laboratory

Figure 7.14: Close Up of the Arroyo Map

- Rough Test Map - A controlled environment designed to prevent LSD from detecting any landing site unless a landing platform is specifically spawned. It was created using Blender and applying white noise perturbations to the elevation of a plane.

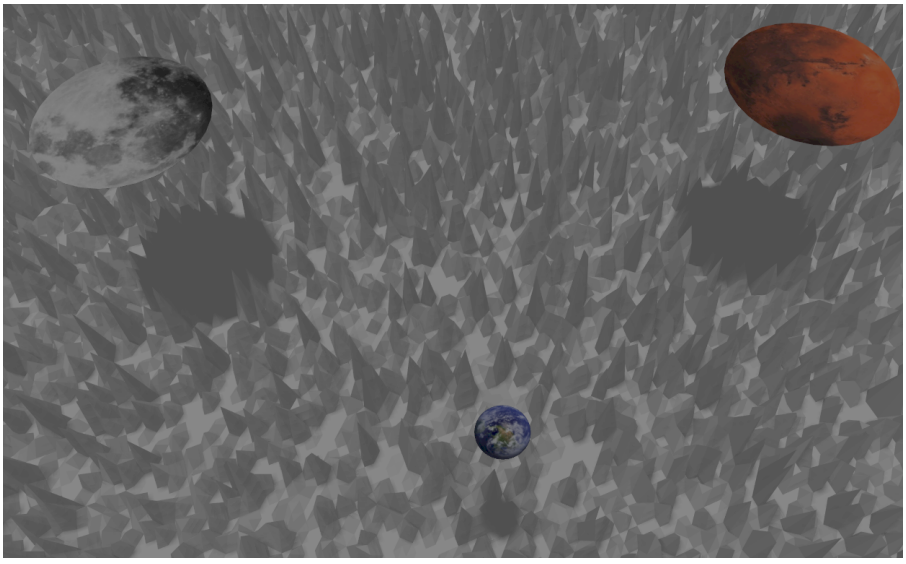


Figure 7.15: Synthetically created, hazardous map with no landing sites apart from inserted landing platforms.

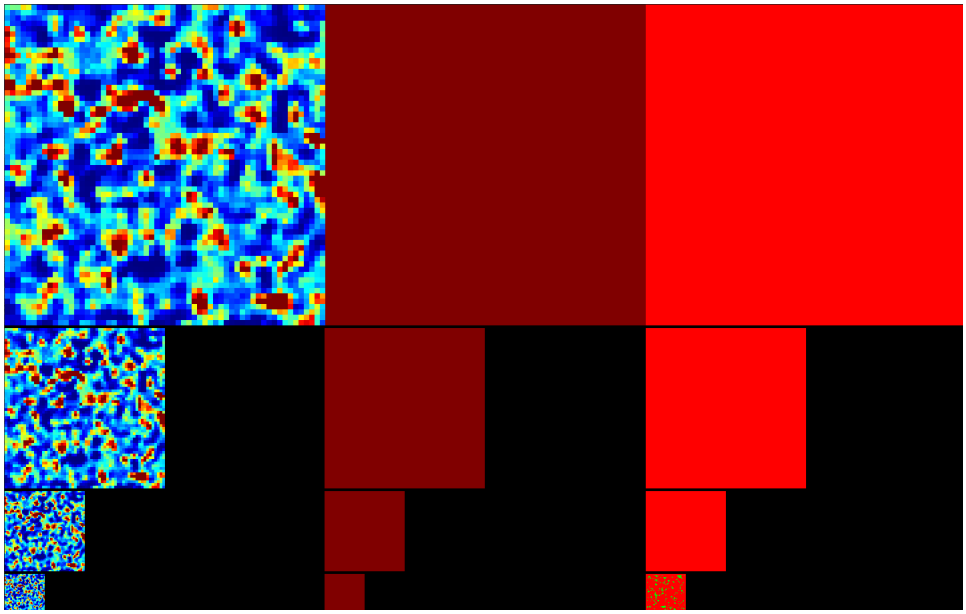


Figure 7.16: LSD debug output shown of the plain rough environment

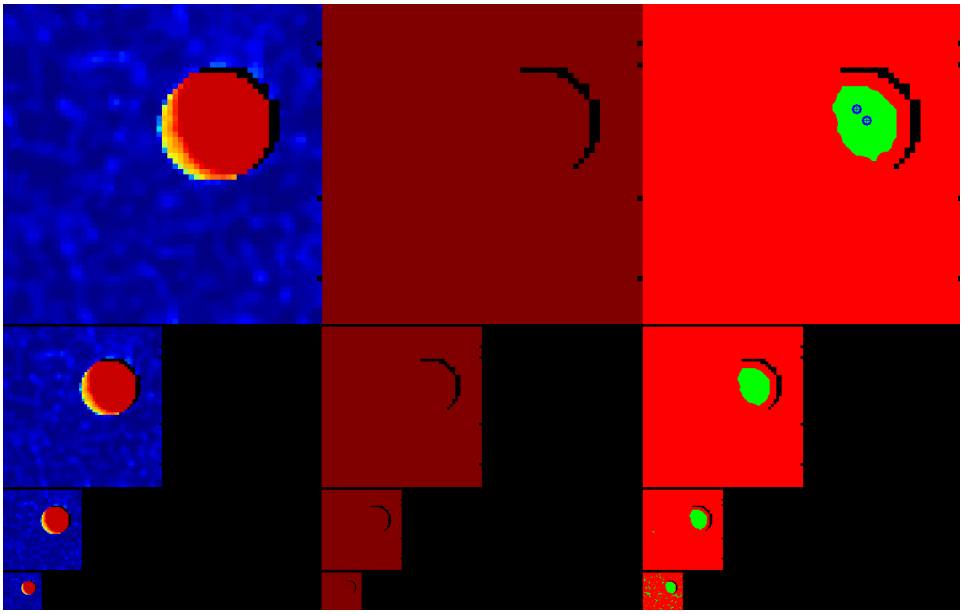


Figure 7.17: LSD debug image of the rough control map with spawned landing platforms

7.3.2 Drone Spawn

The drone was either spawned repeatedly from a default location on the ground (The start location from when the actual field tests were performed) or from a random location. For a simple way of avoiding terrain collisions, the drone was spawned on a randomly positioned disk at 40 m altitude. The start disk's size was only 0.5 m in diameter which prevented it from being considered too good of a landing site by LSD. This is important because the platform implicitly gains quality since it is located higher up than the terrain, leading to a lower distance to the drone when flying at mission altitude.

Figure 7.18: Arroyo Map with Fixed Start Position

Figure 7.19: Arroyo map with randomly positioned spawn platform

7.3.3 Depth Source

As mentioned in section 7.1, at the time of this work, SFM was very fragile as an individual module. Because of this and to evaluate the landing pipeline instead of the depth generation module, ground truth depth was used above the verification altitude. For the sake of completeness, however, an SFM run is also shown in the end. Regardless of whether ground truth or SFM was used, the verification at low altitudes was always performed using the stereo camera depth node.
Note that at the rough map introduced in section 7.3.1, no texture is projected onto the terrain. Therefore, on this map, ground truth was used exclusively.

7.3.4 Success Conditions

Two main metrics were considered to determine whether a flight was successful. First, it was checked whether the landing action in autonomy was initiated (this happens only after the landing site selection and verification are both successful). Secondly, a rosbag was recorded for each flight. To infer the safety of the rotorcraft, the bag was analyzed to check whether either the roll or pitch value exceeded the crash threshold. In practice, a threshold of 1.2 radians or shortly below 70 degrees proved to be an accurate decision boundary to detect a drone crash.

Home Landings

Landing at the home position defines the behavior when the drone does not navigate towards a landing site but instead uses the stored takeoff coordinates to navigate to the spawn location to land. This happens when no landing sites exist to be chosen. This occurs in two scenarios:

- Few landing sites were detected yet failed verification and were therefore banned. Thereafter, no landing sites remained.
- No landing sites were detected in the first place.

  This can happen because of two reasons:

  - The overflown terrain simply does not have a single landing site of decent quality. (see the rough map introduced in section 7.3)
  - The landing site detection algorithm failed to detect landing sites.

In both scenarios, going home is the desired behavior. However, if LSD does not detect landing sites where good candidates should be found, the run cannot be counted as a success, regardless of whether the drone landed safely after taking off. For instance, looking at the Arroyo map shown in fig. 7.14 and running numerous simulated flights on the terrain, it is clear that sufficient landing sites should be found on this map. Therefore, landing at home when flying on the Arroyo map indicates a landing site pipeline issue.

Off Board Mode connection Issues

Occasionally, connection failures between the PX4 flight controller and the autonomy occurred, leading to the deactivation of the off-board mode and, therefore, the loss of control of the autonomy over the rotorcraft. These issues arose most likely because the MAVROS connection in between failed to send a necessary repeated heartbeat, and thus, the connection was intercepted.

Self-evidently, this did not result in a guaranteed successful landing and was not counted as such. However, as these connection issues did not occur due to insufficiencies in the pipeline presented in this work, they were not considered failures of the pipeline either.
Therefore, to conclude the listing of the possible success states:

- Successful: The drone initiated the landing and did not fall over throughout the whole flight.
- Connection loss: The autonomy lost the connection to the flight controller, and therefore, no information can be gathered about this flight and the landing behavior's performance.
- Crash: Either the roll or pitch angle of the drone exceeded the predetermined threshold of 70 degrees. This is a definite failure of the pipeline.
- Home landing: The drone did not settle at a perceived landing site and returned to the home position instead. On a map with sufficient landing opportunities, this indicates a failure. When actually not traversing terrain with valid landing sites, as was often the case for the rough terrain shown in section 7.4.4, landing home is the desired behavior.

7.3.5 Visual Analysis

To further analyze the randomized flights in a bit more detail, visual landing attempt projections are used. Such an analysis image is shown in fig. 7.20.
The green points indicate successful verifications, which lead to the initiation of the landing action shortly afterward. (If no rotation above a failure threshold was detected, this is considered a successful landing.)
The yellow points are landing attempts, where a chosen landing site was not verified and thus banned from further consideration. It is crucial to note that not being able to verify a landing site is no issue at all. Landing sites detected at high altitudes merely provide a preliminary indication of potential landing zones. The subsequent phase, responsible for verifying these sites, yields the refined final landing site knowledge used for landing. So selecting a landing site at a high altitude, not being able to verify it, and subsequently choosing a nearby landing site might even be called the most sustainable chain of actions in the pursuit of autonomous landing.
Blue points indicate successful landings achieved through the trigger of the landing-at-home action. As previously mentioned, When flying on a map with obvious landing opportunities, triggering the landing-at-home action indicates an issue with the landing site detection mechanism.
Lastly, red dots indicate landing sites that were detected during a flight that was disrupted by a connection issue.
If multiple landing attempts are made, connection lines are drawn between the failed attempts and the final landing, indicating which subsequent sites were chosen.
It has to be noted that these plots were generated using non-orthographic background images, and the actual landings took place at locations slightly closer to the image center.
As the created points are rather large, fig. 7.21 shows the empty background image for reference.

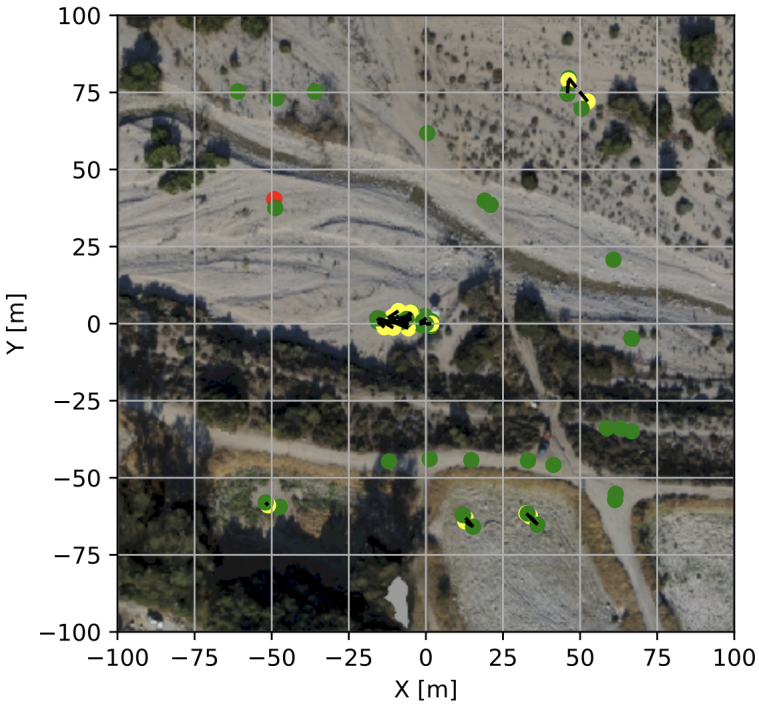


Figure 7.20: Landing Attempt dummy image - Green: successful landing, Yellow: verification failure, Blue: landing at home position

Figure 7.21: Non-orthographic Arroyo background image for visual evaluation of the test flights

7.4 Test Flights

The following tests were performed. Each test consisted of 100 flights simulated in Gazebo.

7.4.1 Arroyo - Randomized Waypoints

When starting 100 times from the fixed position indicated in fig. 7.18 and flying to random mission waypoints at 100 m altitude in a 70x70m vicinity on the Arroyo map, the following results were achieved:

Table 7.1: Results with fixed takeoff and random waypoint: Con. Loss stands for connection loss of the off-board mode.

# Flights	# Successes	# Con. Loss	# Crashes	# Home Landing
100	93	7	0	0

The landing attempt numerics are shown here:

Table 7.2: Landing attempts with fixed takeoff and random waypoint

# Flights	# Landing on first attempt	# 2 attempts	# 3 attempts
93	58	33	2

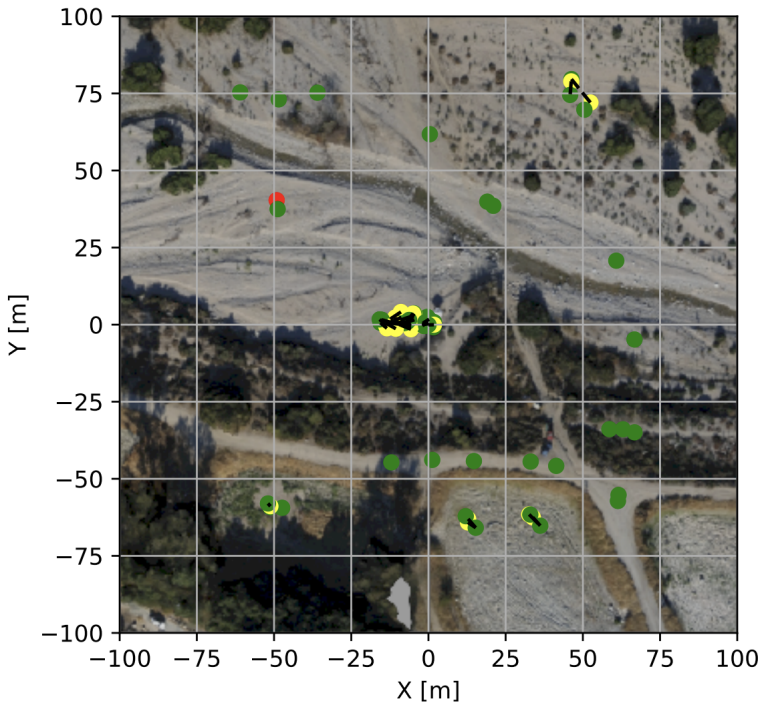


Figure 7.22: Landing Attempts - Green: successful landing, Yellow: verification failure, Blue: landing at home position

Numeric Discussion

table 7.1 shows the numeric outcome of the flights. 7 timeouts occurred, but each performed flight by the autonomy was a success, landing safely and controlled. No home landings were necessary, which, as described above, indicates a very robust LSD performance. A high number of flights where two landing site verification attempts were performed is also well within acceptable boundaries because the first landing site might as well be used as an indicator to draw the drone closer to an area where high-quality sites can be detected using the stereo camera. Twice it was the case that 3 landing attempts were needed.

Visual Discussion

Let's consider the landing image 7.22. The final landing sites indicated in green span exclusively decent landing areas. The verification attempts that fell short are indicated with yellow and annotated by a line connection to the successful attempt thereafter. Subsequent landing sites were always detected short distances away from failed candidates which is exactly the correct behavior in order not to lose time pursuing a far away follow-up landing site. As indicated in section 6.1.2, this is thanks to the verification altitude, which incentivizes the selection of other landing sites detected at the current verification altitude. Lastly, landing attempts performed before losing the connection to the flight controller are indicated in red. Note, an example log file of a flight which lost connection to the flight controller is shown in chapter 10.
The last thing to mention is the clustering of the landing sites around the takeoff location. The reason for this is twofold:

- Quality:

  The takeoff position in the simulation was chosen to be at the same location where the field tests were started in the real world. It was chosen exactly because of its even and smooth characteristics. Therefore, it is no surprise that many landing sites were detected around that location.

- OMG Conversion:

  During ascent, when using either stereo or ground truth depth, the same area is perceived repeatedly. This leads to a convergence of OMG certainty in the map aggregation step of LSD. As landing sites have a higher chance of being detected on terrain with low uncertainty, this converged takeoff area is most likely selected repeatedly and sent to the autonomy until it leaves the rolling buffer map.

  Note: the autonomy framework accounts for repeated detection of the same landing site. LSD however does not. Therefore, in such a case where the same terrain has been viewed for a long period, it is possible for the exact same landing site proposals to be sent to the autonomy repeatedly.

This test is a very clear demonstration of the applied chosen heuristics. The best landing site detected is very likely one that is detected at the takeoff position. Around this clustering of landings, there is a notable space where no attempts have been performed. This can be attributed to the competing motives of the landing site selection. The quality of the landing site defines the autonomy's choice until the drone's distance to that landing site is too big, and a closer one is chosen.

7.4.2 Arroyo - Randomized Takeoff and Waypoints

In this set of test flights, randomized takeoff positions were used as shown in fig. 7.19. Missions were built using a randomized waypoint in a 70x70m surrounding.
The numerical results look as follows:

Table 7.3: Results with random takeoff location and random waypoint

# Flights	# Successes	# Con. Losses	# Crashes	# Home Landing
100	94	6	0	0

Table 7.4: Landing attempts with random takeoff and random waypoint

# LS Landings	# 1 attempt	# 2 attempts	# 3 attempts	# 4 attempts
94	88	6	0	0

And the visual outcome is shown in fig. 7.23.

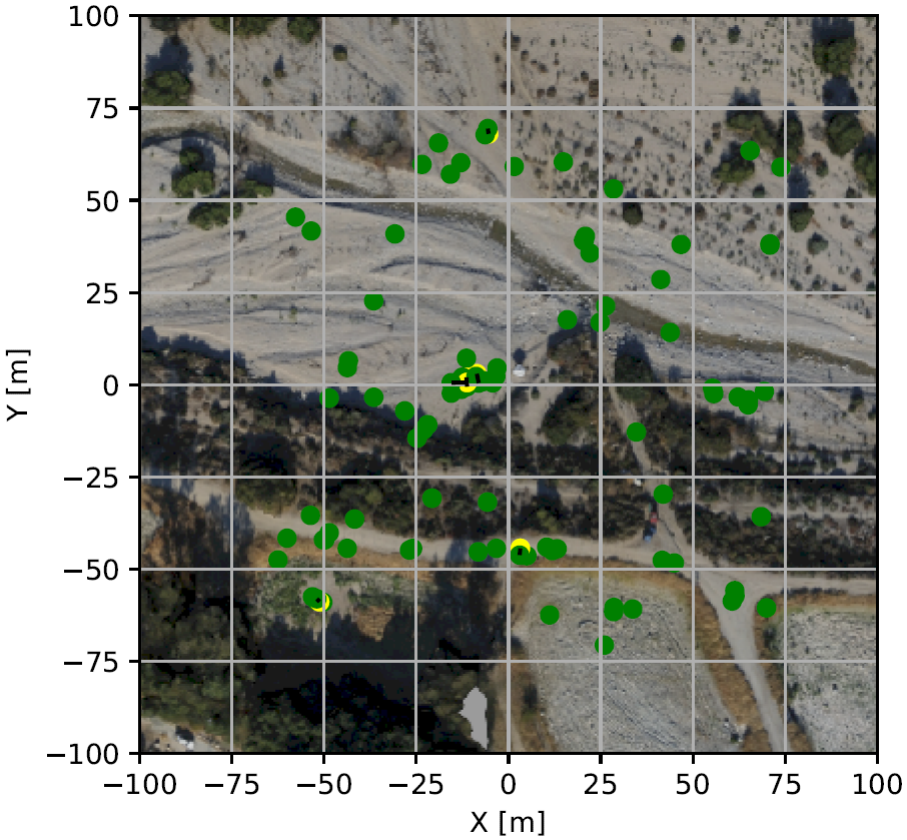


Figure 7.23: Landing attempts with randomized drone spawn - Green: successful landing, Yellow: verification failure, Blue: landing at home position

Numeric Discussion

6 timeouts occurred in this 100-flight experiment. Of the successful 94 runs, 88 landed at the very first attempt, and only 6 needed one more landing site. This is a pleasant outcome, which is, of course, correlated with the ground truth's high accuracy. Nevertheless, it has to be noted that multiple landing attempts at different sites are no indication of inferior performance.
The relatively high number of timeouts is undesirable, but as previously mentioned, it is not a fault of the presented pipeline and is therefore merely unfortunate but not worrying.

Visual Discussion

Compared to fig. 7.22, the landing attempts are more spread out over the map, and fewer landings were attempted at the takeoff position. This is the case because the random mission trajectories didn't always cover this area, and since the drone spawned at randomized locations, it did not spend the time during takeoff to repeatedly scan this plateau. Thus, the elevation map did not converge at that location leading to a smaller chance of landing sites being detected there.
As for the fixed spawn location, the second attempts were always at landing sites close to the false previous candidates. This is, of course, desirable to save time.
Lastly, the selected landing sites were all at plausible locations. To verify this, consider the empty reference image shown in fig. 7.21

7.4.3 Rough Map - Mission Covering Platforms

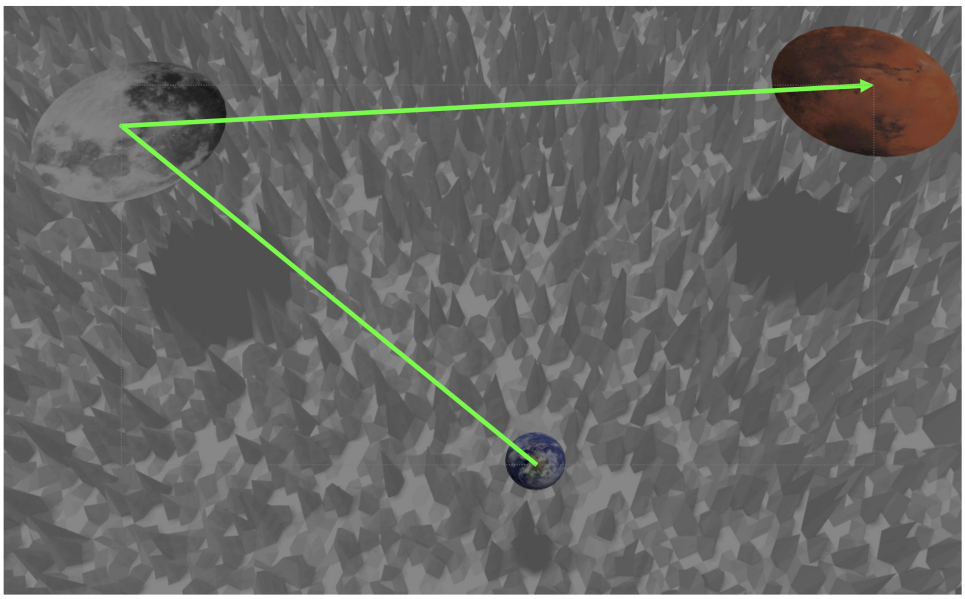


Figure 7.24: Test flights on rough map with waypoints covering the landing sites

In this test, the drone flew a randomized two-waypoint pattern which always covered the synthetically added landing platforms. The idea behind this experiment was to eliminate other landing possibilities and test whether the approach at hand would be able to detect the controlled good landing sites.
The numerical result thereof is displayed below:

Table 7.5: Results - Rough map with platform coverage

# Flights	# Successes	# Con. Losses	# Crashes	# Home Landing
100	99	1	0	0

Table 7.6: Landing attempts rough map with platform covering mission

# LS landing	# Landing on first attempt	# 2 attempts	# 3 attempts
99	98	1	0

As both the spawn platform and the landing platforms were spawned randomly, evaluating the visual outcome is redundant. For the sake of completion, however, it is still displayed in fig. 7.25.

Numerical Discussion

One timeout occurred, and one verification failed. The residual flights all succeeded on the first landing site detected.
All landing site candidates except one were verified. This convincing result is not surprising as the inserted landing sites are of very high quality, and are therefore unlikely to trigger a verification failure. The verification rate is not perfect, however, as the landing sites were only thin disks, as shown in fig. 7.17.
To determine the pipeline's ability to choose the best available landing area, the final landing locations were compared to the platforms spawned. This was the result:

Table 7.7: Final Landing Platform Choice

# Landings	# On Earth (spawn)	# Moon (1st pf)	# Mars (2nd pf)
99	4	2	93

Table 7.8: Landing locations chosen by this experiment - The Earth platform denotes the small spawn location of the drone. The Moon corresponds to the platform covered by the first mission waypoint, and the Mars platform is where the missions ended.

This result shows, as desired, that the last platform, which is the one closest to the drone at the time of the last waypoint, was chosen most of the time.
When the Earth platform, which is the takeoff location, was close enough to the last waypoint, it was sometimes chosen because the landing sites discovered on it during takeoff were detected closer above ground than the ones at course altitude. The same happened when the first platform could be seen during ascent and was relatively close to the last waypoint.

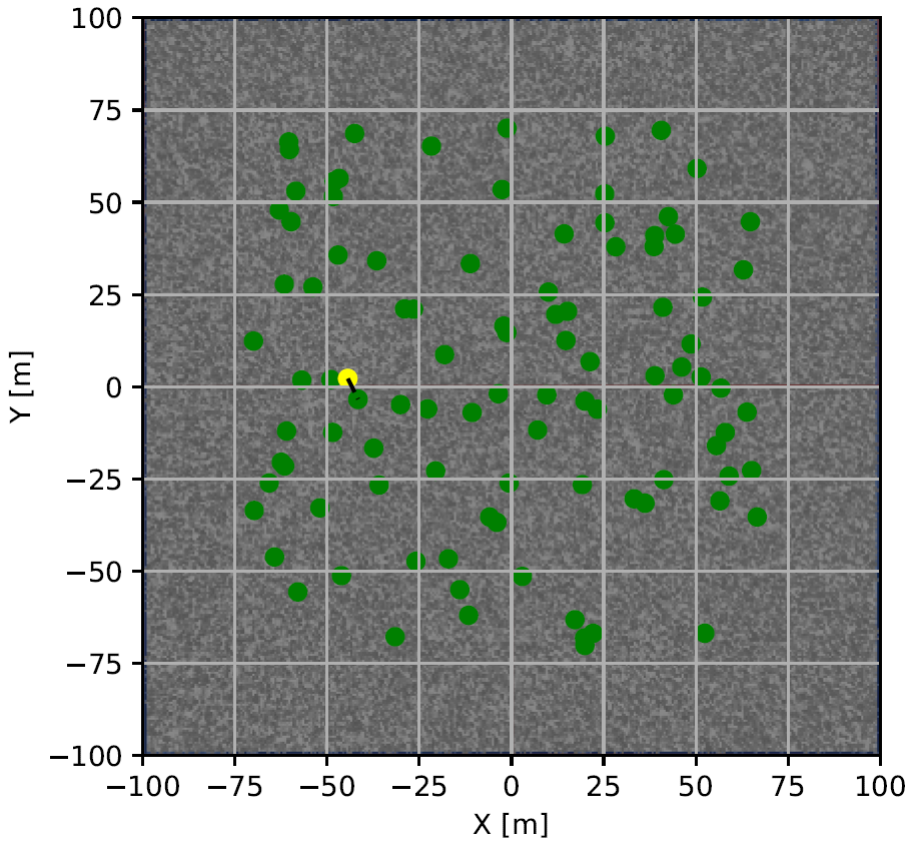


Figure 7.25: Landing attempt locations for the rough map with platform covered by the mission

7.4.4 Rough Map - Random Waypoints

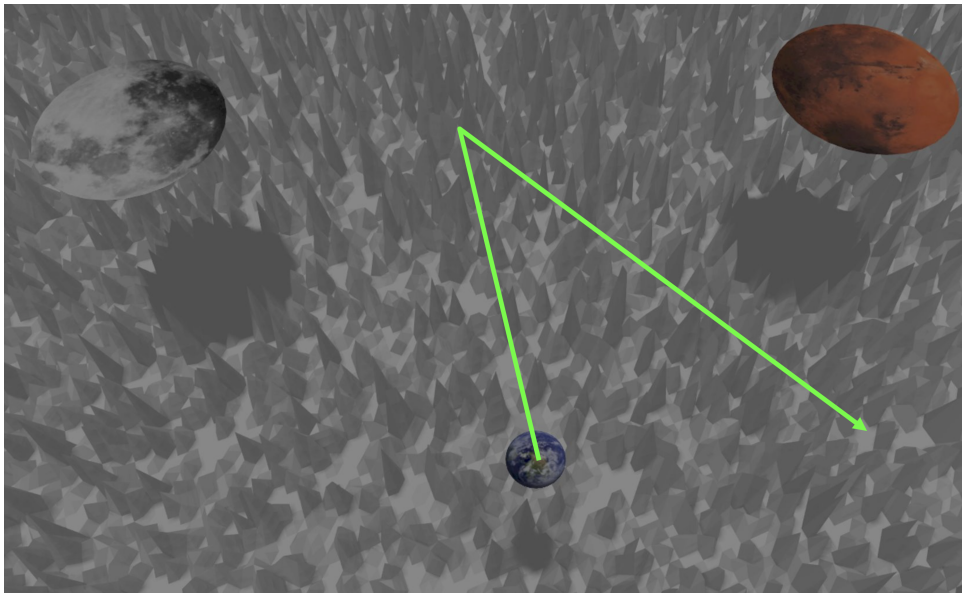


Figure 7.26: Test flights on rough map with waypoints covering the landing sites

In this test, changing the setup a bit, the drone flew a completely randomized two-waypoint pattern without the guarantee of flying over a safe landing site. This test set analyzed the pipeline's capabilities to deal with the case when no landing sites are detected initially.
First, in this experiment of 100 flights, when the drone did not detect any landing sites, it flew only minimal landing site search patterns within a 1 m radius. This should serve as a baseline to see how many sites are approximately found without significant additional search actions.
The numerical result thereof is displayed below:

Table 7.9: Results - Rough map with platform coverage

# Flights	# Successes	# Con. Losses	# Crashes	# Home Landing
100	98	2	0	54

Table 7.10: Landing attempts rough map without platform covering mission

# LS landings	# Landing on first attempt	# 2 attempts	# 3 attempts
44	43	1	0

Numerical Discussion

Looking at table 7.9, what stands out are the 54 times the drone went to the home position. This is the desired behavior when no landing site is found, and it is therefore to be considered a success. It is also rather expected as the minimal search pattern radius likely prevents the drone from finding additional landing sites.

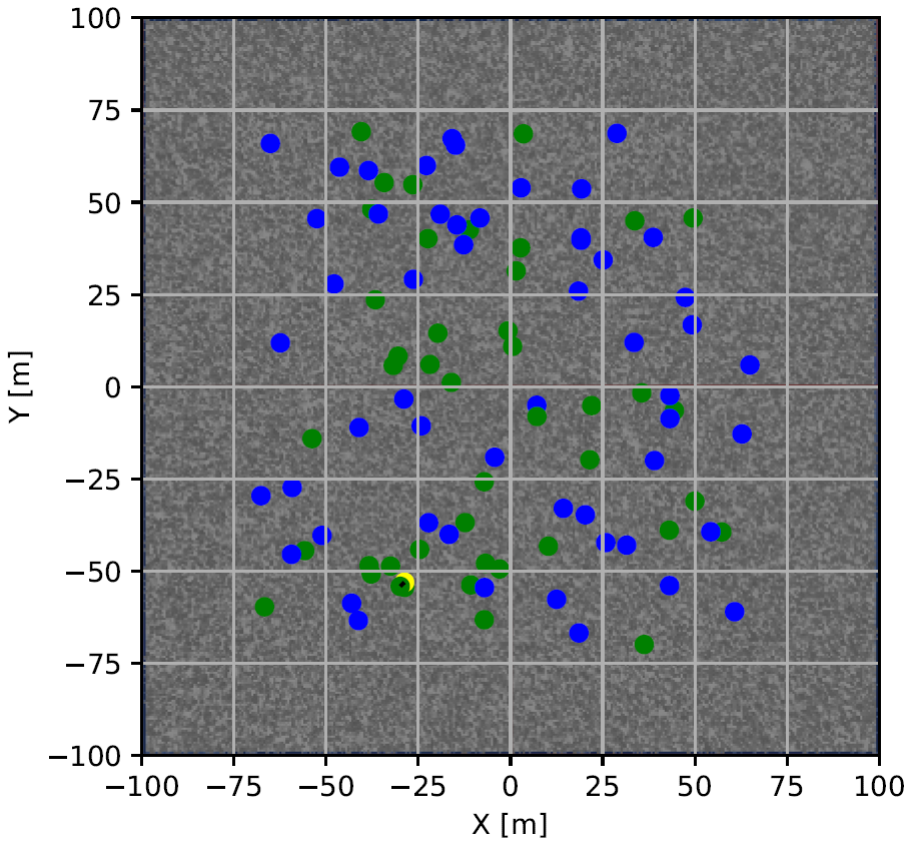


Figure 7.27: Landing attempt locations for the rough map with completely random missions

Like for the case when the platforms were covered by the mission waypoints, the landing sites were almost always verified as they were of high quality by design.

7.4.5 Rough Map - Random WPs with Larger Search Radius

When flying the same experiment setup as introduced in section 7.4.4 but with a larger search pattern radius, we arrive at the following outcome:

Table 7.11: Results - Rough map with platform coverage

# Flights	# Successes	# Con. Losses	# Crashes	# Home Landing
100	98	2	0	34

Numerical Discussion

Compared to the same randomized run on the hostile terrain shown in section 7.4.4, the drone went home only 34 times as opposed to the previous 54. This shows the impact of the additional search patterns flown when no platforms are seen during the mission.
6 times a second landing site had to be chosen. These were all picked on the same platform the initial landing was attempted.

Table 7.12: Landing attempts rough map without platform covering mission

# LS Landings	# Landing on first attempt	# 2 attempts	# 3 attempts
66	60	6	0

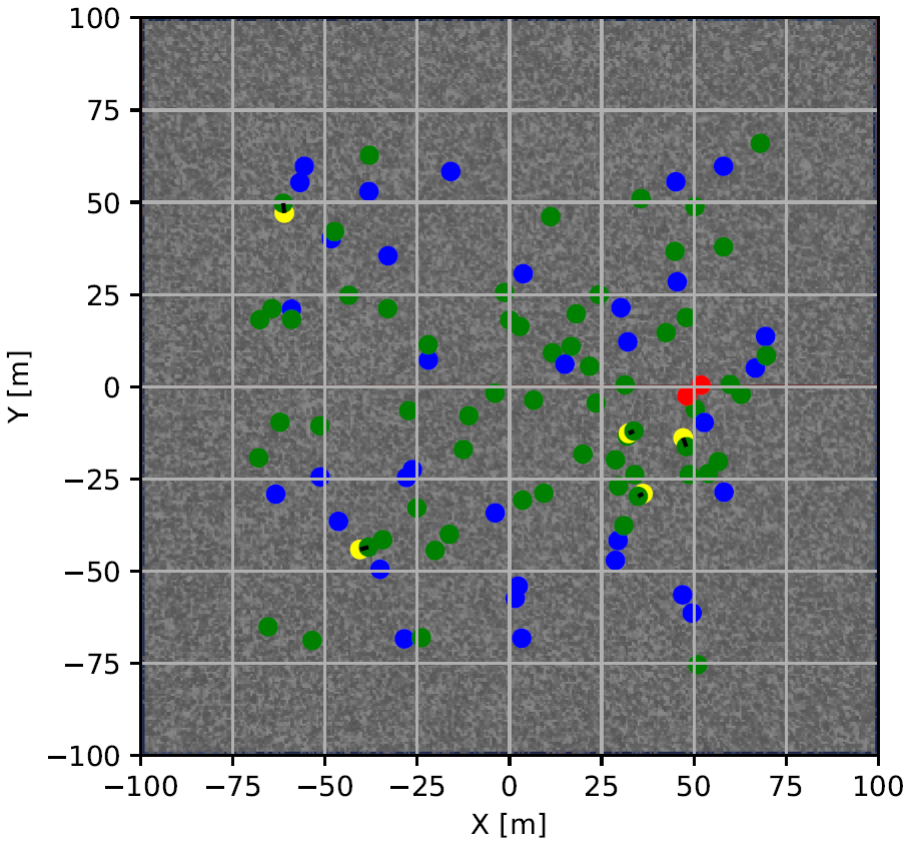


Figure 7.28: Landing attempt locations for the rough map with completely random missions and an increased landing site search action radius

7.4.6 Arroyo - Randomized Takeoff and Mission with SFM

As mentioned in section 7.1, SFM's implementation at the time of this work was not well suited for high-altitude flights. For the sake of completeness, however, the pipeline was flown with SFM and analyzed in the following.
The setup was the same as for section 7.4.2 with randomized spawn and mission locations. However, this time, SFM was used to generate depth.

Table 7.13: Results - SFM with random spawn and way points

# Flights	# Successes	# Con. Losses	# Timeouts	# Home Landing
100	77	18	5	0

Table 7.14: Landing attempts of successful flights when flown with SFM

# Flights	# 1st attempt	# 2 attempts	# 3 attempts
77	45	30	2

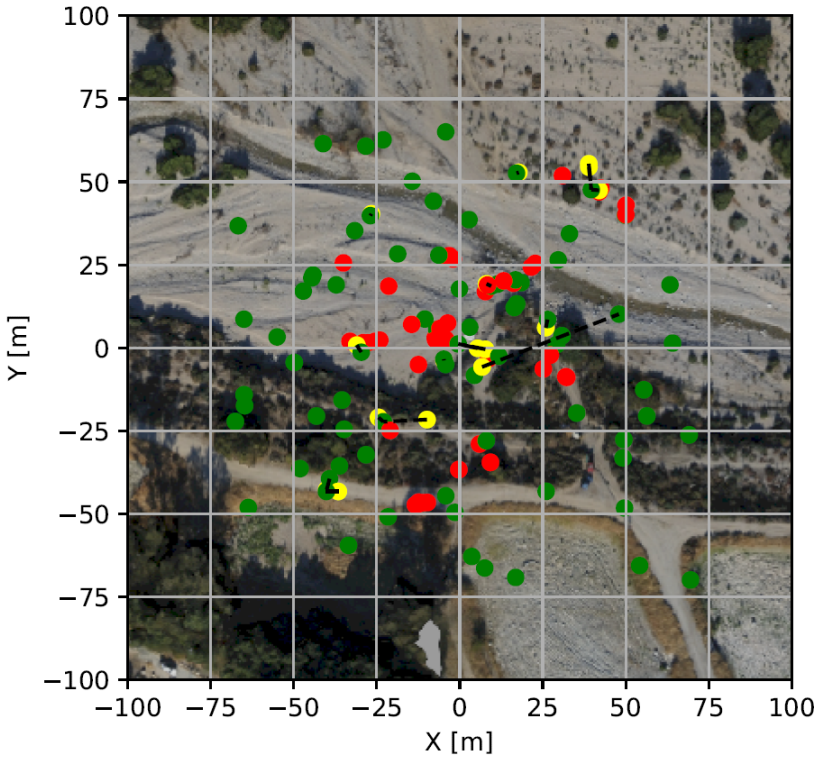


Figure 7.29: Landing attempt locations when flying a mission with randomized drone spawns and randomized waypoints with SFM

Numerical Discussion

Compared to the other tests, in this setting, many connection losses occurred.

Additionally, the nominal procedure timed out five times, meaning that numerous landing site verifications were necessary, which ultimately prevented the flight from completing within the 10 minutes before the next iteration was started.
This occurred because of the following reasons:

- The landing site proposals supplied by SFM are of less quality than the ground truth leading to a higher probability of necessitating the selection of another landing site.
- Secondly, overruling SFM measurements using stereo inputs takes substantial time. In practice the 15 s verification duration was not always sufficient to yield good estimates of the surrounding. Because of this, further landing sites of lower quality were selected, which were detected on SFM data from 100 m altitude. This led to further failures. Secondly, these landing sites were also more often farther away, which made the procedure slower.

This definitely has to be accounted for. As mentioned in section 3.2.2, a potential remedy for excessive convergence could be time inflation of the OMG cells of the elevation map.

Visual Discussion

The first thing to note is the high number of red points. This is both because of the numerous connection losses and because of the landing attempts, which took too long and, therefore, didn't lead to a landing. The landing sites are, in general, of lesser quality than for the GT cases. Lastly, as can be seen in a successful run, subsequent landing sites were not always close by. As previously mentioned, this occurred when the stereo measurements did not overwrite the SFM information in time, so other, far-away SFM landing sites were selected.

7.5 Theoretical Edge Cases

section 7.4 shows that the implemented landing procedure performs well on terrain that is roughly comparable to that on Mars. These tests represent rather ordinary cases, though. In the following, a number of theoretical edge cases are introduced, and their handling by this work's pipeline is discussed.

7.5.1 Arrival on Mars

One concept for the rotorcraft's arrival on Mars is its takeoff during the descent of the main payload. In the Mars Sample Return mission, which has since been aborted, for instance, this payload would have been the landing platform to which the Mars samples would be returned. From that lander, the rotorcraft would take off mid-descent and land on its own.
Playing this scenario through, a small mission or pattern needs to be flown in order to move laterally and thus detect landing sites. Additionally, as the platform from which we took off is descending with neck-breaking velocity, we need to consider a different location for the choice of our home position. This can be done prior to the flight using a mixture of HiRISE images and information provided by Opportunity and Ingenuity. The ladder two options also ensure a sufficient image resolution to determine a home position safely.
If a landing site is successfully detected during this mission, the landing procedure is initiated as usual. If no candidates for suitable landing areas are found, however, further landing site search patterns at random locations are flown. If the landing

site buffer remains empty, the aforementioned synthetic home position is navigated to and landed at.

7.5.2 Flying on Large Scale Inclined Terrain

The tests performed all happened on generally even terrain with rough and inclined areas spread throughout. What happens, however, when we want to fly the drone over large-scale inclined terrain? An example of this could be flying out of the Jezero crater which is where the Opportunity rover was deployed.
In this case, the necessary precautions must be taken during the mission planning stage. Assuming a mission was created at a cruise altitude high enough to allow lateral traversal to the farthest mission waypoint without danger of collision, the landing behavior would be the same.
Two possible break points exist, however:

- Lack of number and quality of landing sites

  Fewer landing sites are detected on inclined terrain. Secondly, a pipeline blind spot is unstable landing sites like larger rocks. Though the probability of landing on such a rock and thus making it fall over is very small, they exist, even more so on inclined terrain.

- Collision danger when flying LS search patterns

  The mission waypoints are set deterministically. Therefore, the terrain can be accounted for during their creation.

  This is not the case for the random flight patterns executed when no landing sites are available. The center point of a rectangular search pattern is set randomly at a fixed distance from the location where the action was invoked. If, during the mission, a landing site was found at a very high altitude and turned out to be a false candidate, the LS search is initiated at that location. This case is schematically depicted in fig. 7.30. Note how the landing site was detected at a significant distance from the last mission waypoint.

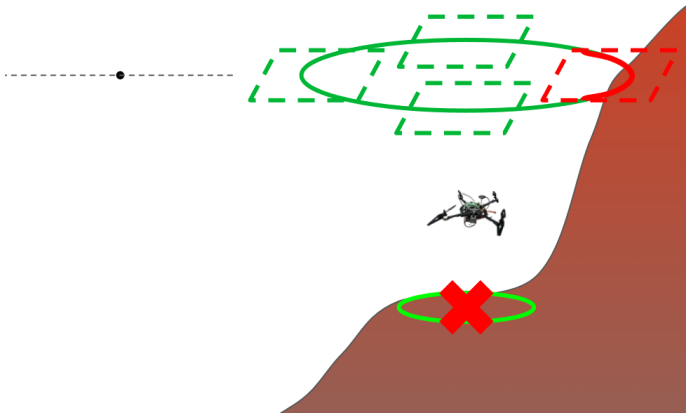


Figure 7.30: Break case of landing site search pattern. The black dot indicates the last mission waypoint at cruise altitude. The bright green circle with the red cross shows an invalid landing site, and the circle above the drone shows the possible center points of search patterns.

One more break case for the presented pipeline is overhanging terrain. In that case, the ascent to a clearing altitude does not constitute a safe approach.

It must be noted, however, that the exploration of terrain as advanced as overhanging ground and cave systems is not considered part of the LORNA endeavor.

Chapter 8

Conclusion

This work added new contributions, and the existing pipeline was used extensively throughout. The following concludes both.

8.1 Visual Pipeline

8.1.1 SFM

First and foremost, The structure from motion approach is a valid approach to create accurate depth information at all altitudes. However, as shown in section 7.1, the current implementation lacks robustness, often crashing on startup or during yaw rotations. This is a fundamental issue that needs to be resolved, as such an occurrence stops the detection of the landing site.
Additionally, due to the jumps in the selected reference keyframe in SFM's stereo implementation, LSD's map is often partially erased, preventing landing sites from being detected until the map cells have sufficiently converged again.

8.1.2 LSD

The landing site detection mechanism showed great robustness and quality regardless of the depth source it was provided with or the altitudes flown at. To detect landing sites at high altitudes, sufficient DEM layers must be used to enable adequate cell sizes at the coarsest layer.
When supplied by different depth sources, it merged the information well and lastly, as tested in section 7.4.3 and section 7.4.4, LSD didn't detect landing sites where there should be none detected, and it did detect sites upon seeing good candidates. In this work, LSD's output was enhanced to not only return a single landing site's location but instead provide three landing sites which are the non-maximum suppressed peaks selected on the binary landing site map. Additionally, these landing sites are assigned further properties like their size, roughness, and slope values. This aids the autonomy in making more informed landing site selection decisions and comparatively weighing the different given characteristics against each other according to the current situation. Furthermore, a new quantity, a landing site's obstacle altitude, was introduced. This allows for efficiently determining the necessary clearing altitude when traversing to a landing site in a fail-safe manner.

8.1.3 Note on the Simulator

The Gazebo simulator used in this project offers both rendering and a physics engine. It is user-friendly and can be set up quickly for less complex endeavors.

However, for our more complex goal, Gazebo proved very cumbersome.

- First, Gazebo is an open-source software, so only very limited documentation can be found.
- The simulator's versioning and compatibility with other software are confusing. This often leads to online documentation not being up-to-date anymore and to the necessity of additional software entities to bridge the gaps with other nodes.
- There were bugs in the source code implementation of the depth camera as described in chapter 13. These could be resolved, but issues like these rattled our trust in the simulator.

8.2 Stereo Camera Depth

To avoid the necessity of lateral motion at low altitudes, a stereo camera was implemented to perceive depth instantaneously and statically. A laser range finder-based switching mechanism allows for the mutually exclusive usage of SFM and stereo-vision, reducing computational overhead. The stereo camera depth implementation showed very convincing results. This was established by comparing LSD debug images when supplied with the stereo camera depth and depth camera ground truth. Furthermore, utilizing the stereo camera depth node at low altitudes and flying autonomous missions, previously detected landing sites were successfully verified or invalidated based on their re-detection status.

8.3 Autonomous Landing Pipeline

An autonomous landing behavior was implemented, leveraging the novel LSD output and the behavior tree architecture within the autonomy framework.
The autonomy's interface with the LSD was updated, introducing new mechanisms like re-detection, verification, and banishment when handling landing sites. This allows for both the refinement of landing site information and the double-checking of a selected landing zone before the final descent.
New modular actions for smooth, safe, and efficient landing preparations were introduced, and the reactive landing behavior was implemented as a behavior tree.
Extensive tests were performed in the Gazebo simulator, using a Mars-like map and a control map without landing sites. Repeated flights were performed with randomized waypoints and fixed and random UAV spawn locations. Despite rare unrelated flight controller connection issues, the runs showed promising results when using ground truth depth and stereo camera depth at low altitudes.
When flying the test missions using structure from motion with the aforementioned current issues, 77 out of 100 flights were successful. The remaining flights either lost connection to the flight controller or timed out verifying the landing sites at low altitudes because the SFM data wasn't overridden in time. Not a single flight crashed, which, considering the current state of SFM, speaks strongly for the pipeline's robustness.[1]
Furthermore, when flying over hazardous terrain that did not have any landing sites on it and spawning synthetic high-quality landing sites over which a mission was flown, the drone landed predominantly on the platform below the last mission

[1] It has to be noted that the timeout occurred due to the limited time assigned to such a randomized test. In practice, a low battery state would eventually initiate the land-at-home action.

waypoint, indicating a well-performing landing site selection mechanism. When flying completely random missions on the otherwise same setup, the drone would perform landing site search patterns starting from randomized center positions. Comparing two such experiments, one flown with a search pattern radius of 20 m and the other with 1 m, the rotorcraft went home 34 and 54 times, respectively. This demonstrates the impact of the landing site search pattern action if no landing sites were detected.
Overall, the work shown in this thesis builds upon the existing LORNA project and combines several modules to achieve an autonomous landing pipeline suited for the challenge of Mars exploration.

Chapter 9

Outlook

9.1 Resolving Prerequisites

Before putting together the entire LORNA pipeline shown in fig. 3.1, the fundamental building blocks need to work well.
Therefore, as the entire LORNA approach builds upon the foundational depth acquisition using structure from motion, this method must work robustly during lateral motion. Specifically, SFM must be more robust concerning yaw and pitch angles, initialization in general, and high-altitude keyframe handling.
LSD is very robust regarding the reception of point clouds from different depth sources. Once LSD's DEM converged, however, incoming measurements sometimes had trouble overwriting the existing information. This could be resolved by adding time inflation to the cells in the elevation map to assign less weight to older measurements.
The map-based localization algorithm, which reduces the global drift of the state estimator, is not crucial for this thesis but a necessary component for the LORNA pipeline. This node was still in development at the time of this writing and would need to be ready to finish the whole LORNA mission concept.

9.2 Putting together the current pipeline

Given that the individual nodes work as desired, the pipeline can be put together. The state estimator can be combined with LORNA's vision-based pipeline and autonomy framework and deployed on embedded hardware.
The question of the embedded processor's computational feasibility will arise, and a decision will have to be made about the processing unit.

9.3 Expansion on the current approach

This project endeavors to fly science missions on Mars with onboard state estimation and autonomous landing capabilities. Naturally, this mission could be expanded in various domains to interact more with the environment. However, in the following, I will outline possible enhancements for this approach to pursue the currently pronounced objectives.
SFM
To make the keyframe switch and the associated LSD map movement less radical, it might be beneficial to change the keyframe selection approach to always switch out the keyframes as long as incoming frames are above a certain quality threshold.

This would lead to more frequent yet smaller map movements, allowing LSD to retain most of the map consistently and, therefore, many more landing sites.
A possible change to slightly increase computational efficiency and ease of understanding would include the stereo camera node within the SFM node. In this case, the laser range finder measurements would decide between the two algorithms to be performed. That way, there is only one depth supplying node that needs to be run during flight.

LSD

As indicated in section 7.2, LSD is currently neglecting most of the information present at high altitudes. An alternative to counteract this would be considering different map sizes at the different layers. That way, the high-resolution layers can retain detailed elevation information at the center of the DEM, while the coarsest layers span a much larger area. When pursuing this implementation, however, the information pooling for DEM consistency across the layers would not be as straightforward.

Autonomy

The modular architecture of the autonomy allows for a manifold of incremental behavior optimizations. A few that come to mind are listed hereafter:

- A possible autonomy enhancement is the introduction of path planning capabilities. The detected landing sites indicate flat, benign terrain and could be used as a prior indication of safely traversable terrain. This knowledge could be used to decrease the necessary safety buffer of the clearing altitude and thus save time and energy ascending to excessive heights.
- Instead of pursuing a single landing site, clusters of potential sites could be considered for initial deployment. That way, less weight is placed on the first site, which is pursued in favor of an increased chance of detecting a high-quality landing site on the second attempt.
- Furthermore, the weighing of the different properties entering the heuristic of a landing site is alterable. This should be used to change the weight distribution of these characteristics depending on the battery level. For instance, the distance to the drone could be weighed higher in case of a low battery level.

Gazebo Simulator

Gazebo, despite being free to use and having a reputation for being user-friendly, will probably have to be replaced for future development. This is because it lacks maintenance, scarce documentation, occasional errors in the software, and a general feeling of unreliability in its performance.

Bibliography

[1] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme, "Vision-based autonomous landing of an unmanned aerial vehicle," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3. IEEE, 2002, pp. 2799–2804.

[2] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics*, Shanghai, China, 2017.

[3] L. Mu, Q. Li, B. Wang, Y. Zhang, N. Feng, X. Xue, and W. Sun, "A vision-based autonomous landing guidance strategy for a micro-uav by the modified camera view," *Drones*, vol. 7, no. 6, p. 400, 2023.

[4] S. Bosch, S. Lacroix, and F. Caballero, "Autonomous detection of safe landing areas for an uav from monocular images," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Beijing: IEEE, 2006.

[5] R. Brockers, P. Bouffard, J. Ma, L. Matthies, and C. Tomlin, "Autonomous landing and ingress of micro-air-vehicles in urban environments based on monocular vision," in *SPIE Defense, Security and Sensing*, 2011.

[6] V. Desaraju, M. Humenberger, N. Michael, R. Brockers, S. Weiss, and L. Matthies, "Vision-based Landing Site Evaluation and Trajectory Generation Toward Rooftop Landing," *Autonomous Robots*, vol. 39, no. 3, pp. 445–463, 2015.

[7] R. Brockers, M. Hummenberger, S. Weiss, and L. Matthies, "Towards autonomous navigation of miniature uav," *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 645–651, 2014.

[8] A. Johnson, A. Klumpp, J. Collier, and A. Wolf, "Lidar-based hazard avoidance for safe landing on Mars," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, no. 5, 2002.

[9] S. Scherer, L. Chamberlain, and S. Singh, "Autonomous landing at unprepared sites by a full-scale helicopter," *Journal of Robotics and Autonomous Systems*, 2012.

[10] A. Johnson, N. Villaume, C. Umsted, A. Kourchians, D. Sternberg, N. Trawny, Y. Cheng, E. Giepel, and J. Montgomery, "The Mars 2020 lander vision system field test," in *Proc. AAS Guidance Navigation and Control Conference (AAS-20-105)*, 2020.

[11] A. Johnson, J. Montgomery, and L. Matthies, "Vision guided landing of an autonomous helicopter in hazardous terrain," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2005.

[12] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[13] S. Daftry, M. Das, J. Delaune, C. Sorice, R. Hewitt, S. Reddy, D. Lytle, E. Gu, and L. Matthies, "Robust vision-based autonomous navigation, mapping and landing for MAVs at night," in *International Symposium on Experimental Robotics (ISER)*, 2018.

[14] F. Neves, L. Branco, M. Pereira, R. Claro, and A. Pinto, "A Multimodal Learning-based Approach for Autonomous Landing of UAV," *arXiv, Mai 21, 2024*, 2024.

[15] S. Abdollahzadeh, P.-L. Proulx, M. Allili, and J.-F. Lapointe, "Safe landing zones detection for uavs using deep regression," in *2022 19th Conference on Robots and Vision (CRV)*, 2022, pp. 213–218.

[16] H. Tovanche-Picon, J. González-Trejo, A. Flores-Abad *et al.*, "Real-time safe validation of autonomous landing in populated areas: from virtual environments to Robot-In-The-Loop," *Virtual Reality*, vol. 28, no. 66, p. 66, 2024.

[17] M. Domnik, P. Proenca, J. Delaune, J. Thiem, and R. Brockers, "Dense 3D-Reconstruction from Monocular Image Sequences for Computationally Constrained UAS," *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021.

[18] P. Schoppmann, P. F. Proenca, J. Delaune, M. Pantic, T. Hinzmann, L. Matthies, R. Siegwart, and R. Brockers, "Multi-Resolution Elevation Mapping and Safe Landing Site Detection with Applications to Planetary Rotorcraft," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[19] P. F. Proenca, J. Delaune, and R. Brockers, "Optimizing Terrain Mapping and Landing Site Detection for Autonomous UAVs," *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[20] Y. Zhao, X. Gao, L. Zhang, Q. Li, R. Fan, and Y. Fu, "Feature-based visual simultaneous localization and mapping: a survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 2, pp. 920–938, 2020.

[21] L. Di Pierno, R. Brockers, and R. Hewitt, "Autonomous Long-Range Flight Execution for Future Mars Rotorcraft," 2024.

[22] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 328-341, 2008.

Chapter 10

Appendix: In-depth flight analysis

In the following, a detailed flight log is shown for the test setting of spawning at a fixed location on the Arroyo map and flying a random mission at 100 m altitude. This example shows an ordinary flight where the drone took off from the ground, flew to a mission way point at cruise altitude and attempted to land twice which the second attempt was successful.

10.0.1 Landing position

As for chapter 7, the landing attempts are shown in fig. 10.1.
The mission way point in this run was quite close to the center. Therefore, it comes to no surprise that a landing site at the start plateau was chosen. One landing site was chosen but not verified and a close by site was subsequently chosen.

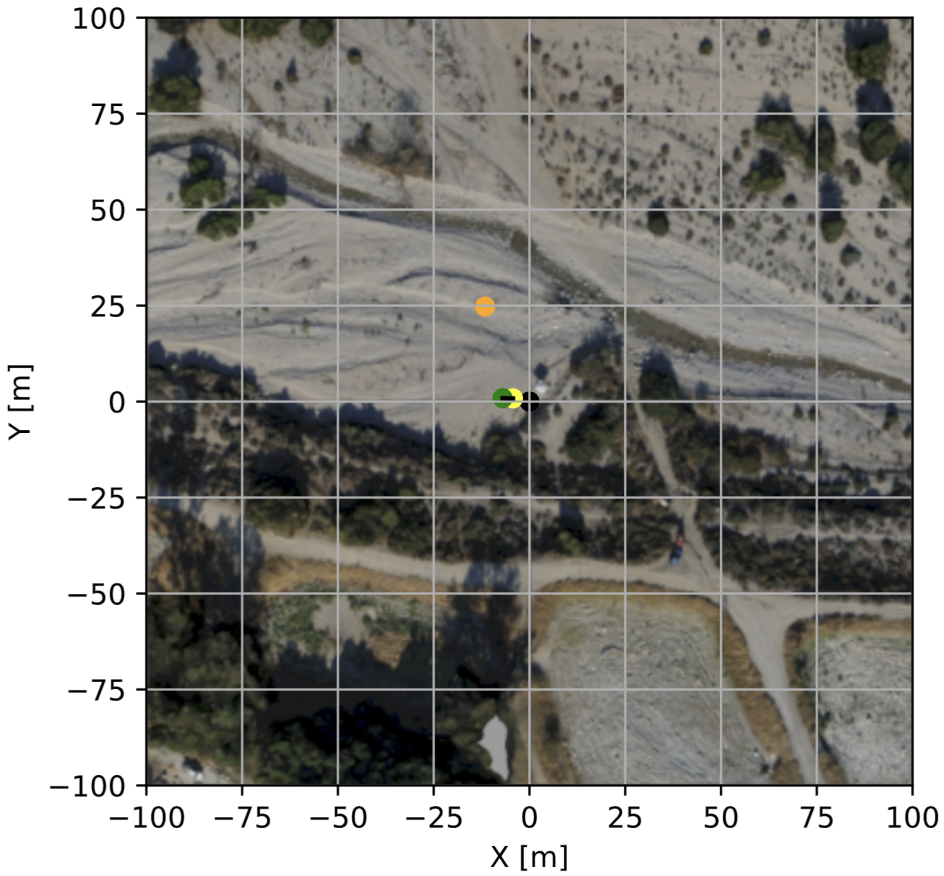


Figure 10.1: Visual Analysis of demo landing: Black is the spawn of the drone, orange indicated the mission way point, yellow the first landing attempt which wasn't verified and green shows the final successful landing attempt.

10.0.2 Autonomy Log

A summarizing log containing the most important information is shown below:

```
Iteration1:
Spawn Randomization Disabled - Drone spawns at arroyo's \n
default position: 0 0 0 0 0 3.5
No additional landing sites spawned.
Mission Waypoints added:
Waypoint 0: (x: -24.739966478190148, y: -11.563968813290693, z: 100.0)
```

```
Evaluation 1
```

```
Run took 384.94275641441345s
Landing Site Log:
LS 1: [2024-04-21_23:44:43] [INFO]  Selecting Landing Site \n
with ID 12 at (x: -0.834371, y: -4.30025, z: -0.4452)

Landing Site Verification 1: Failure
LS 2: [2024-04-21_23:47:26] [INFO]  Selecting Landing Site \n
```

```
with ID 1834 at (x: -0.884371, y: -7.00025, z: -0.533367)

Landing Site Verification 2: Successful
Landing Sequence was initiated
Checks:
Rotation Check: Successful

Conclusion: Success
```

The entire autonomy log is displayed in the following:

```
[2024-04-21_23:42:07] [INFO]  Estimated Time to execute Takeoff \n
and get to Flight Altitude! [Estimated Time: 41.000000 ]
[2024-04-21_23:42:07] [INFO]  State-Machine: \n
State transition table setup
[2024-04-21_23:42:07] [INFO]  Entering State Idle ...
[2024-04-21_23:42:07] [INFO]  State-Machine initialized
[2024-04-21_23:42:07] [INFO]  Initializing Autonomy
[2024-04-21_23:42:07] [INFO]  Waiting 12 seconds for \n
System to boot ...
[2024-04-21_23:42:19] [WARNING]  Failed to Set Parameter[MPC_Z_VEL_MAX_UP]
[2024-04-21_23:42:19] [WARNING]  Failed to Set Parameter[MPC_XY_CRUISE]
[2024-04-21_23:42:19] [WARNING]  Failed to Set Parameter[MPC_YAWRAUTO_MAX]
[2024-04-21_23:42:19] [WARNING]  Failed to Set Parameter[MPC_XY_VEL_ALL]
[2024-04-21_23:42:19] [INFO]  Initialized \n
Navigation Parameters successfully!
[2024-04-21_23:42:19] [WARNING]  Error while \n
Setting the Navigation Parameters!
[2024-04-21_23:42:19] [INFO]  Waiting 12 seconds for \n
System to boot ...
[2024-04-21_23:42:31] [INFO]  Initialized \n
Navigation Parameters successfully!
[2024-04-21_23:42:31] [INFO]  [PX4] GPS activated!
[2024-04-21_23:42:31] [INFO]  [PX4] Height Reference set to GPS!
[2024-04-21_23:42:31] [INFO]  [PX4] External Vision deactivated!
[2024-04-21_23:42:32] [INFO]  System Initialized (with EKF2 GPS Parameters!
[2024-04-21_23:42:32] [INFO]  ROS Parameter Server updated!
[2024-04-21_23:42:32] [INFO]  Starting Autonomous Mission Execution!
[2024-04-21_23:42:32] [EVENT]:  Complete
[2024-04-21_23:42:32] [INFO]  Idle Behavior completed!
[2024-04-21_23:42:33] [EVENT]:  Complete
[2024-04-21_23:42:33] [INFO]  Exiting State Idle ...
[2024-04-21_23:42:33] [STATE TRANSITION]  State transition: Idle -> Init
[2024-04-21_23:42:33] [INFO]  Entering State Init ...
[2024-04-21_23:42:38] [INFO]  Start Pose: \n
-0.277228, -0.0262564, -0.0260597, 0.0273177, -0.0170902, -3.04208
[2024-04-21_23:42:38] [INFO]  Initial Pose and Home Position saved!
[2024-04-21_23:42:38] [INFO]  Ready for GPS flights!
[2024-04-21_23:42:38] [INFO]  Successfully parsed CSV Mission File!
[2024-04-21_23:42:38] [WARNING]  Waypoints already in the \n
Target Reference Frame! [ Reference Frame: LOCAL ]
[2024-04-21_23:42:38] [INFO]  Current Location x: -0.282364 y: -0.0325935
[2024-04-21_23:42:38] [INFO]  First Waypoint Location x: -24.74 y: -11.564
[2024-04-21_23:42:38] [INFO]  First Waypoint is 27.039729 meters away from \n
```

```
the current position! Adding a new Waypoint at the current position!
[2024-04-21_23:42:38] [INFO]  Created Mission!
[2024-04-21_23:42:38] [INFO]  Mission Plan created!
[2024-04-21_23:42:38] [INFO]  Mission Plan ready to execute!
[2024-04-21_23:42:38] [INFO]  Estimator Interface is not used / disabled!
[2024-04-21_23:42:38] [INFO]  Skipped Initializing Healthguard!
[2024-04-21_23:42:38] [EVENT]:  Complete
[2024-04-21_23:42:38] [INFO]  Initialized System!
[2024-04-21_23:42:38] [EVENT]:  Complete
[2024-04-21_23:42:38] [INFO]  Exiting State Init ...
[2024-04-21_23:42:38] [STATE TRANSITION]  State transition: \n
Init -> PreMissionChecks
[2024-04-21_23:42:38] [INFO]  Actuator Checks skipped! Ready to arm!
[2024-04-21_23:42:38] [INFO]  Battery check completed --> \n
Ready for Mission!
[2024-04-21_23:42:38] [EVENT]:  Complete
[2024-04-21_23:42:38] [INFO]  Pre-Checks completed! \n
Ready to arm!
[2024-04-21_23:42:38] [EVENT]:  Complete
[2024-04-21_23:42:38] [STATE TRANSITION]  State transition: \n
PreMissionChecks -> Takeoff
[2024-04-21_23:42:38] [INFO]  Entering Takeoff State ...
[2024-04-21_23:42:38] [INFO]  Maximal Ascending Speed set to: 5.000000 !
[2024-04-21_23:42:38] [INFO]  Estimated Takeoff time is: 41.000000!
[2024-04-21_23:42:38] [WARNING]  Connected to the FCU!
[2024-04-21_23:42:38] [INFO]  Offboard Mode activated!
[2024-04-21_23:42:38] [INFO]  Arming completed!
[2024-04-21_23:42:38] [INFO]  Takeoff Action Started!
[2024-04-21_23:42:38] [WARNING]  Start Altitude is set to: -0.058790 m \n
and the Takeoff Altitude is set to: 1.941210 m!
[2024-04-21_23:42:38] [INFO]  Takeoff Action Running ...
[2024-04-21_23:42:43] [INFO]  Takeoff Action Running ...
[2024-04-21_23:42:48] [INFO]  Takeoff Action Running ...
[2024-04-21_23:42:53] [INFO]  Takeoff Action Running ...
[2024-04-21_23:42:55] [INFO]  Takeoff Waypoint reached!
[2024-04-21_23:42:57] [INFO]  Hold Pose Action completed!
[2024-04-21_23:42:57] [EVENT]:  Complete
[2024-04-21_23:42:57] [INFO]  Takeoff altitude reached!
[2024-04-21_23:42:57] [EVENT]:  Complete
[2024-04-21_23:42:57] [STATE TRANSITION]  State transition: \n
Takeoff -> Mission
[2024-04-21_23:42:57] [INFO]  Entering Mission State ...
[2024-04-21_23:42:57] [INFO]  Current Waypoint: 1 (Out of: 2)
[2024-04-21_23:42:57] [INFO]  Mission state was set to \n
NavigateToWaypoint.
[2024-04-21_23:42:57] [INFO]  Mission initialized!
[2024-04-21_23:42:57] [INFO]  In total 2 waypoints to fly
[2024-04-21_23:44:06] [INFO]  Waypoint reached!
[2024-04-21_23:44:06] [INFO]  All current Waypoint-Actions completed!
[2024-04-21_23:44:06] [INFO]  Current Waypoint: 2 (Out of: 2)
[2024-04-21_23:44:06] [INFO]  Mission state was set to \n
NavigateToWaypoint.
[2024-04-21_23:44:42] [INFO]  Waypoint reached!
[2024-04-21_23:44:42] [INFO]  Velocity set! (Setpoint: 2.000000 m/s))
```

```
[2024-04-21_23:44:43] [INFO]  Holding position finished!
[2024-04-21_23:44:43] [INFO]  All current Waypoint-Actions completed!
[2024-04-21_23:44:43] [INFO]  Mission Complete!
[2024-04-21_23:44:43] [EVENT]:  Complete
[2024-04-21_23:44:43] [INFO]  Mission Complete!
[2024-04-21_23:44:43] [EVENT]:  Complete
[2024-04-21_23:44:43] [INFO]  Exiting Mission State ...
[2024-04-21_23:44:43] [STATE TRANSITION]  State transition: \n
Mission -> Land
[2024-04-21_23:44:43] [INFO]  Entering State_Land...
[2024-04-21_23:44:43] [INFO]  Initializing Landing Tree ...
[2024-04-21_23:44:43] [INFO]  Landing Tree initialized!
[2024-04-21_23:44:43] [INFO]  Check Landing Site Action
[2024-04-21_23:44:43] [INFO]  Landing Site List Check: \n
larger than 0
[2024-04-21_23:44:43] [INFO]  GetLandingSiteAction
[2024-04-21_23:44:43] [INFO]  Getting Best landing sites \n
at current location x: -25.013 y: -11.2604 z: 100.056
[2024-04-21_23:44:43] [INFO]  Loss of LS 12 at position \n
x:-0.834371 y: -4.30025 z: -0.4452 loss: 0.864074, \n
contrib size: -0.649404, contrib dist: 1.03603, \n
contrib veralt: 0.0890372, contrib var: 0.000254673, \n
contrib rough: 0.0498131
[2024-04-21_23:44:43] [INFO]  Loss of LS 42 at position \n
x:-4.28437 y: -4.65025 z: -0.450161 loss: 0.983715, \n
contrib size: -0.742495, contrib dist: 1.02834, \n
contrib veralt: 1.00468, contrib var: 0.00103761, \n
contrib rough: 0.0762918
[2024-04-21_23:44:43] [INFO]  Loss of LS 33 at position \n
x:-3.98437 y: -8.75025 z: -0.580863 loss: 0.993144, \n
contrib size: -0.371124, contrib dist: 1.02841, \n
contrib veralt: 0.148057, contrib var: 0.000307739, \n
contrib rough: 0.0574909
[2024-04-21_23:44:43] [INFO]  LSM: Landing Site has been selected
[2024-04-21_23:44:43] [INFO]  Selecting Landing Site with ID 12 at \n
(x: -0.834371, y: -4.30025, z: -0.4452)
[2024-04-21_23:44:43] [INFO]  GetLandingSiteAction: \n
Initial altitude was defined by current altitude which is 99.9901
[2024-04-21_23:44:43] [WARNING]  GetLandingSiteAction: \n
Start altitude is above failsafe altitude - choosing start altitude.
[2024-04-21_23:44:43] [INFO]  Landing Site found!
[2024-04-21_23:44:43] [INFO]  AscendToClearAltitudeAction: \n
Altitude reached!
[2024-04-21_23:44:43] [INFO]  RotateTowardsWaypointAction
[2024-04-21_23:44:43] [INFO]  NavigateTowardsWaypointAction
[2024-04-21_23:44:43] [INFO]  Successfully set velocity to navigate to \n
waypoint with coordinates: x: -0.834371, y: -4.30025, z: 99.9901
[2024-04-21_23:45:22] [INFO]  Reached Pose with coordinates: \n
x: -0.834371, y: -4.30025, z: 99.9901
[2024-04-21_23:45:22] [INFO]  ChangeAltitudeLSAction to 2.0548 which is \n
2.5m above the landing site
[2024-04-21_23:47:11] [INFO]  ChangeAltitudeLSAction: \n
Altitude reached!
[2024-04-21_23:47:26] [INFO]  Hold Pose Action completed!
```

```
[2024-04-21_23:47:26] [INFO]  Landing Site Verification Action at: \n
2.14322 meters altitude
[2024-04-21_23:47:26] [WARNING]  Landing site verification failed at \n
2.14322 meters. Verification altitude was: 8.90372 Resetting Landing site...
[2024-04-21_23:47:26] [WARNING]  Removing Landing Site with ID12
[2024-04-21_23:47:26] [WARNING]  Remaining IDs: \n
589, 213, 206, 41, 217, 205, 52, 57, 45, 55, 59, 51, 204, \n
49, 207, 202, 208, 203, 210, 1834,
[2024-04-21_23:47:26] [INFO]  Check Landing Site Action
[2024-04-21_23:47:26] [INFO]  Landing Site List Check: larger than 0
[2024-04-21_23:47:26] [INFO]  GetLandingSiteAction
[2024-04-21_23:47:26] [INFO]  Getting Best landing sites at current location \r
x: -2.05054 y: -2.97383 z: 2.13485
[2024-04-21_23:47:26] [INFO]  Loss of LS 1834 at position \n
x:-0.884371 y: -7.00025 z: -0.533367 loss: -0.315955, \n
contrib size: -0.798207, contrib dist: 0.0496904, \n
contrib veralt: 0.0266822, contrib var: 0.000306336, \n
contrib rough: 0.0359142
[2024-04-21_23:47:26] [INFO]  Loss of LS 589 at position \n
x:-4.33437 y: -7.55025 z: -0.598968 loss: -0.119794, \n
contrib size: -0.725, contrib dist: 0.0579942, \n
contrib veralt: 0.657236, contrib var: 0.103212, \n
contrib rough: 0.0608913
[2024-04-21_23:47:26] [INFO]  Loss of LS 206 at position \n
x:-5.18437 y: -8.10025 z: -0.62499 loss: -0.0702529, \n
contrib size: -0.65625, contrib dist: 0.0661194, \n
contrib veralt: 0.657496, contrib var: 0.10416, \n
contrib rough: 0.0218446
[2024-04-21_23:47:26] [INFO]  LSM: Landing Site has been selected
[2024-04-21_23:47:26] [INFO]  Selecting Landing Site with ID 1834 at \n
(x: -0.884371, y: -7.00025, z: -0.533367)
[2024-04-21_23:47:26] [INFO]  GetLandingSiteAction: \n
Initial altitude was defined by highest detected obstacle at 2.73586
[2024-04-21_23:47:26] [INFO]  XY-Distance based interpolation yielded \n
clearing altitude of 8.93774
[2024-04-21_23:47:26] [INFO]  Landing Site found!
[2024-04-21_23:47:51] [INFO]  AscendToClearAltitudeAction: \n
Altitude reached!
[2024-04-21_23:47:51] [INFO]  RotateTowardsWaypointAction
[2024-04-21_23:47:51] [INFO]  NavigateTowardsWaypointAction
[2024-04-21_23:47:51] [INFO]  Successfully set velocity \n
to navigate to waypoint with coordinates: \n
x: -0.884371, y: -7.00025, z: 8.93774
[2024-04-21_23:48:20] [INFO]  Reached Pose with coordinates: \n
x: -0.884371, y: -7.00025, z: 8.93774
[2024-04-21_23:48:20] [INFO]  ChangeAltitudeLSAction \n
to 1.96775 which is 2.5m above the landing site
[2024-04-21_23:48:32] [INFO]  ChangeAltitudeLSAction: \n
Altitude reached!
[2024-04-21_23:48:47] [INFO]  Hold Pose Action completed!
[2024-04-21_23:48:47] [INFO]  Landing Site Verification Action at: \n
2.22133 meters altitude
[2024-04-21_23:48:47] [INFO]  Landing site was verified at \n
2.75358 meters off the ground
```

```
[2024-04-21_23:48:47] [INFO] Landing at \n
(-0.584371, -7.50025, -0.53225)
[2024-04-21_23:48:47] [INFO] Landing Site Location Updated!
[2024-04-21_23:48:47] [INFO] NavigateTowardsWaypointAction
[2024-04-21_23:48:47] [INFO] Successfully set velocity to \n
navigate to waypoint with coordinates: \n
x: -0.584371, y: -7.50025, z: 2.22133
[2024-04-21_23:48:51] [INFO] Reached Pose with coordinates: \n
x: -0.584371, y: -7.50025, z: 2.22133
[2024-04-21_23:48:51] [INFO] Landing Action: \n
Landing at Z of detected landing site.
[2024-04-21_23:48:54] [INFO] Landing ...
[2024-04-21_23:48:57] [INFO] Landing ...
[2024-04-21_23:49:00] [INFO] Landing ...
[2024-04-21_23:49:03] [INFO] Landing ...
[2024-04-21_23:49:06] [INFO] Landing ...
[2024-04-21_23:49:06] [INFO] Landing detected!
[2024-04-21_23:49:06] [INFO] Disarmed!
[2024-04-21_23:49:06] [EVENT]: Complete
[2024-04-21_23:49:06] [INFO] Landing Behavior completed!
[2024-04-21_23:49:06] [EVENT]: Complete
[2024-04-21_23:49:06] [STATE TRANSITION] \n
State transition: Land -> Terminate
[2024-04-21_23:49:06] [INFO] Vehicle is landed!
```

10.0.3 Flight Controller Connection Loss

Lastly, hereafter I show a run which lost connection to the flight controller mid-flight:

```
[2024-04-23_22:00:08] [INFO] State-Machine: \n
State transition table setup
[2024-04-23_22:00:08] [INFO] Entering State Idle ...
[2024-04-23_22:00:08] [INFO] State-Machine initialized
[2024-04-23_22:00:08] [INFO] Initializing Autonomy
[2024-04-23_22:00:08] [INFO] Waiting 12 seconds for System to boot ...
[2024-04-23_22:00:20] [WARNING] Failed to Set \n
Parameter[MPC_Z_VEL_MAX_UP]
[2024-04-23_22:00:20] [WARNING] Failed to Set \n
Parameter[MPC_XY_CRUISE]
[2024-04-23_22:00:20] [WARNING] Failed to Set \n
Parameter[MPC_YAWRAUTO_MAX]
[2024-04-23_22:00:20] [WARNING] Failed to Set \n
Parameter[MPC_XY_VEL_ALL]
[2024-04-23_22:00:20] [INFO] Initialized \n
Navigation Parameters successfully!
[2024-04-23_22:00:20] [WARNING] Error while Setting \n
the Navigation Parameters!
[2024-04-23_22:00:20] [INFO] Waiting 12 seconds for \n
System to boot ...
[2024-04-23_22:00:32] [INFO] Initialized \n
Navigation Parameters successfully!
[2024-04-23_22:00:32] [INFO] [PX4] GPS activated!
[2024-04-23_22:00:32] [INFO] [PX4] Height Reference set to GPS!
```

```
[2024-04-23_22:00:32] [INFO]  [PX4] External Vision deactivated!
[2024-04-23_22:00:33] [INFO]  System Initialized (with EKF2 GPS Parameters!
[2024-04-23_22:00:33] [INFO]  ROS Parameter Server updated!
[2024-04-23_22:00:33] [INFO]  Starting Autonomous \n
Mission Execution!
[2024-04-23_22:00:34] [EVENT]:  Complete
[2024-04-23_22:00:34] [INFO]  Idle Behavior completed!
[2024-04-23_22:00:34] [EVENT]:  Complete
[2024-04-23_22:00:34] [INFO]  Exiting State Idle ...
[2024-04-23_22:00:34] [STATE TRANSITION]  State transition: \n
Idle -> Init
[2024-04-23_22:00:34] [INFO]  Entering State Init ...
[2024-04-23_22:00:39] [INFO]  Start Pose: \n
59.4398, 32.3157, 39.8456, 0.00186781, -9.09566e-05, -0.00994173
[2024-04-23_22:00:39] [INFO]  Initial Pose and \n
Home Position saved!
[2024-04-23_22:00:39] [INFO]  Ready for GPS flights!
[2024-04-23_22:00:39] [INFO]  Successfully parsed CSV Mission File!
[2024-04-23_22:00:39] [WARNING]  Waypoints already in the \n
Target Reference Frame! [ Reference Frame: LOCAL ]
[2024-04-23_22:00:39] [INFO]  Current Location \n
x: 59.4394 y: 32.3134
[2024-04-23_22:00:39] [INFO]  First Waypoint Location \n
x: -12.7774 y: -19.8106
[2024-04-23_22:00:39] [INFO]  First Waypoint is \n
89.062746 meters away from the current position! \n
Adding a new Waypoint at the current position!
[2024-04-23_22:00:39] [INFO]  Created Mission!
[2024-04-23_22:00:39] [INFO]  Mission Plan created!
[2024-04-23_22:00:39] [INFO]  Mission Plan ready to execute!
[2024-04-23_22:00:39] [INFO]  Estimator Interface is not used / disabled!
[2024-04-23_22:00:39] [INFO]  Skipped Initializing Healthguard!
[2024-04-23_22:00:39] [EVENT]:  Complete
[2024-04-23_22:00:39] [INFO]  Initialized System!
[2024-04-23_22:00:39] [EVENT]:  Complete
[2024-04-23_22:00:39] [INFO]  Exiting State Init ...
[2024-04-23_22:00:39] [STATE TRANSITION]  State transition: \n
Init -> PreMissionChecks
[2024-04-23_22:00:39] [INFO]  Actuator Checks skipped! \n
Ready to arm!
[2024-04-23_22:00:39] [INFO]  Battery check completed --> \n
Ready for Mission!
[2024-04-23_22:00:39] [EVENT]:  Complete
[2024-04-23_22:00:39] [INFO]  Pre-Checks completed! \n
Ready to arm!
[2024-04-23_22:00:39] [EVENT]:  Complete
[2024-04-23_22:00:39] [STATE TRANSITION]  State transition: \n
PreMissionChecks -> Takeoff
[2024-04-23_22:00:39] [INFO]  Entering Takeoff State ...
[2024-04-23_22:00:39] [INFO]  Maximal Ascending Speed set to: 5.000000 !
[2024-04-23_22:00:39] [INFO]  Estimated Takeoff time is: 41.000000!
[2024-04-23_22:00:39] [WARNING]  Connected to the FCU!
[2024-04-23_22:00:39] [INFO]  Offboard Mode activated!
[2024-04-23_22:00:39] [INFO]  Arming completed!
```

```
[2024-04-23_22:00:39] [INFO]  Takeoff Action Started!
[2024-04-23_22:00:39] [WARNING]  Start Altitude is set to: \n
39.860235 m and the Takeoff Altitude is set to: 41.860235 m!
[2024-04-23_22:00:39] [INFO]  Takeoff Action Running ...
[2024-04-23_22:00:44] [INFO]  Takeoff Action Running ...
[2024-04-23_22:00:49] [INFO]  Takeoff Action Running ...
[2024-04-23_22:00:54] [INFO]  Takeoff Action Running ...
[2024-04-23_22:00:59] [INFO]  Takeoff Action Running ...
[2024-04-23_22:01:04] [INFO]  Takeoff Action Running ...
[2024-04-23_22:01:08] [INFO]  Takeoff Waypoint reached!
[2024-04-23_22:01:10] [INFO]  Hold Pose Action completed!
[2024-04-23_22:01:10] [EVENT]:  Complete
[2024-04-23_22:01:10] [INFO]  Takeoff altitude reached!
[2024-04-23_22:01:10] [EVENT]:  Complete
[2024-04-23_22:01:10] [STATE TRANSITION]  State transition: \n
Takeoff -> Mission
[2024-04-23_22:01:10] [INFO]  Entering Mission State ...
[2024-04-23_22:01:10] [INFO]  Current Waypoint: 1 (Out of: 2)
[2024-04-23_22:01:10] [INFO]  Mission state was set to \n
NavigateToWaypoint.
[2024-04-23_22:01:10] [INFO]  Mission initialized!
[2024-04-23_22:01:10] [INFO]  In total 2 waypoints to fly
[2024-04-23_22:01:53] [INFO]  Waypoint reached!
[2024-04-23_22:01:53] [INFO]  All current Waypoint-Actions completed!
[2024-04-23_22:01:53] [INFO]  Current Waypoint: 2 (Out of: 2)
[2024-04-23_22:01:53] [INFO]  Mission state was set to \n
NavigateToWaypoint.
[2024-04-23_22:03:24] [INFO]  Waypoint reached!
[2024-04-23_22:03:24] [INFO]  Velocity set! (Setpoint: 2.000000 m/s))
[2024-04-23_22:03:25] [INFO]  Holding position finished!
[2024-04-23_22:03:25] [INFO]  All current Waypoint-Actions completed!
[2024-04-23_22:03:25] [INFO]  Mission Complete!
[2024-04-23_22:03:25] [EVENT]:  Complete
[2024-04-23_22:03:25] [INFO]  Mission Complete!
[2024-04-23_22:03:25] [EVENT]:  Complete
[2024-04-23_22:03:25] [INFO]  Exiting Mission State ...
[2024-04-23_22:03:25] [STATE TRANSITION]  State transition: \n
Mission -> Land
[2024-04-23_22:03:25] [INFO]  Entering State_Land...
[2024-04-23_22:03:25] [INFO]  Initializing Landing Tree ...
[2024-04-23_22:03:25] [INFO]  Landing Tree initialized!
[2024-04-23_22:03:25] [INFO]  Check Landing Site Action
[2024-04-23_22:03:25] [INFO]  Landing Site List Check: \n
larger than 0
[2024-04-23_22:03:25] [INFO]  GetLandingSiteAction
[2024-04-23_22:03:26] [INFO]  Getting Best landing sites at current location \n
x: -12.5608 y: -19.6724 z: 99.9481
[2024-04-23_22:03:26] [INFO]  Loss of LS 499 at position \n
x:-0.470386 y: -12.2437 z: -0.597542 loss: 0.916079, \n
contrib size: -0.997011, contrib dist: 1.01542, \n
contrib veralt: 1.00566, contrib var: 0.189603, \n
contrib rough: 0.0114712
[2024-04-23_22:03:26] [INFO]  Loss of LS 58 at position \n
x:27.3796 y: -1.14372 z: 3.56173 loss: 1.38689, \n
```

```
contrib size: -0.192888, contrib dist: 1.05966, \n
contrib veralt: 0.964176, contrib var: 0.139055, \n
contrib rough: 0.0408271
[2024-04-23_22:03:26] [INFO] Loss of LS 39 at position \n
x:28.0296 y: -0.293723 z: 3.59092 loss: 1.40956, \n
contrib size: -0.110222, contrib dist: 1.06338, \n
contrib veralt: 0.963344, contrib var: 0.140886, \n
contrib rough: 0.0481275
[2024-04-23_22:03:26] [INFO] LSM: Landing Site has been selected
[2024-04-23_22:03:26] [INFO] Selecting Landing Site with ID 499 at \n
(x: -0.470386, y: -12.2437, z: -0.597542)
[2024-04-23_22:03:26] [INFO] GetLandingSiteAction: \n
Initial altitude was defined by current altitude which is 99.924
[2024-04-23_22:03:26] [WARNING] GetLandingSiteAction: \n
Start altitude is above failsafe altitude - choosing start altitude.
[2024-04-23_22:03:26] [INFO] Landing Site found!
[2024-04-23_22:03:26] [INFO] AscendToClearAltitudeAction: \n
Altitude reached!
[2024-04-23_22:03:26] [INFO] RotateTowardsWaypointAction
[2024-04-23_22:03:26] [INFO] NavigateTowardsWaypointAction
[2024-04-23_22:03:26] [INFO] Successfully set velocity to \n
navigate to waypoint with coordinates: x: -0.470386, y: -12.2437, z: 99.924
[2024-04-23_22:03:43] [INFO] Reached Pose with coordinates: \n
x: -0.470386, y: -12.2437, z: 99.924
[2024-04-23_22:03:43] [INFO] ChangeAltitudeLSAction to \n
1.87749 which is 2.5m above the landing site
```

As can be seen, the flight controller connection was lost shortly after the descent to the verification altitude was commenced.

Chapter 11

Appendix: LSD Layer Analysis

In the following, a visual comparison of the LSD output with different numbers of layers is shown. Note that better results were achieved using more layers. This, however, comes at the price of higher computational overhead. A tradeoff must be found to achieve good LSD performance on a limited embedded processor.

11.1 2 Layers

fig. 11.1 shows the issue of flying with too few layers. At two layers, the coarsest resolution is too detailed for the incoming points at high altitude. Therefore, by the time LSD converges on the SFM information, SFM already exchanges the keyframe which leads to a map movement and therefore the loss of information. Because of this, LSD has a hard time detecting any landing sites.

11.2 3 Layers

fig. 11.2 shows the same chain of events but with 3 layers. Note that the input is less sparse and the DEM is able to detect landing sites before the map is moved.

11.3 4 Layers

Again fig. 11.3 shows the map updates for 4 layers. With this setting, landing sites were perceived more consistently and much faster.

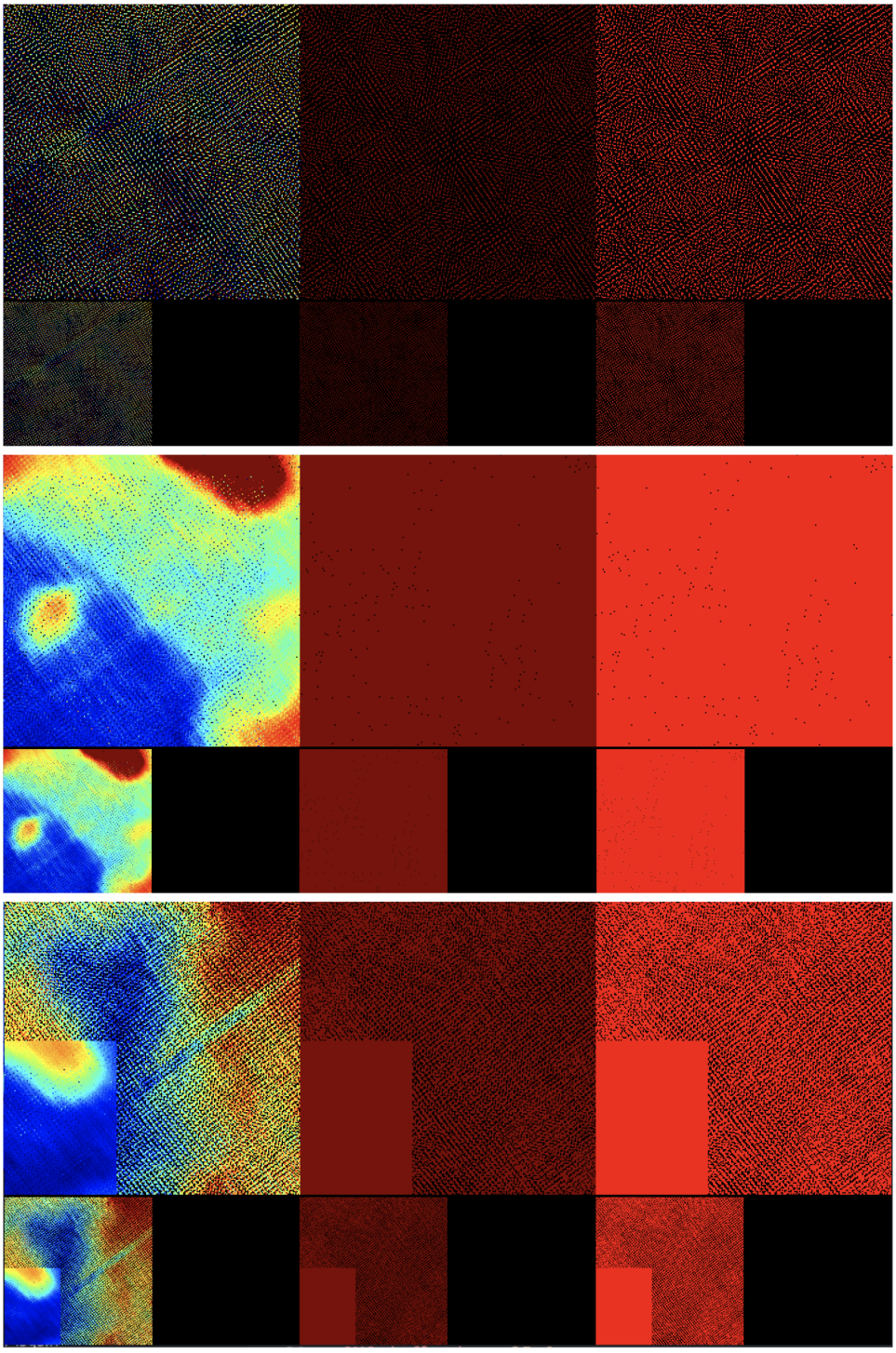


Figure 11.1: LSD DEM with two layers: The top third shows the map after it received an iteration of sparse inputs from SFM. The middle image shows the map after some conversion time. The bottom third shows the map after the map movement. Note the retained information in the bottom left corner

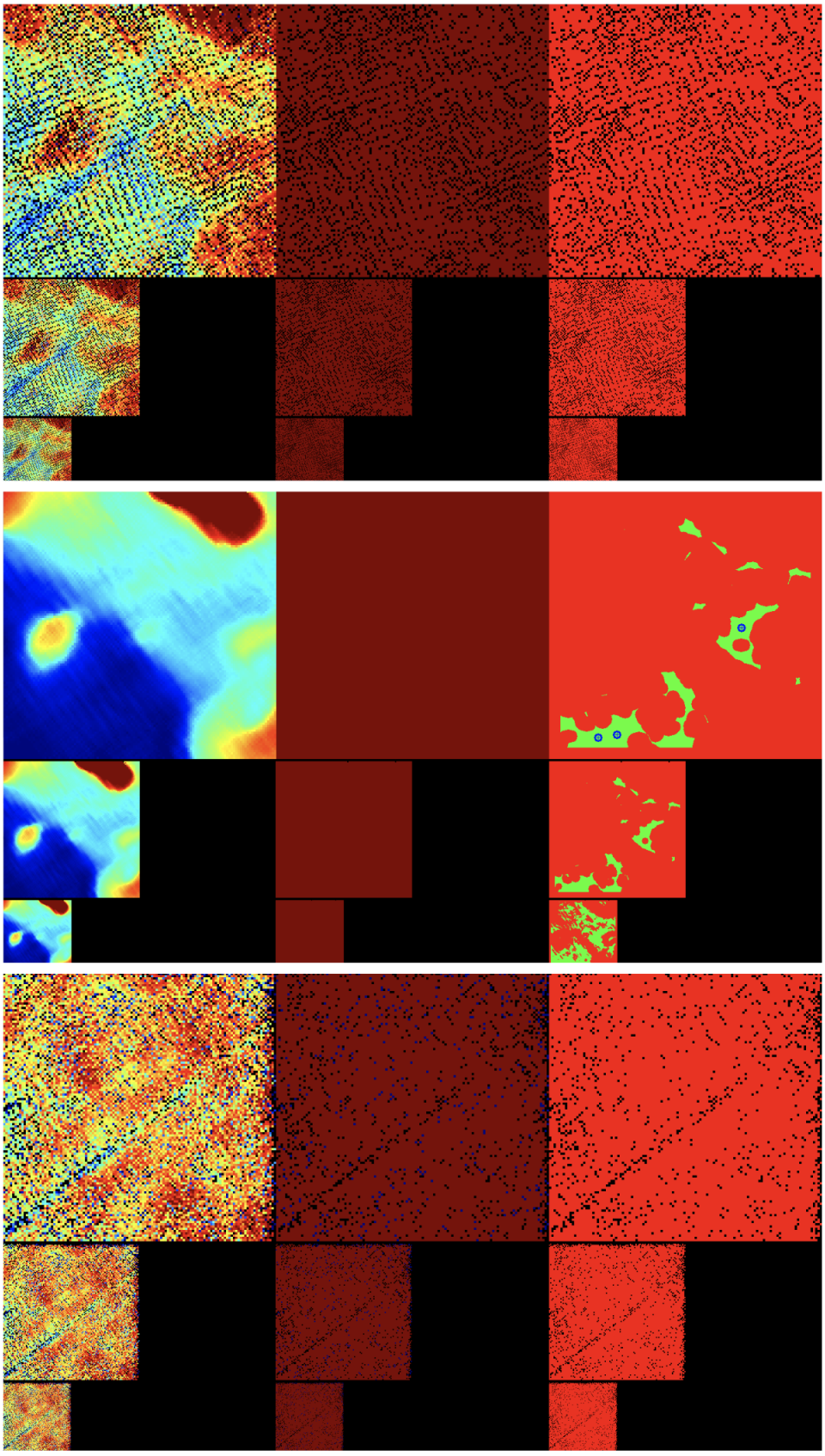


Figure 11.2: LSD DEM with three layers: The top third shows the map after it received an iteration of sparse inputs from SFM. The middle image shows the map after some conversion time. The bottom third shows the map after the map movement. Note the retained information in the bottom left corner

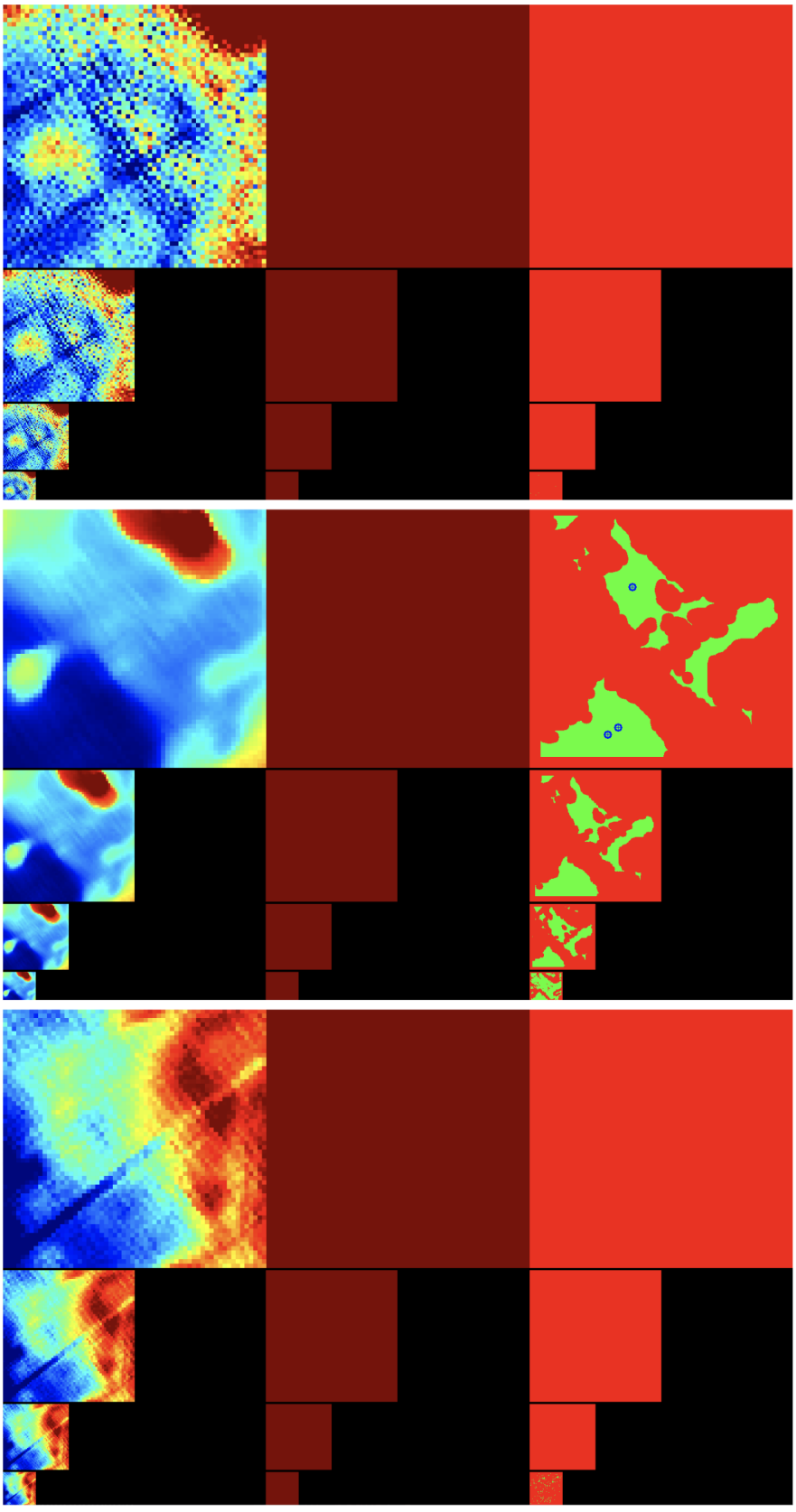


Figure 11.3: LSD DEM with four layers: The top third shows the map after it received an iteration from SFM. The middle image shows the map after some conversion time. The bottom third shows the map after the map movement. In this case, the keyframe location difference before and after the movement was so big that the entire DEM was emptied and the LS segmentation doesn't find valid sites again.

Chapter 12

Drone Hardware

The drone used at the time of writing this thesis is a Snapdragon drone with a Pixhawk flight controller and a VOXL2 embedded processor. fig. 12.1 shows an image of this rotorcraft hardware. On this drone setup there is a tracking camera and a stereo camera. Note that the landing supports of the drone are significantly farther apart than for the model in the simulation. This is why the simulated stereo camera in Gazebo had to be placed further away from the drone while in reality it was mounted normally on the drone's body.

Figure 12.1: Snapdragon drone setup

fig. 12.2 shows the stereo camera according to which the simulation model was implemented.

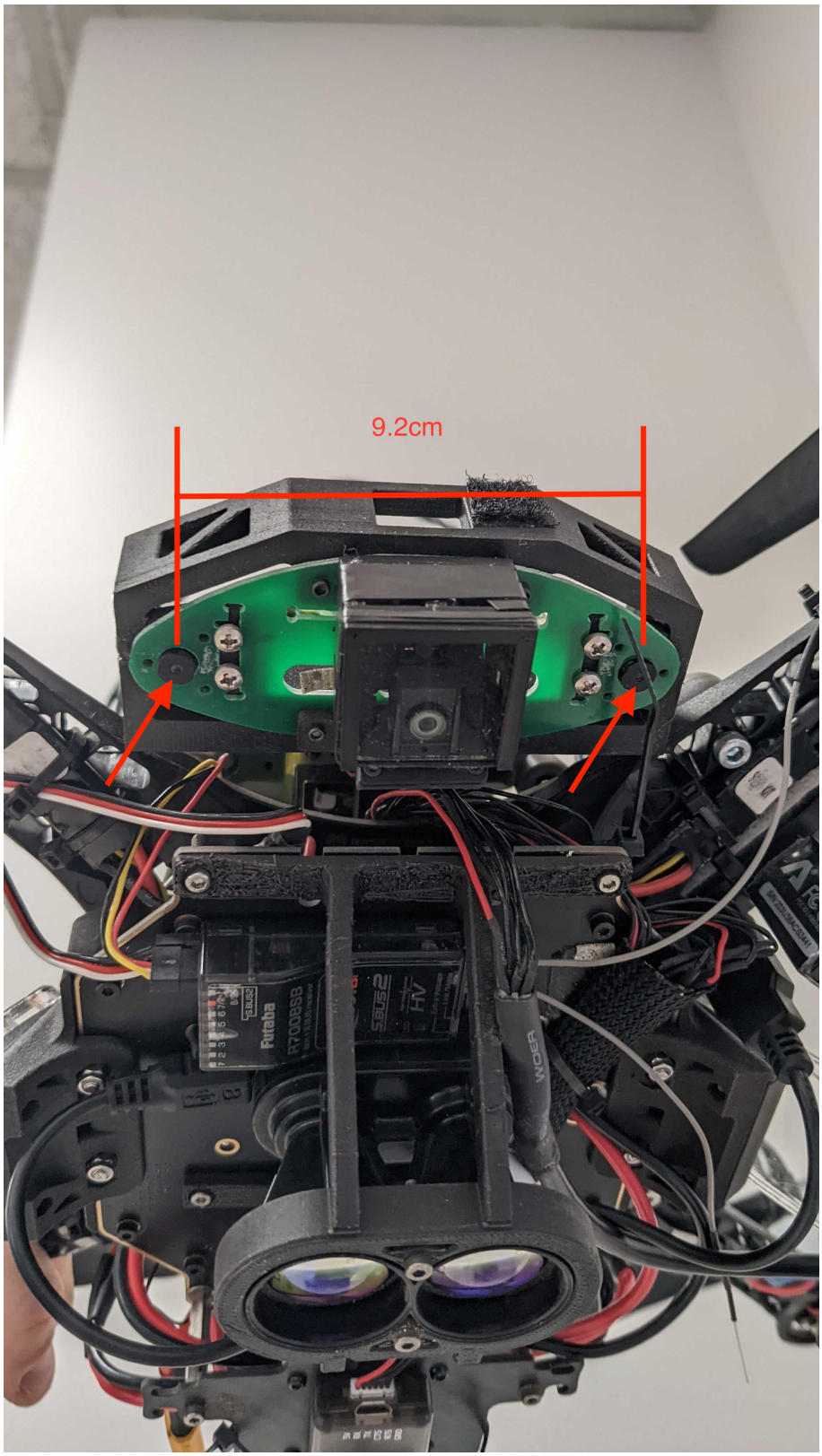


Figure 12.2: Stereo camera on physical drone

Chapter 13

Appendix: Simulation Depth Camera Misalignment

When first attempting to implement a depth camera as ground truth I noticed a misalignment of the depth camera FOV and a normal camera's image. This was the case even with equal camera parameters:
Looking at fig. 13.1, we can see a misalignment of the depth camera's field of view and the normal reference camera at the same location with the same parameters. Later tests showed that there was an actual error in the Gazebo Garden source code. No matter what camera parameters were passed, the depth camera had the same FOV.
Thanks to my friend and colleague Simone Nascivera, this bug could be fixed.

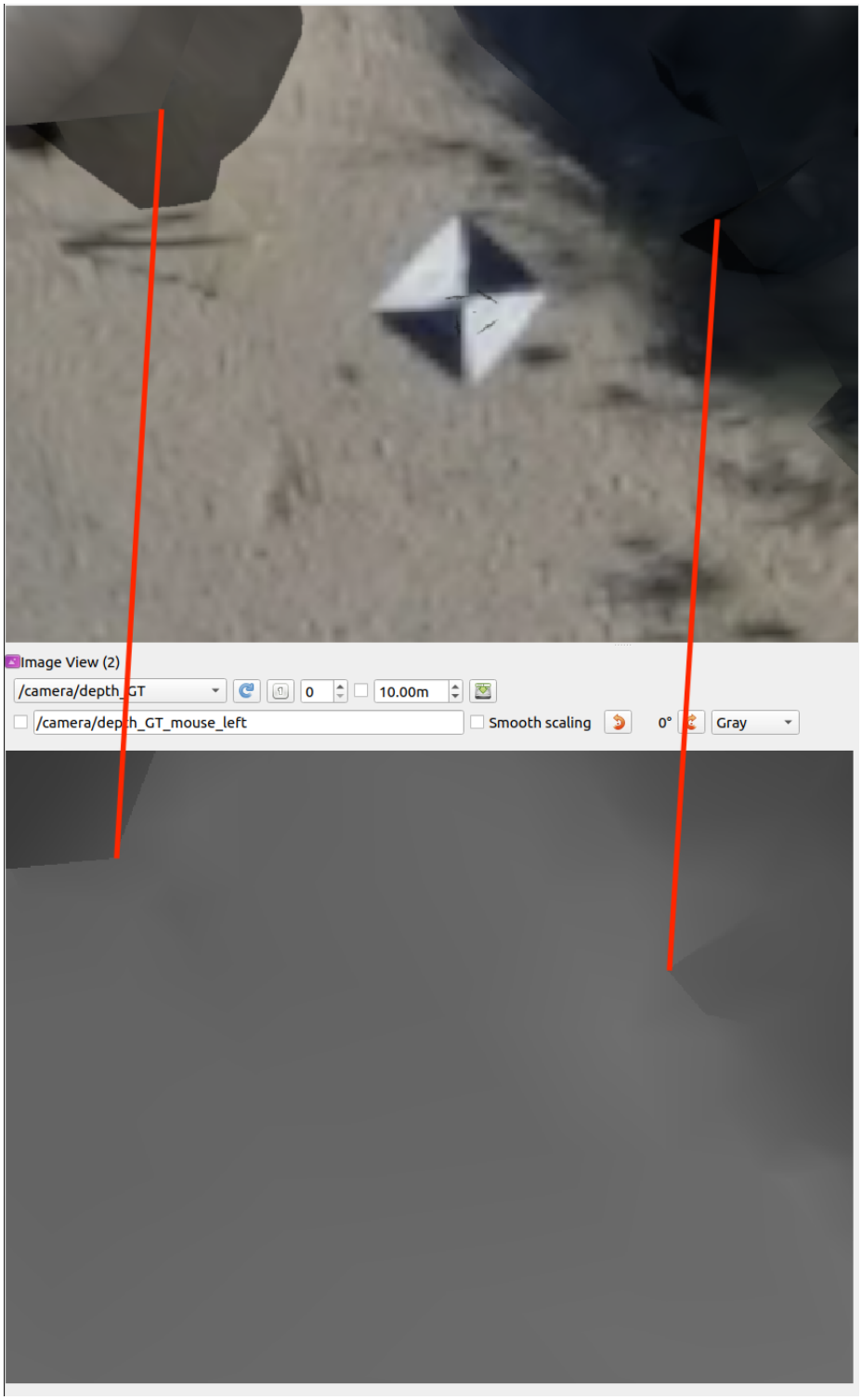


Figure 13.1: Depth camera misalignment. Top: reference camera image of the stereo pair, Bottom: depth camera image