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

System Overview

fig. 1.1 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 in order to estimate the current state of the rotorcraft at any given time. Using map based localization (MBL) on HiRISE satellite maps allows for global localization. These poses are given to the flight controller and the output from that internal estimator is fed to the autonomy as well as the 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 the transmission of waypoint information and the setting of parameters.
The sensor images and their respective associated drone pose are given the landing site pipeline. In a first step, the 3D terrain is reconstructed using a structure from motion node. The reconstructed terrain is then given to the landing site detection algorithm(LSD) which first aggregates the information in a robot-centric multi resolution depth map and then segments valid landing sites on that. Detected landing sites are given to the autonomy to make the required adaptive landing decisions.

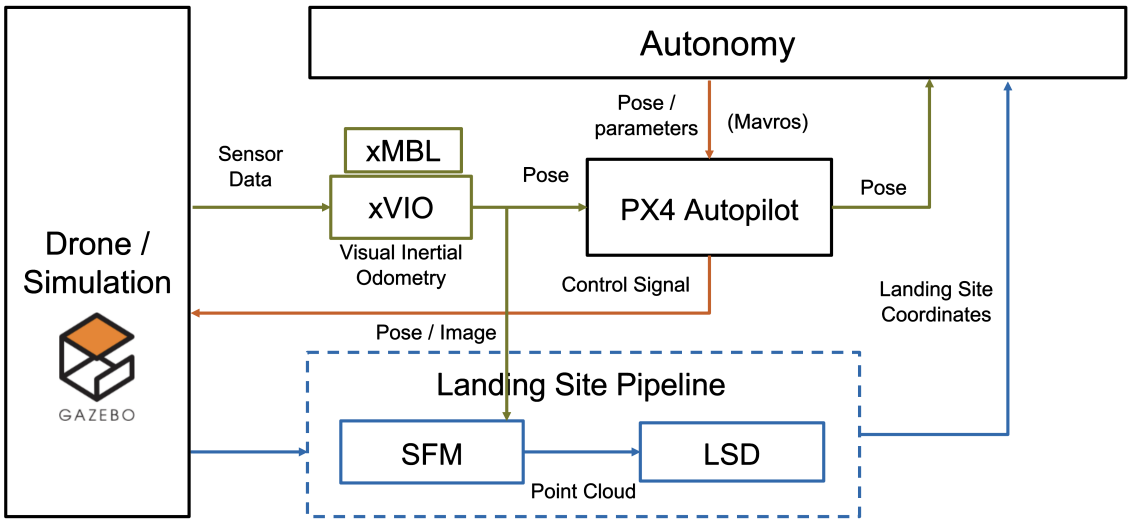

Figure 1.1: LORNA Project Setup

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. 1.2

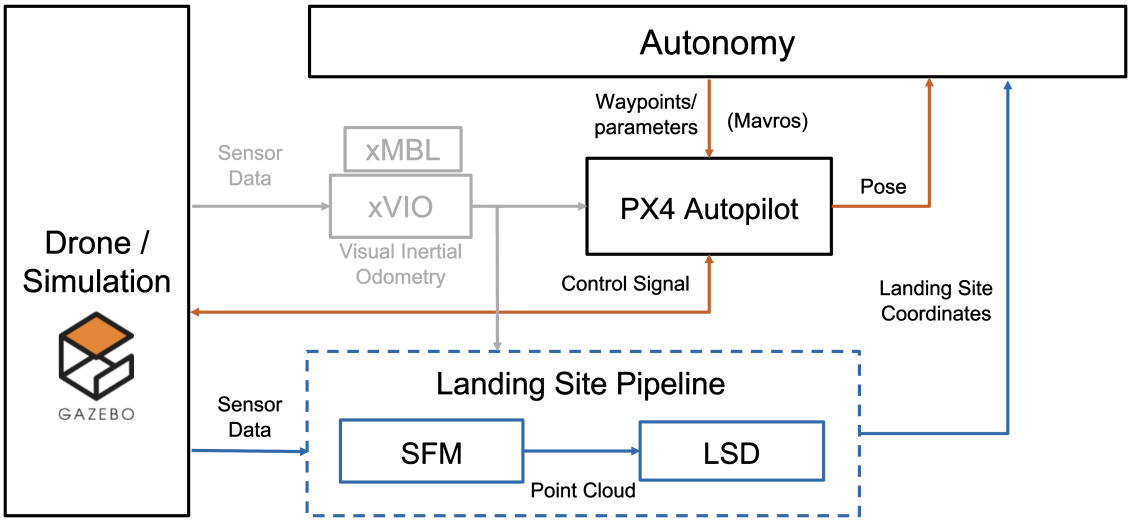

Figure 1.2: LORNA Project Setup with GT pose

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

1.1 Simulation

In this work I almost exclusively used a drone simulation. This allowed for a facilitated development with respect to both speed and safety. The choice of simulator, which is Gazebo Garden, was made prior to this work during the development of the autonomy.

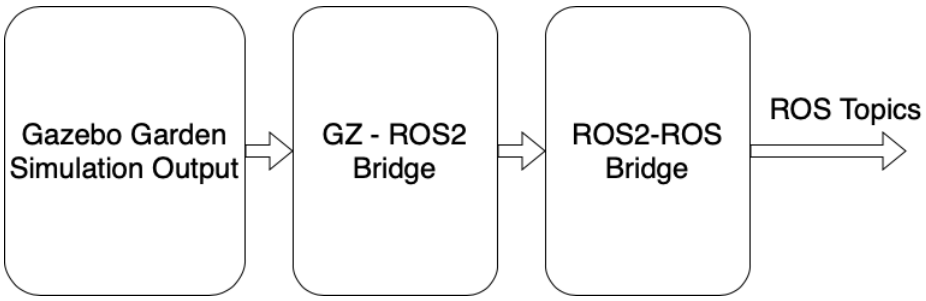

Figure 1.3: Gazebo ROS Bridge

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

1.2 Landing Site Detection Pipeline

The landing site detection pipeline consists of two nodes. A structure from motion node [1] which creates a point cloud using a key frame based stereo depth approach on monocular images and a landing site detector algorithm [2, 3] which aggregates the depth measurements into a rolling buffer based multi-resolution depth map and segments landing sites on the created DEM. These found landing sites are then supplied to the autonomy.

1.2.1 Structure From Motion (SFM)

The structure from motion approach retains images and their associated pose priors as key frames in a sliding window buffer. These key frames are used for a bundle adjustment pose refinement and subsequently, using a previously stored key frame 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 key frames 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 has been covered between the last key frame and the incoming frame, 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**

To advance the buffer over time, new information should be added. Therefore, new features should be detected on an incoming image. To match the incoming image with the existing key frames however, sufficient features must also be shared with the previously detected key frames.

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

- **Number of matched features between the two frames**
- **Overlap of the two image footprints**

  Using the simple pinhole model formula for an image footprint and the calculated baseline, the overlap of the areas are calculated.

- **Feature area ratio**

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

If these three metrics lie above a certain threshold, the key frames 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, leading to the loss of the oldest previous key frame.

Bundle Adjustment

Before doing 3D reconstruction, the poses are refined using a bundle adjustment algorithm. This is beneficial as the state estimator will always come with a certain error which drifts over time.

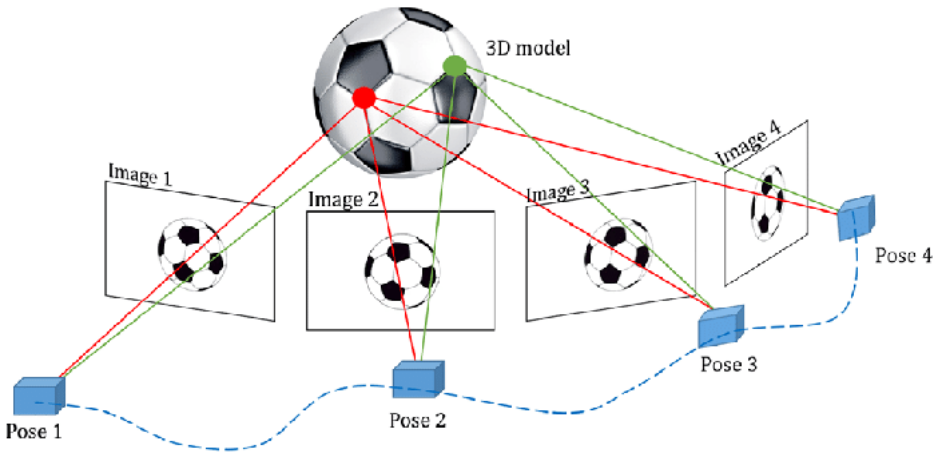

Figure 1.4: Bundle Adjustment Procedure as shown in Zhao et al. [4]

fig. 1.4 shows the change in reprojection, which the bundle adjustment optimization technique thrives to diminish. Generally, it does so by considering a number of key frames and optimizing their poses as well as the camera parameters used to acquire the images. In this case, only the pose refinement is considered for the key frames in question.

Stereo Depth

Having chosen adequate key frames and refined their poses, the crucial step of 3D reconstruction can be performed. For this, the key frames are checked to have an image overlap with the new image that is sufficient but not fully overlapping and to have a small enough baseline inclination so that assuming the same altitude above ground is reasonable. If the checks are successful, the key frame with the largest baseline to the new frame is chosen for the stereo procedure.
Using the two frames and the parallax distance between the detected features due to lateral motion between the images, the disparity values can be derived. With the camera's intrinsic parameters and the disparity values, a depth image of the terrain can be created. 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.

1.2.2 Landing Site Detection (LSD)

Depth Aggregation

The foundation of the landing site detection mechanism is a rolling buffer based multi resolution depth map as indicated in fig. 1.5. Each base layer cell is represented with 4 cells at a higher resolution layer.

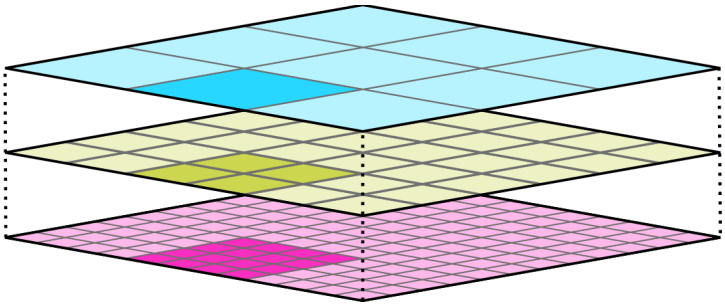

Figure 1.5: Multi resolution depth map as displayed in [2]

As introduced in Proenca et al. [3], 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 point cloud input iteration, a depth measurement and the associated stereo depth uncertainty is received. The measurements are placed in the respective cells based on the level of detail (LoD) of the perceived points. The LoD being $\frac{z}{f}$, where z is the depth measurement and f is the focal length of the camera. Specifically, an incoming measurement is placed in the highest resolution layer of all layers which are coarser than the incoming measurement. If the measurement has a larger pixel footprint than any of the layers, it is entered in the coarsest layer which is the top. In a subsequent step the measurements are pooled up to the top layer and down again in order to make the DEM more consistent and interpolate 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 second task that collectively pools all cells, achieves the same with fewer updates.
Similar to Kalman filters, the OMG cells' uncertainties improve over time as more measurements are entered. Because of this the DEM's terrain estimate converges over time.

Hazard Segmentation

On this created DEM, landing sites are then detected. 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 to store 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. 1.6:

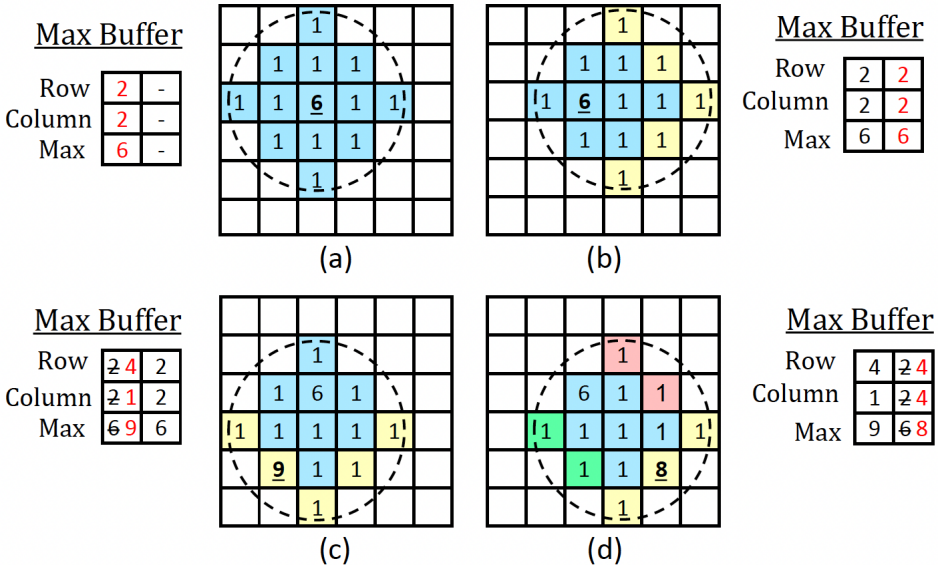

Figure 1.6: Optimized roughness assessment as shown in Proenca et al. [3]

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 area surrounding it, a plane is fitted using the linear least squares equation $Ax = b$ where A is a matrix containing in each row the x and y coordinates as well as a homogeneous coordinate for the plane's offset. 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.

Figure 1.7: Binary landing site map from the LSD debug image (Proenca et al. [3])

Landing Site Selection

After applying a distance transform on the landing site map, which yields the minimum pixel distance to a non-landing site, and performing non-maximum suppression on the landing site sizes, the landing sites with the largest valid circular areas are found.
Their positions are then 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. [3].
fig. 1.8 shows the landing map with the final shifted maximum area landing sites. These optimized landing site candidates are indicated with the blue crosses.

Figure 1.8: Binary landing site map after non-maximum suppression

Debug Images

Figure 1.9: Gazebo Simulation Reference

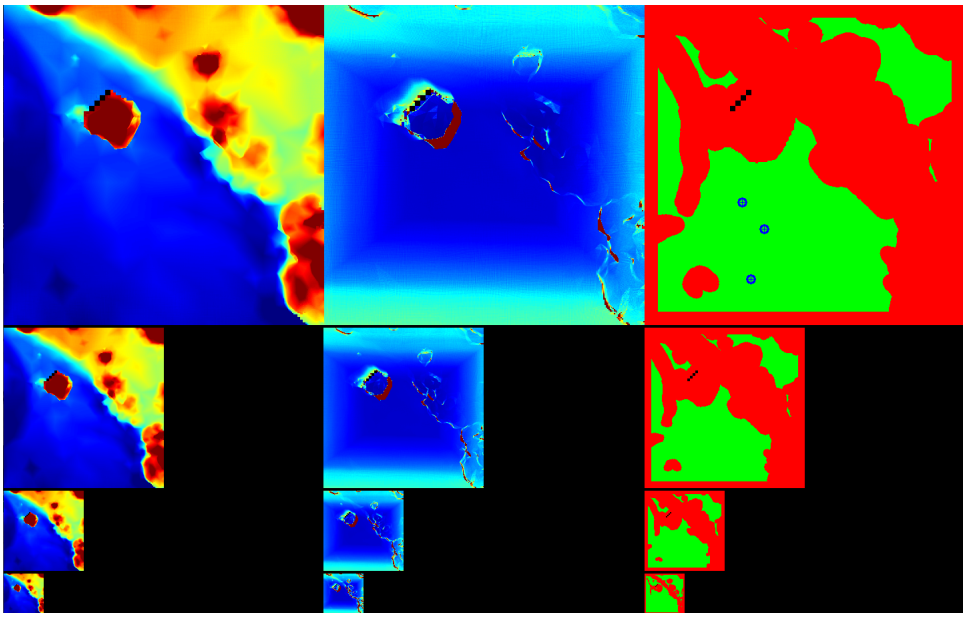

Figure 1.10: LSD Debug Image - Left: DEM, Middle: Uncertainties, Right: LS Map

The landing site detection debug image is a good comprehensive visualization of the landing site detection algorithm.
On the left, one can see the multi-resolution map displaying the same terrain area in different resolutions. Red pixels are closer, blue 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 above-mentioned binary landing site map. Green indicates valid landing sites, and the blue crosses indicate the chosen non-maximum suppressed and mean shifted landing sites.

1.3 Autonomy

The autonomy framework was developed within the LORNA project as a master's thesis ([5]). It is the overarching instance governing all the necessary behaviors during a mission and constituting the interface between all the different nodes of the LORNA pipeline. As shown in fig. 1.1, it is connected to the flight controller through the MAVROS wrapper of the MAVLink protocol and using a ROS connector, it also interacts with other LORNA nodes such as the landing site detector. The core of the autonomy however, is a state machine. A visualization of the setup is shown in fig. 1.11.

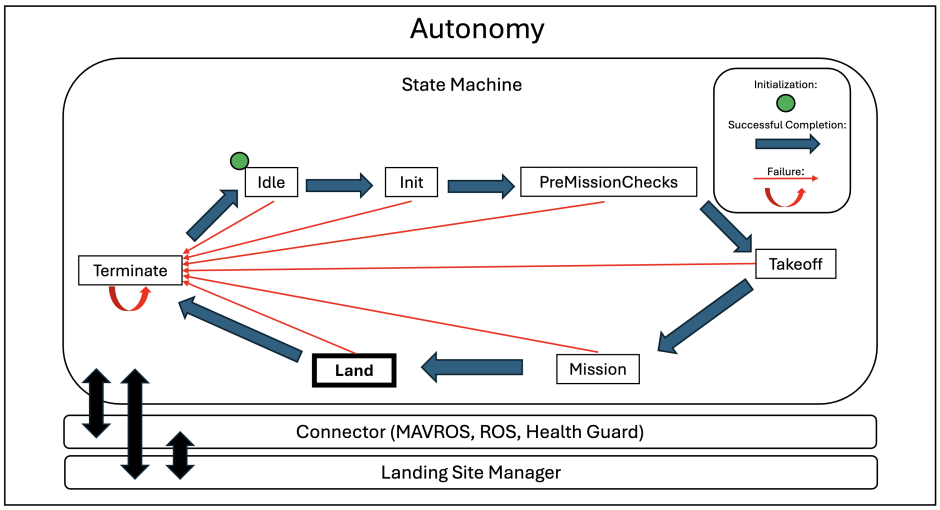

Figure 1.11: Simplified overview of the autonomy

The MAVROS connector establishes the off board mode in the PX4 autopilot in order to control the parameters and waypoints through the autonomy directly, effectively replacing the flight controller's decision-making behavior.
The ROS connector receives health events from the health guard, the drone's current pose from the flight controller's state estimator and, important for this work, the detected landing sites from the LSD algorithm.
In parallel to both the state machine and the connector, the autonomy contains another separate entity, the landing site manager. It processes the incoming landing sites from the LSD algorithm which were received by the ROS connector. Each landing site is ordered according to a heuristic to determine for each point in time what the best landing site is. In this thesis the landing site manager was expanded significantly regarding the heuristics considered and the general handling of detected landing sites.
Both the landing site manager and the connector are implemented as singleton nodes for easy 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 of instantiation of a landing site manager object instance. This

is achieved by removing the copy constructor of the class as well as the assignment operator and creating an instance acquisition function, which returns a pointer to the single static instantiation of the class.

1.3.1 Finite-State Machines

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

Working Principle

A finite-state machine is a mathematical model of computation which allows the hierarchical definition 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 order 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 the implementation of a rotorcraft mission.

The Finite-State Machine in the Autonomy Framework

In this setup, the autonomy framework itself acts like a finite state machine. At any given moment it is in a state and a tick is executed. During nominal conditions, this simply denotes a tick in the finite-state machine. In the unfortunate case of a health event however, the state machine handles the health event based on in what state the FMS currently is. 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. For more complicated procedures a behavior tree is used. This is the case for the takeoff state, the emergency state as well as the landing state. For this thesis the landing node indicated in bold in fig. 1.11 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.

1.3.2 Behavior Tree

A behavior tree is tool for systematic hierarchical plan execution. It enables the creation of complex tasks which are comprised of various simple tasks without having to worry about the implementation of such a small modular task. In contrast to a state machine, the core of a behavior is a task as opposed to a state. This is a matching choice of framework for the creation of complex adaptive flight behaviors. Especially since a rotorcraft mission can be easily split into small modular maneuvers such as ascension, lateral motion and rotation.

Working Principle

As the name suggests, a behavior tree can be visualized as a directed tree where the starting point of the tree is the root node.
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 which 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, the decorator nodes are auxiliary nodes which directly alter the workflow of action nodes in order 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 which have a parent node but no child nodes and decorator nodes have both one parent and one child node which they affect.
Similarly to a state machine, the workflow of a behavior tree starts with the root node which continuously sends a tick signal to its child node.

Control Flow Nodes

The most imported control flow nodes which were used in this thesis are the following:

- Fallback Node

  Attempts to execute the first child node and if successful, it returns a success state. Else 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 children nodes ran successfully. Otherwise, it returns false. Here an analogy would be the logical AND boolean operator.

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 fails or a timeout is reached.

- Retry Node

  Similar to the repeat node loop, but it repeats the node a number of times only until it succeeds, or a defined timeout is reached.

- Timeout Node

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

Action Nodes

There are a multitude of actions 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 in order to accomplish smooth landing.

- NavigateToWaypointAction

  Lateral movement to the given waypoint. Same proximity based slow-down mechanism as ChangeAltitudeAction and LandingAction.

- RotateTowardsWaypointAction

  Rotates the drone to face the given waypoint.

Bibliography

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

[2] 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.

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

[4] 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.

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