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

Autumn Term 2024

Supervised by:
Roland Brockers
Robert Hewitt
Marco Hutter

Author:
Lasse Fierz

Contents

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

Methodology

As the endeavor of this thesis was the merger and enhancement of various aspects of the LORNA project, the complexity lied rather in the understanding of the existing work and the interfaces thereof 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[1] allows us to fly independent missions at cruise altitude 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 also at such high altitudes. This information can be used by LSD in order to detect landing sites during mission.
At low altitudes SFM works as well but surrounded with obstacles, the need for lateral motion poses significant risk. This is because a local state estimator is by nature prone to accumulate an estimation error and the same holds for the structure from motion approach. Our terrain knowledge can thus become void.
To overcome this issue, a range sensor can be used. As LiDAR might come to mind. However, a LiDAR sensor produces rather sparse point clouds unless a newer sensor like for instance Ouster's OS1-128 sensor is used. In that case, there remains the weight issue with the sensor's 495g. As the drone is going to fly on 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 ($\tilde{1}0g$). Therefore, in this thesis I present a stereo camera range node implementation to remedy the shortcomings at low altitudes.

1.1 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 input to the stereo camera depth node are the two camera images and the drone's base link pose. Processing this information is different from for the existing SFM algorithm. Therefore, a new depth generation node was put in place.
As mentioned in **??**, 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 implementations of Heiko Hirschmüller's approach ([2]).
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 **??**. JPL has its own visual library called JPLV containing a stereo matching method which might be a more adequate choice for the future. This is because throughout various projects at JPL, issues with the StereoSGBM algorithm occurred. **Roland: Which?** Also, JPLV contains implementations for the often used CAHVORE camera model which OpenCV natively does not support. However, for the time of this work, especially using camera images without distortion, there was no specific reason to switch away from the working StereoSGBM implementation, therefore, the continuation thereof was pursued.
Furthermore, OpenCV's StereoBM variation was considered which is in general faster but less accurate than StereoSGBM. An analysis thereof is shown in **??**.

1.1.1 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 the need for lateral motion in itself is an undesirable necessity for a rotorcraft in unknown terrain.
In this setup the structure from motion approach is based on a key frame 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. [3] uses 6 key frames. Therefore, for a single point cloud it is necessary to move laterally 6 times in order to start perceiving depth. Following the depth error formula from a stereo disparity image (**??**) and assuming an altitude of 2.5 m above ground with a focal length of 256 pixels and a disparity error of 0.5 pixels, the necessary baseline in order to keep the depth error below a critical 5 cm is:

Software vs Hardware Depth Perception

Structure from Motion, being 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

As described in **??** the multi-resolution DEM used for depth aggregation in LSD is based on Optimal Mixture of Gaussian cells and thus converges over time.
According to **??** the landing sites chosen are likely on terrain with low uncertainty. Because of this landing sites are more likely to be detected and have in general a better quality when the terrain perceived has been viewed.

When a landing site has been selected we need to make sure that the landing site is actually correctly detected and of good quality. For this we would like to (re-)detect landing sites on rather converged terrain. Structure from Motion needs constant lateral motion for this. A stereo camera depth node simply hovers in place for any given amount of time.

Efficiency

All in all the stereo camera setup allows us to perceive a landing site at course altitude and after having traversed horizontally to that location, we can simply descend to a stereo camera friendly altitude for the verification. Compared to repeated lateral coverage of the area in question this is a huge increase in efficiency. Looking in depth at the stereo alternative of depth generation, we can first analyze the theoretical threshold of this system.

1.2 Ground Truth Depth

For evaluation purposes as well as proof of concept aspirations, a ground truth is required.
Additionally, GT was important, because at the time of this work the structure from motion node showed frequent signs of unreliability. See **??** for the evaluation thereof.

1.2.1 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[1], a ground truth point cloud could be created.
The depth camera creates the image using traced rays which fill a pixel with the center most range value that a ray detected.
As expected, the ground truth point clouds yielded very clean and easily interpretable LSD DEMs:

1.2.2 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 has to make sure 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 for the drone 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 diameter were not seen by the GT at a cruise altitude of 100 m.
For this, the following test was designed.
On a flat textured plane, 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 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.

[1] 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 **??**.

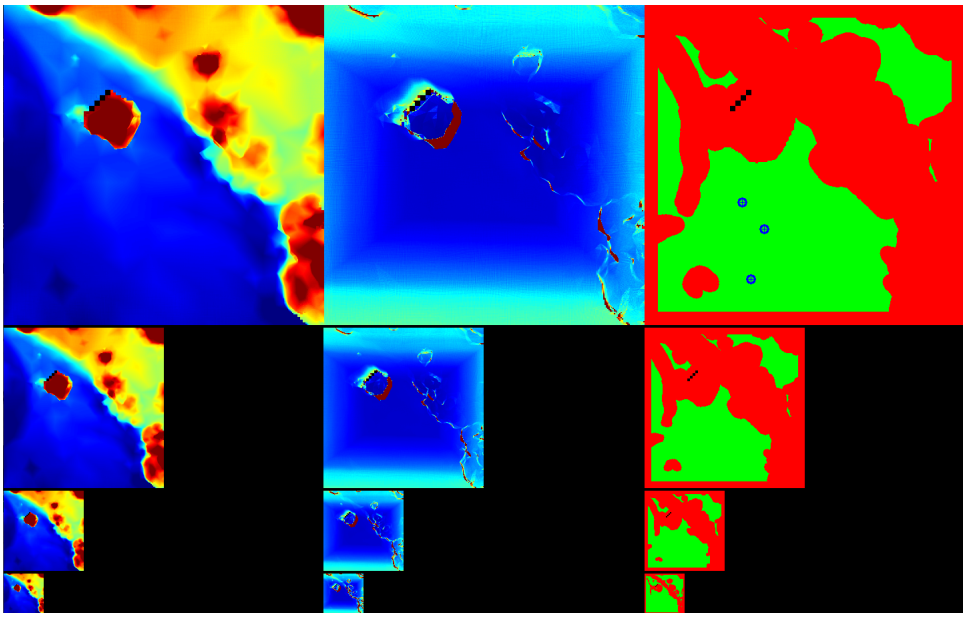

Figure 1.1: LSD debug image using a GT point cloud

Looking at fig. 1.4 and fig. 1.5, 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.
Therefore, the ground truth depth could be used for the testing of the autonomous landing pipeline without making a landing site verification step at low altitudes redundant.

1.3 Autonomous Landing Procedure

Having implemented a stereo camera as a low altitude alternative to SFM, and after 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 in order 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 which 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.

1.3.1 LSD - Autonomy Interface

For the interface with the autonomy, both the autonomy and the landing site detection node had to be altered.

- Landing site detection output

  Prior to this work, LSD only published one single landing site's location. To give the autonomy more information to make adequate decisions and to avoid

Figure 1.2: LSD debug image simulation terrain reference

the stagnation on a single overconfident landing site, LSD was changed to also 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. The incoming candidates are ordered according to a novel landing site heuristic and further landing site handling concepts like re-detection and banishment were introduced.

1.3.2 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 **??**.
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.

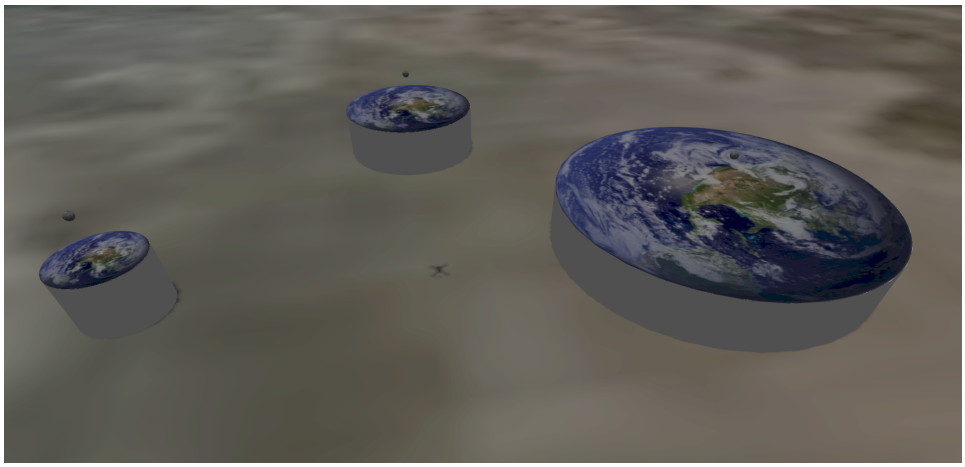

Figure 1.3: GT test setup: Rocks of 10 cm diameter where placed on platforms of different sizes to be detected by both the ground truth and SFM

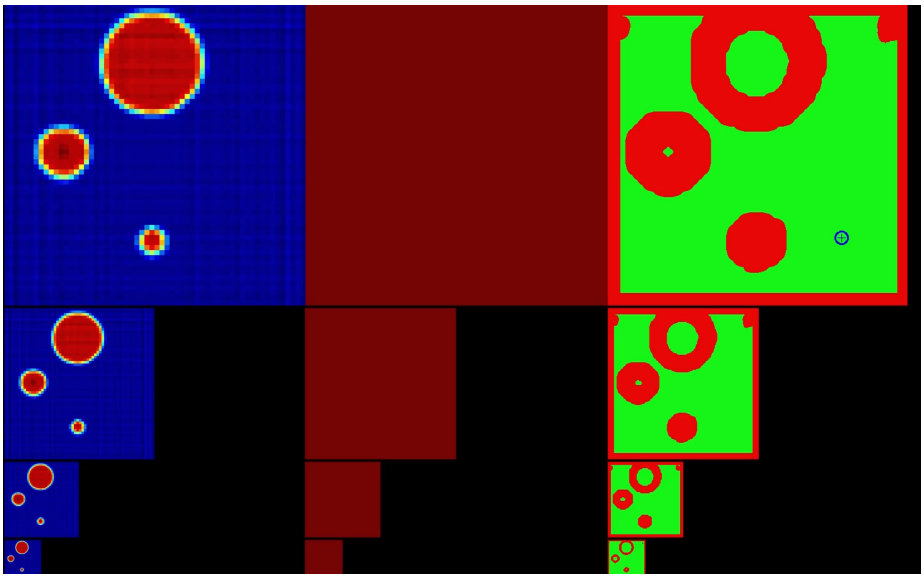

Figure 1.4: Ground truth result of the GT test

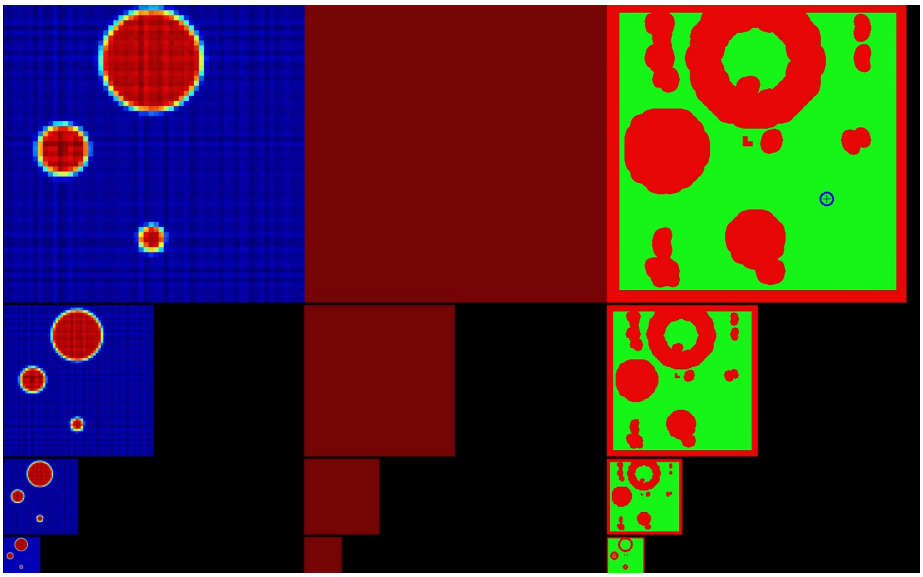

Figure 1.5: SFM result of the GT test

Chapter 2

Autonomous Landing Procedure

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

2.1 LSD - Autonomy Interface

In order to make high quality landing decisions in the autonomy framework one needs high quality information sent to the system by the landing site detection algorithm.

2.1.1 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 **??** the landing site detection algorithms segments hazards based on roughness and slope. Thereafter, it considers the size of a landing site as well as the uncertainty associated with a certain selected location.
Simply outputting the location of a landing site is therefore a waste of information when so many characteristics are at hand 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 averaged uncertainty across the area around a given landing site.

- Roughness

  The roughness value the exact value already used for the hazard segmentation step in the landing site detection. See **??** 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 in order 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. Calculating the physical value, 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 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 obstacles heights to avoid in the vicinity of a certain landing site. More on this can be read in section 2.3.1.

The final landing site detection output is a custom landing site ROS message containing the above-mentioned characteristics of the detected spot. For more detailed explanations of these properties see **??** and **??**.
Note: The supplied landing site from LSD already underwent a filtering procedure utilizing the roughness, slope, size and uncertainty properties. The value in 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, in order for the autonomy to be able to weight 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. [4], the uncertainty values rise significantly in monotonous areas. Thus, a reweighing of the properties is required for optimal performance.

2.1.2 Landing Site Heuristic

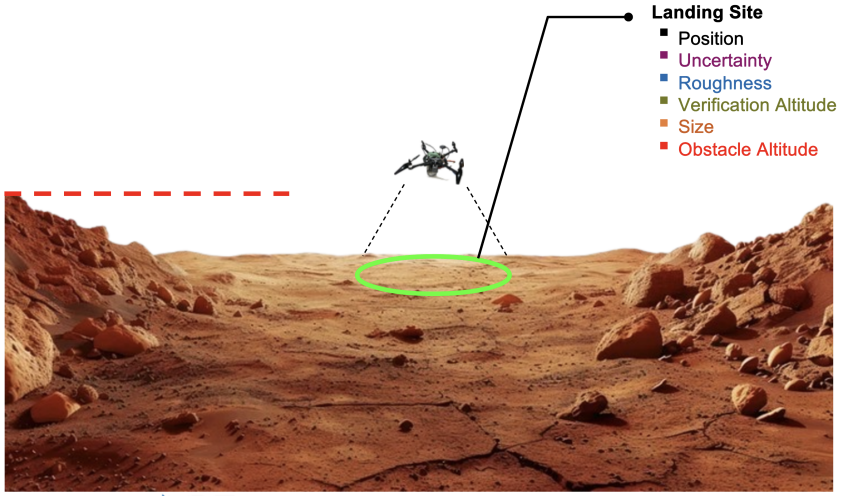

Figure 2.1: Schematic of new properties

The autonomy processes the in section 2.1.1 listed values in order to arrive at the in fig. 2.1 shown properties:

- Current Distance to Drone L_{dist}

  Well likely the single most important characteristic of a landing site[2]. Each iteration the current distance to the drone's position is calculated 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 being 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 the size comes from the landing site detection directly. 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.

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

  Also, as can be seen in eq. (2.1), the size contribution enters the loss function with a negative sign. This is due to the fact that compared to all other characteristics, the size defines a property that we would like 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 overlapping 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.

  - 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.

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

- 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 in fact a square loss function:

$$L_{LS} = w_{dist}L_{dist} + w_{rough}L_{rough} + w_{var}L_{var} + w_{size}L_{size} + w_{verAlt}L_{verAlt} \quad (2.1)$$

2.1.3 Landing Site Manager

Prior to this thesis, the landing site manager received artificially generated landing sites from a dummy landing site node. In this work the LSM was connected to the actual landing site detection output. See **??** for an introduction of the landing site manager.
With the new landing site properties introduced in section 2.1.2 the metric according to which a landing site is evaluated 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 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 mechanic of assessing the proximity of an incoming landing site to the already considered landing sites in the buffer. If an incoming landing site is close enough to a previously detected site, this is considered a re-detection. In that case, the new landing site is not entered into the buffer but instead, the previously detected landing site has its properties updated.
However, this property update is only performed, when the incoming landing site's was 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 given to the landing sites detected at a closer distance.
The benefit of the update of the characteristics lies in the refinement of the site's information and therefore the retention of a higher quality landing site candidate.
The schematic re-detection procedure is shown in fig. 2.2.
The new heuristics are indicated with the colored squares according to their introduction in fig. 2.1. 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 like the distance to the drone are derived and attached in the ROS connector.

Selection

In the already existing autonomy framework, the landing site handling was very similar. Landing sites were ordered, and upon entering the landing state, the best one is shared with the navigation nodes through the blackboard variable interface.

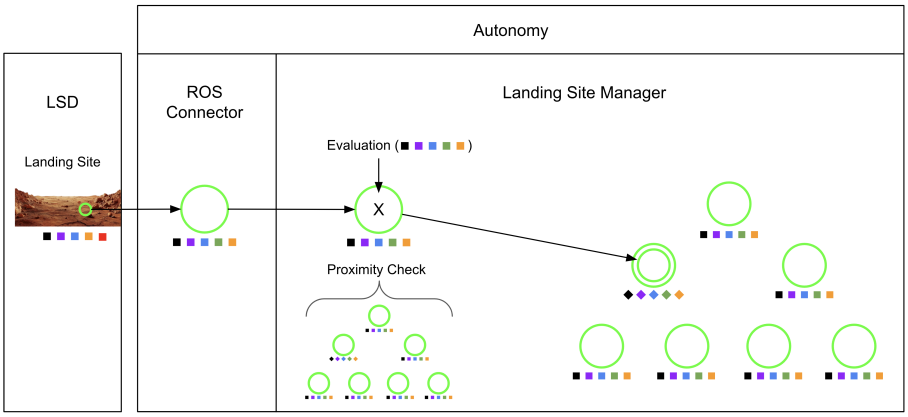

Figure 2.2: LS re-detection: When the proximity check is successful, the incoming landing site updates an existing landing site instead of entering the buffer itself.

The selection happened therefore only in the signaling of the landing site manager 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 to 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 2.1.3, 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 2.1.2. What's more, we can use this as a verification tool.
When pursuing a landing site which 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 **??**, 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 the removal of 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 and if a re-detection of an incoming landing site with a banned one is triggered, the incoming landing site is ignored.
The reason for this mechanism lies in the exclusion of the possibility of repeatedly

detecting and pursuing promising false candidate.
The final procedure is shown below in fig. 2.3. A received landing site is checked regarding its 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 discarded in the case of the ban list and otherwise used for re-detection. If it's not the case, the landing site is entered into the LS buffer, as shown in **??**.

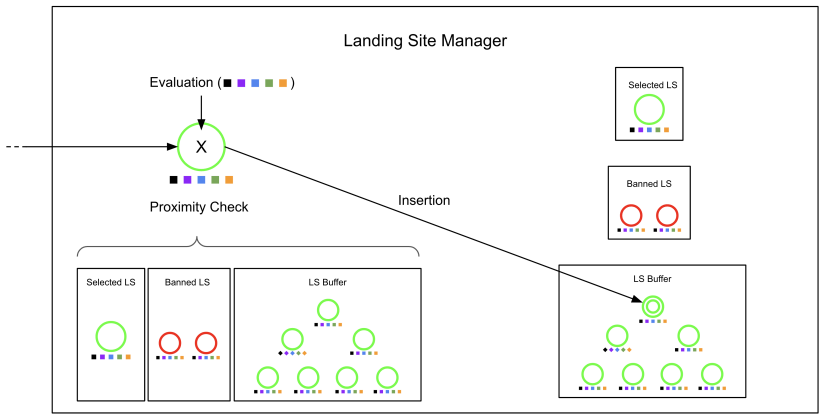

Figure 2.3: Complete LSM landing site handling

2.2 Conceptual Behavior

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

2.2.1 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 the 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 and detects point clouds which are given to LSD for the map aggregation and landing site segmentation. After reaching the switching threshold mentioned in **??**, 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 upon reaching the last one, the landing sequence is initiated.

- Landing

  The landing behavior uses the perceived landing sites in order to reactively and safely determine, validate and approach a landing site. When the landing decisions have concluded, the drone lands aka it descends until ground contact is established. The decision-making is explained in detail in section 2.2.2.

- Termination

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

2.2.2 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, within this safe behavior space, the effort was made to derive an efficient behavior implementation.
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, 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.

6. Perform the verification check introduced in section 2.1.3.

   (a) If verification is successful, initiate landing.

   (b) If verification failed, ban landing site and return to point 1.

2.3 Software Implementation

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

2.3.1 Action Definition

The autonomy framework already defined the core control flow nodes as well as some action nodes required for the landing behavior. These are introduced in **??**. Hereafter displayed is a list of additionally 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 2.3.2

- CheckLandingSiteAction

  Simple utility action which checks whether any landing sites have been found by querying the LSM's landing site buffer length.

- ChangeAltitudeLSAction

  This action was implemented with the purpose to allow us to descend to a certain fixed altitude above a chosen landing site. The creation of another action for this purpose is simply a utility which lets us avoid having to pass a function pointer in the action definition as the arguments passed in a behavior tree are always evaluated at the time of the creation.

- 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 **??** introduced loss function.

  This is done by using the landing site manager in order 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 as to not waste energy for the increased ascent / descent distances.

  This is where the aforementioned obstacle altitude from **??** 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 the detection of the landing site at the very

edge of the DEM shortly before significantly higher terrain starts. Therefore, one has to anticipate this case.

In practice a safe altitude buffer is implemented by assuming the worst case 45-degree incline of the terrain 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 which needs to 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

$$z_{\text{clear}} = z_{\text{obst.}} + z_{\text{buffer}} + d_{\text{drone-LS}} \tag{2.2}$$

$$z_{\text{clear}} = \min\left(z_{\text{clear}}, z_{\text{fail-safe}}\right) \tag{2.3}$$

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. 2.4.

- LandingSiteVerificationAction

  Once a landing site is chosen, we have to make sure 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 by SFM at 100 m altitude it will be overwritten when re-detected at 2.5 m above the ground.

  This is the mechanism exploited in order 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 processing incoming landing sites and if one is close enough to an existing one, it is considered a re-detection. In that case, the landing site is verified, and the landing action can be triggered.

  In case of verification failure, the landing site is not only removed but actively banned in order to prevent future false positives at high altitudes.

  Additionally, as previously mentioned, the verification hovering at low altitudes leads most probably to the detection of close by landing sites. This is arguably as important as the verification of a previous landing site as it yields a good candidate in close proximity. In practice, it turned out, it is equally likely to verify a landing site as it is to not verify one but detect a high quality landing site close by.

- 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 through fault of the landing site detection setup or due to 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 the 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 sub-mission is cancelled upon detecting a single landing site. In that case the usual landing procedure is continued.

- GetNextPatternCenterWPAction

  In case of failure when looking for additional landing sites, the adaptive procedure is to simply move to a new location and try anew.

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

  Note: This procedure is repeated a fixed number of times using the retry control node described in **??**. Optimally however, this would be performed as long as the battery state permits it.

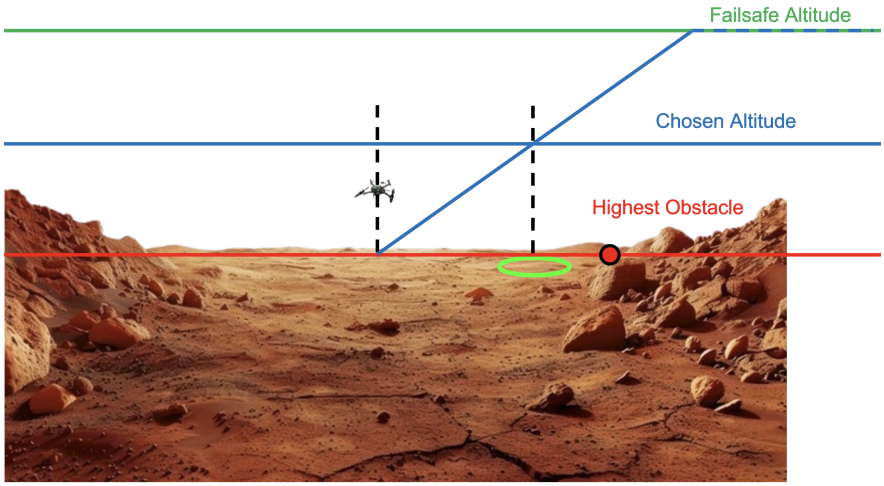

Figure 2.4: Schematic of the clearing altitude decision procedure

2.3.2 Behavior Tree Implementation

In the following, the behavior tree visualizations follow the scheme introduced in **??**.

High Level Behavior

To improve the readability, first, the high level landing behavior is shown in fig. 2.5: When the landing state is entered, the root node is initiated. First, the core landing behavior outlined in section 2.2.2 is attempted repeatedly using the retry decorator node. If it is successful, the fallbackRootNode returns success and the landing Action as well as the disarmAction are executed. If the node fails a certain number of times, it returns the failure state and the sequence node to land at the home position is initiated.

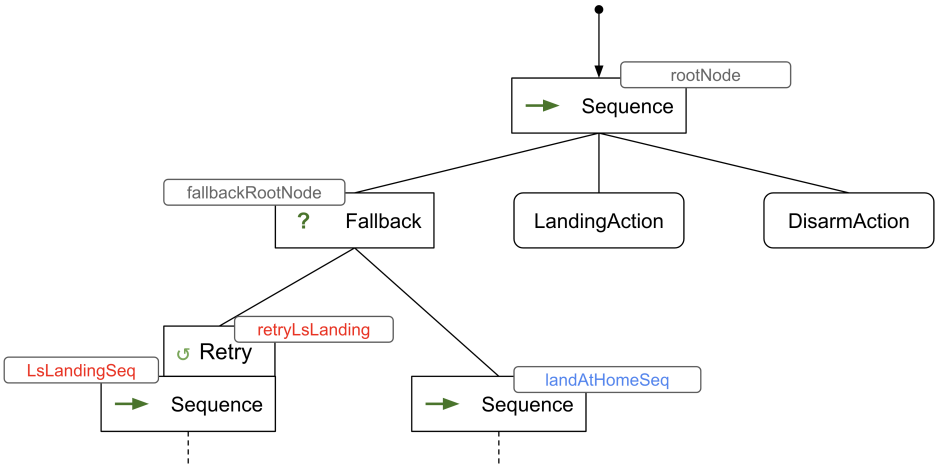

Figure 2.5: High level landing behavior represented as BT

Core Landing Behavior utilizing Landing Sites

The conceptual implementation of section 2.2 is shown in fig. 2.6. As for the behavior tree example from the system overview in **??**, 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 (aka verification behavior sequence) were placed onto two lines. Their order is to be read according to the line connections from the sequence node left to right.

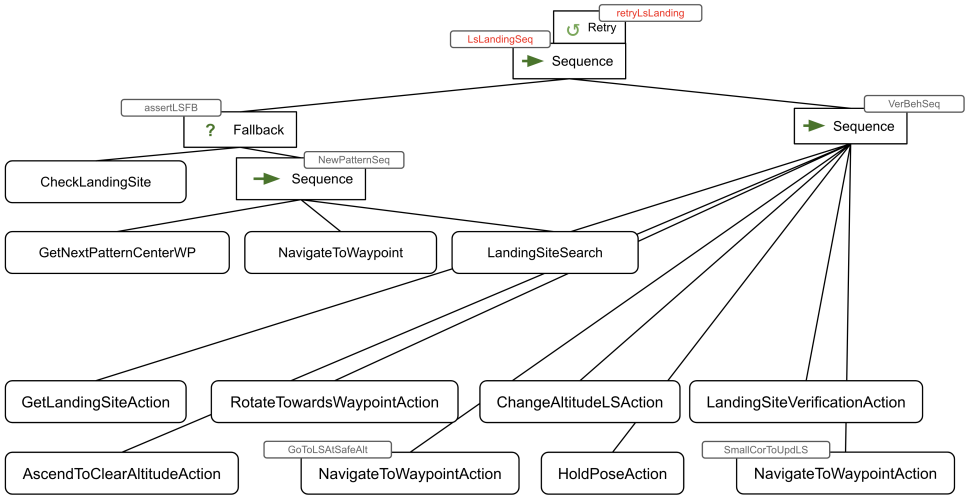

Figure 2.6: Landing procedure implementation as behavior tree

The left part summarized by the assertLSFB (aka 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 side is covered by numbers 1. a) and 1. b) of the conceptual behavior in section 2.2.

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. 2.7.

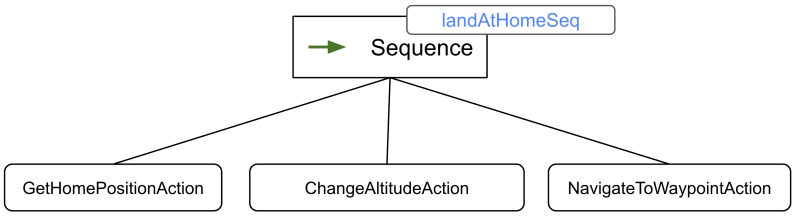

Figure 2.7: Behavior tree of the implementation when landing at the home position

The behavior when landing at the home position is very similar to the landing site based landing, however, 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.

2.3.3 Full Pipeline in Action

In the following, a simulated flight demonstrating this landing behavior is shown. fig. 2.8 Shows the setup of the demonstration flight. Arroyo Seco is the area outside the Jet Propulsion Laboratory in Pasadena, California. The drone takes off to 100 m altitude and traverses to the indicated location where the landing behavior is initiated.

Figure 2.8: Demonstration flight setup: Left: Arroyo Seco Gazebo map, Right: QGroundControl mission to be flown at 100 m altitude

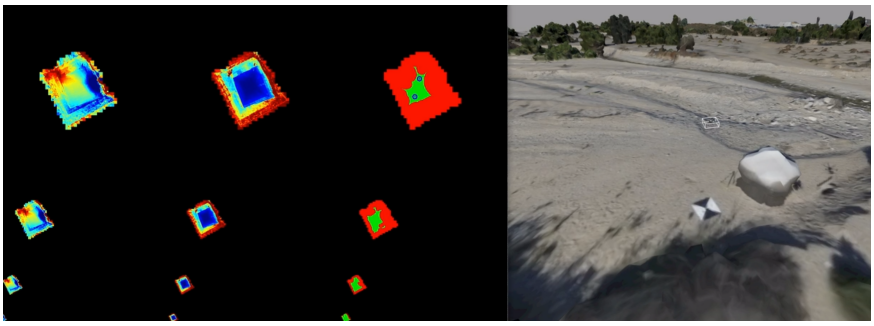

Figure 2.9: Takeoff of the drone: Left: LSD debug image, Right: Ascending drone in the simulation

Takeoff

fig. 2.9 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.

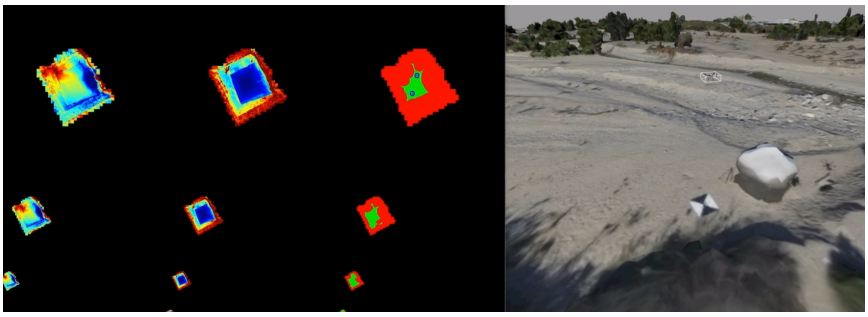

Figure 2.10: Later point of the drone's takeoff

fig. 2.10 shows the same takeoff at a later point in time. Note how the LSD debug image stayed the same despite the drone having risen to a higher altitude. This is because the stereo camera depth node has been switched off by the laser range finder readings and SFM took 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 in order for landing site to be detected on them. This is shown in fig. 2.11.
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. 2.6, a landing site is selected.

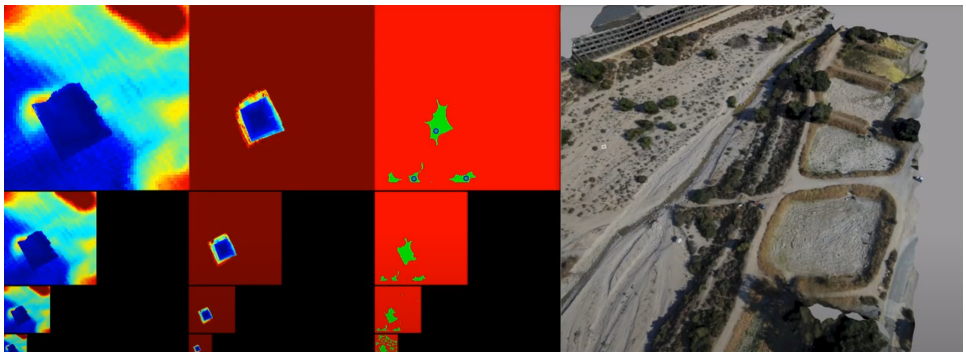

Figure 2.11: Initial traversal in the mission state upon reaching cruise altitude.

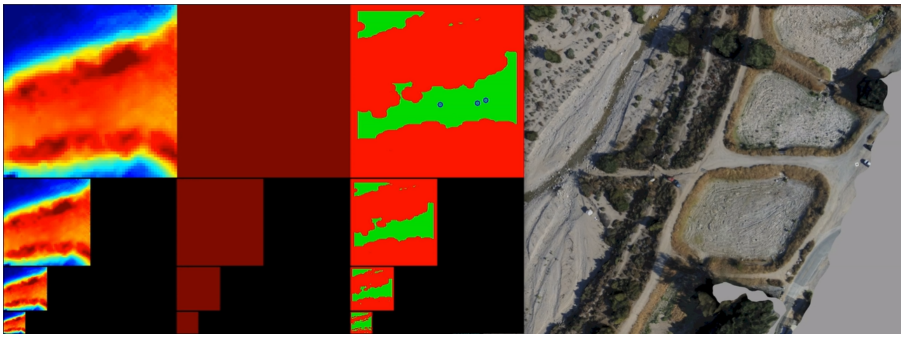

Figure 2.12: End of mission state

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. 2.14.
Upon having verified 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. 2.15.

Final Landing Action

Finally, the drone can land safely. It descends with a constant velocity which is decreased upon entering a minimum proximity to the landing altitude. fig. 2.16 shows the final location of the rotorcraft after the landing procedure.

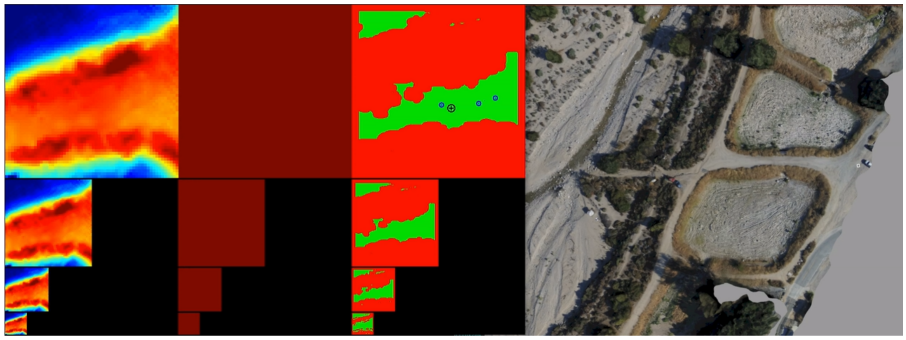

Figure 2.13: Start of landing sequence: the best landing site is selected and indicated in the LSD debug window with the black encircled cross.

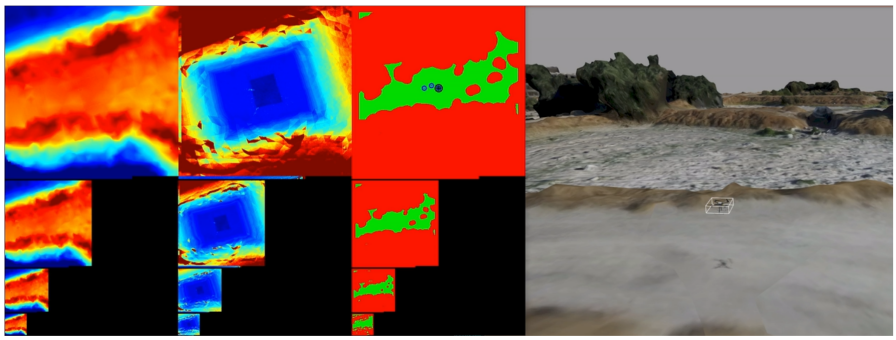

Figure 2.14: Rotorcraft hovering on top of landing site and trying to re-detect chosen landing site

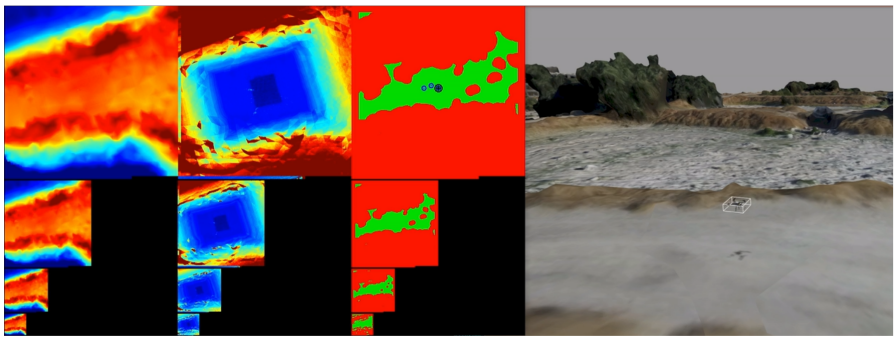

Figure 2.15: Rotorcraft adjusts position to refined landing site location

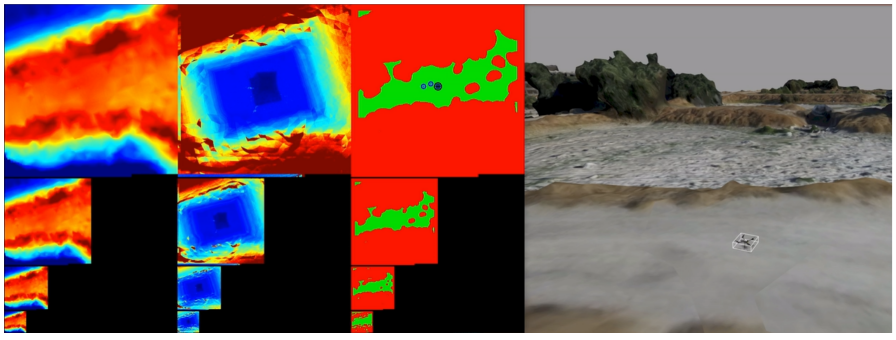

Figure 2.16: Final landing of the drone chosen by LSD

Bibliography

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

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

[3] M. Domnik, P. Proenca, J. Delaune, J. Thiem, and R. Brockers, "Dense 3D-Reconstruction from Monocular Image Sequences for Computationally Constrained UAS," 2021.

[4] P. F. Proenca, J. Delaune, and R. Brockers, "Optimizing Terrain Mapping and Landing Site Detection for Autonomous UAVs," 2022.