Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Aerospaziale

A.a 2021/2022

Study, Implementation and Test of a Navigation Architecture for an Autonomous Lunar Nano Drone

Relatore:
Prof. Paolo Maggiore

Candidato:
Giuseppe Bortolato

Correlatori:
Dott. Stefano Pescaglia
Prof. Piero Messidoro

Abstract

The Lunar Nano Drone (LuNaDrone) mission concept was conceived in light of the Artemis Accords signed by Italy and USA in 2020 and is associated with Italy's call to contribute to the development of enabling technologies for lunar surface exploration. The long term vision involves permanent or semi-permanent outposts on the moon, which require new approaches and solutions involving the study of lunar caverns and lava tubes. The goal of this mission is to explore possible entrances to lava tubes via an autonomous spacecraft, for which the Politecnico di Torino has been leading the feasibility and preliminary design studies since early 2020.

The focus of this thesis is on the design of the navigation system for LuNaDrone. First, the importance and characteristics of lunar lava tubes, their entrances (skylights), an overview of the discoveries to date and a description of the LuNaDrone mission, spacecraft and flight profile are briefly presented. After studying the requirements for the mission and the state of the art of navigation systems in a similar space application, the object of this thesis becomes the implementation of a visual navigation algorithm in MATLAB in the form of an Extended Kalman Filter. The algorithm makes use of a minimal set of sensors (Camera, IMU and Rangefinder) to operate as a fully self-contained system in difficult environments where no external assistance might be available and within the strict requirements of mass and volume of LuNaDrone. After an introduction to Kalman filters, there follows the description of the mathematical equations necessary for the implementation of the navigation algorithm. These include inertial propagation time update equations and camera and rangefinder measurement update equations involving a hybrid SLAM and MSCKF paradigm, as well as details of state management and vision processing routines. Finally, a prototype of the sensor package along with a mini computer, to record datasets, is developed and built and the data gathered is used for qualitative testing and validation of the algorithm implementation.

Acknowledgements

I would like to thank all the people that supported and guided me in the creation of this work: my thesis supervisor, Prof. Paolo Maggiore, without whom nothing in this thesis would have been possible, Dr. Stefano Pescaglia, for the help during this long and arduous work, and Prof. Piero Messidoro, for the invaluable and continued support of the LuNaDrone project.

I'd also like to thank my family, friends and relatives. They allowed me to reach this milestone with their patience, support, kindness and wisdom through all these years.

Contents

Chapter 1

Introduction

Interest in lunar exploration is rising again and several plans are being made to restart human exploration on the Moon, involving collaborations between different countries and space agencies. A Gateway station in cislunar orbit is planned to serve as an outpost for human exploration and other missions such as the ESA European Large Logistic Lander are in study. There is a future vision of permanent or semi-permanent outposts on the moon, which will bring many challenges requiring new approaches and solutions involving the study of lunar caverns and lava tubes. Much of the knowledge and technology could be one day used to extend human exploration further into the solar system and on other planets such as Mars.

1.1 Importance of lava tubes

The absence of an atmosphere and intrinsic magnetic field makes the lunar surface vulnerable to impacts of meteorites as well as energetic particles and radiations and gives place to extreme temperature variations during the day. Thus, places suitable for human settlement on the Moon could be underground volcanic tubes and rille systems [1]. A rille is a remnant of the volcanic tube, whose roof has capsized creating a valley, but at places the roofs of such tubes has not collapsed and remains intact, with a hollow interior.
While extremes at the lunar surface range from $-180\,^{\circ}\mathrm{C}$ to $100\,^{\circ}\mathrm{C}$ in its diurnal cycle, a lava tube interior is estimated to provide a constant $-20\,^{\circ}\mathrm{C}$ relatively benign temperature [1]. It offers the potential for a dust-free environment by sealing off entrances and the opportunity to use extremely lightweight construction materials, since radiation shielding mass would not be required and inflatable structures could take advantage of lighter fabrics and thin foil materials.

Lava tubes are common in basalt flows on the Earth, but whether they formed on the moon has been a subject of scientific discussion in the past[2, 3, 4]. It was suggested that sinuous rilles originated as lava-fed channels, some of which formed tube-like structures, based on their similarity to terrestrial lava channels and tubes. In 2011 a lunar spacecraft, Chandrayaan-1, detected with its Terrain Mapping Cam-

era a buried, uncollapsed and near horizontal lava tube in the vicinity of Rima Galilaei, estimating the roof thickness in the range of 45-90 meters [5]. More recently, the existence and entity of other such voids has been measured and inferred to be up to hundreds of meters in height and tens of kilometers in length by gravity mass deficits with GRAIL mission [6] and radar sounder measurements with SELENE mission [7]. Their stability has been studied with models, suggesting that under lunar gravity conditions voids up to 1 km across can be stable under only few meters-thick roofs [8]. In terms of scientific importance, observation of these voids and their entrances would reveeal the layered sequence of lava flows, piled one on top of another, which preserves a record of the composition and mineralogy of the domains in the Moon's mantle that melted over time.

The only known possible access to these voids are vertical shafts called skylights, which are the objective of the LuNaDrone mission.

1.2 Characteristics of lunar pits

The first lunar pit, a 60 m of diameter hole in Marius Hills region, was discovered in 2009 by SELENE cameras and identified as a possible opening to a lava tube under the surface [9]. In 2010, two more pits were identified by the same mission in mare regions [10]. Lunar Recoinassance Orbiter Camera NAC pictures at 1m resolution were used to confirm lateral openings to subsurface voids.
Following these discoveries, a search for other pits revealed 228 previously unknown pits with diameters ranging from 5 m to 900 m. The majority of these newly discovered pits are located in impact melt deposits, though five of the new pits are found in mare materials outside of impact melt deposits and two are located in non-impact-melt highland materials [11]. The number of mare pits rose after more of them were discovered in subsequent studies [12, 13, 14], bringing the total number to 16 (plus one more highland pit for a total of three).

1.2.1 Impact melt pits

Impact melt pits are frequently irregular in outline, as shown in figure 1.1, and their sizes range from a few meters to almost a kilometer across, but most of them are at the small end of that range, resulting in a median maximum diameter of 16 m [11]. Lava tube skylights on Earth and collapse pits on other planetary bodies are frequently arranged in chains or clusters, likely corresponding to the shape of the underlying void space [15, 16]. Pits within impact melt deposits are only rarely found in chains or clusters when not associated with fractures. Usually, impact melt pits do not occur in organized geometric patterns within their host craters. One of the few patterns that does occur is seen in at least six craters, most notably Copernicus, where pits form within long sinuous depressions [11].
King crater impact melt pits are distinctive in several regards: the melt pond that contains most of the pits is located outside the crater and has the highest concentration of non-fracture pits found in any melt pond to date and presents unusual

pits formed in positive relief features. In one such case, two pits formed adjacent to each other in a dome over the same subsurface void, separated by an 8 m wide, 5 m thick rock bridge [17, 18]. Slewed imaging of this feature revealed an overhang that extends at least 5 m south from the south end of the bridge. It is possible that many of the positive relief features host voids, with their surfaces occasionally perforated with pits.
The impact melt deposit on the floor of Tycho crater exhibits a high density of fractures, fracture pits, and pits and there are numerous wide linear collapses. Furthermore, this deposit contains both chains of pits in straight lines, which have little or no sagging in between them, and lone pits. Much like in the King melt pond, the fractures and linear collapses in Tycho crater generally correlate with the perimeters of kilometers-wide subtle topographic depressions [11].
Impact melt pits reside in areas with great changes in elevation and presence of boulders and craters and generally uneven terrain.

1.2.2 Mare and higland pits

All the known mare pits are generally circular to elliptical in shape and larger in diameter as shown in in figure 1.2.
Marius Hills and Mare Tranquillitatis pits were both confirmed to have overhangs with void space going back more than 12 m and 20 m respectively [17]. A void space going back at least 7 m has been identified in the Southwest Mare Fecunditatis pit and a one going back at least 20 m under the west wall of the Ingenii pit has been found, although there is no overhang to the east. The Lacus Mortis pit does not have overhangs on any side. The remaining mare pits either have not been imaged with appropriate geometry to identify an overhang or are too shallow for any void space to be identifiable with the resolution constraints of the NAC [11].
Two mare pits have been found near known volcanic structures. The Marius Hills pit is situated inside a rille, which was likely formed by high-discharge effusive flows from the Marius Hills volcanic complex, and the Mare Tranquillitatis pit is near the center of the Cauchy shield volcano, which is proposed to cover the eastern half of Mare Tranquillitatis [19]. Four of the mare pits occur within 10 km of large-scale tectonic features. The Lacus Mortis pit lies adjacent to, but not within, a graben, perhaps indicating a tectonic origin. The two Mare Fecunditatis pits and the Mare Tranquillitatis pit are each located approximately 10 km from a graben or wrinkle ridges, too far to immediately suggest a tectonic origin [11].
Wagner and Robinson [11] also suggest that, since the maria are generally older than 2 Ga [20], pits that formed as primary features have been erased, which is consistent with the number and size of pits discovered within the maria. The 100 m diameter Mare Tranquillitatis pit and the 58 m diameter Marius Hills pit both formed in maria older than 3.3 Ga, a surface age with a crater equilibrium diameter [21] of 290 m [22]. They propose that the mare pits could not have survived in their current state of preservation if they are primary features of mare emplacement. Therefore, the mare pits were almost certainly formed when the area above subsurface voids collapsed at some point after the maria were emplaced. These collapses may have been triggered by impacts [23, 24], by single or cumulative seismic events, or by a combination thereof. Because the mare pits did not form in active lava tubes, they

are not strictly speaking skylights, but rather collapse pits.
The mare pits are surrounded by relatively flat terrain compared to highlands and impact melt pits. Only three highland pits were identified that are not related to impact melt craters and they are circular in shape with large funnels, but unlike mare pits they reside in uneven and cratered terrain like impact melt pits.

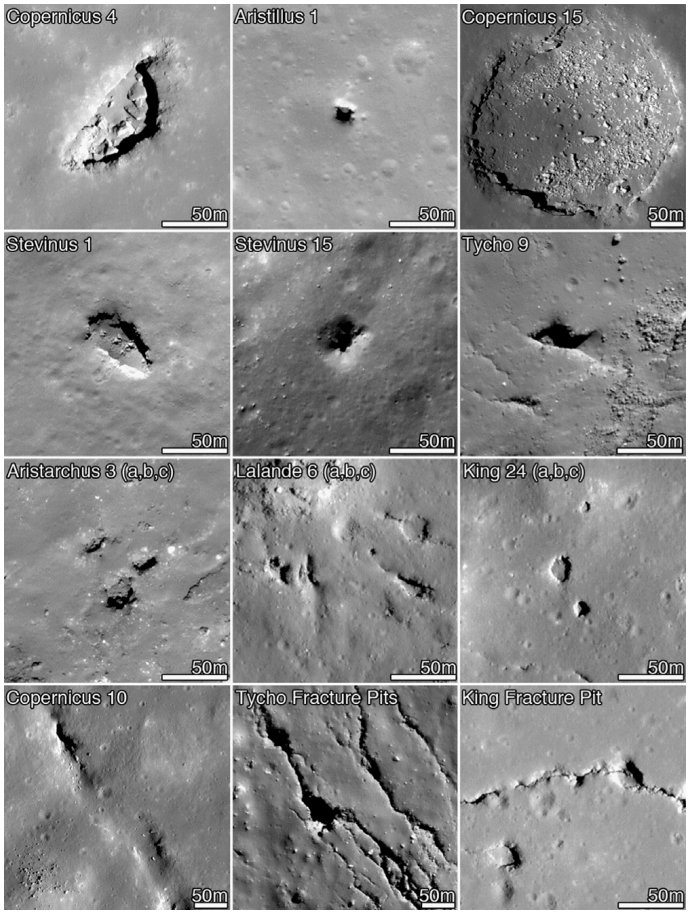


Figure 1.1: Examples of impact melt pits. Credits: [11]

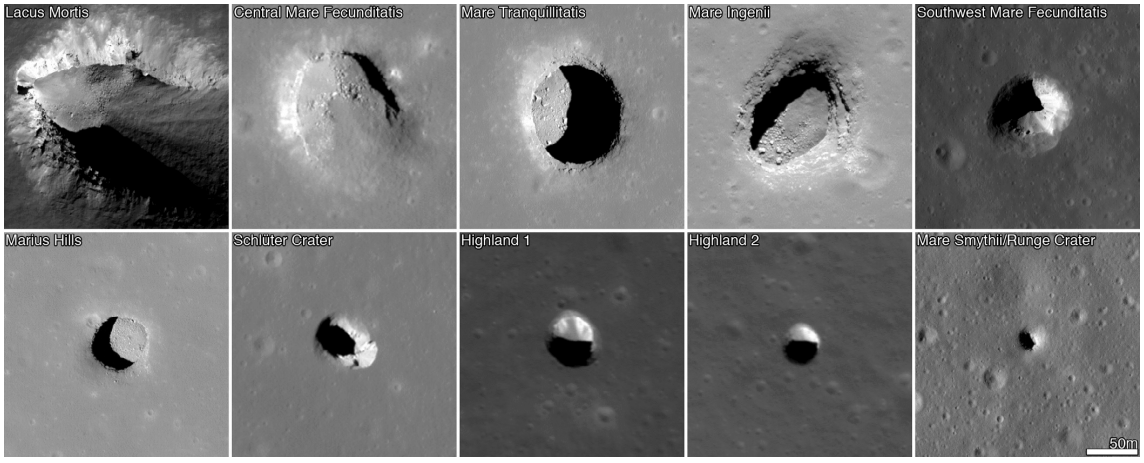


Figure 1.2: Mare and are and highlands pits. Credits: [9, 10, 11]

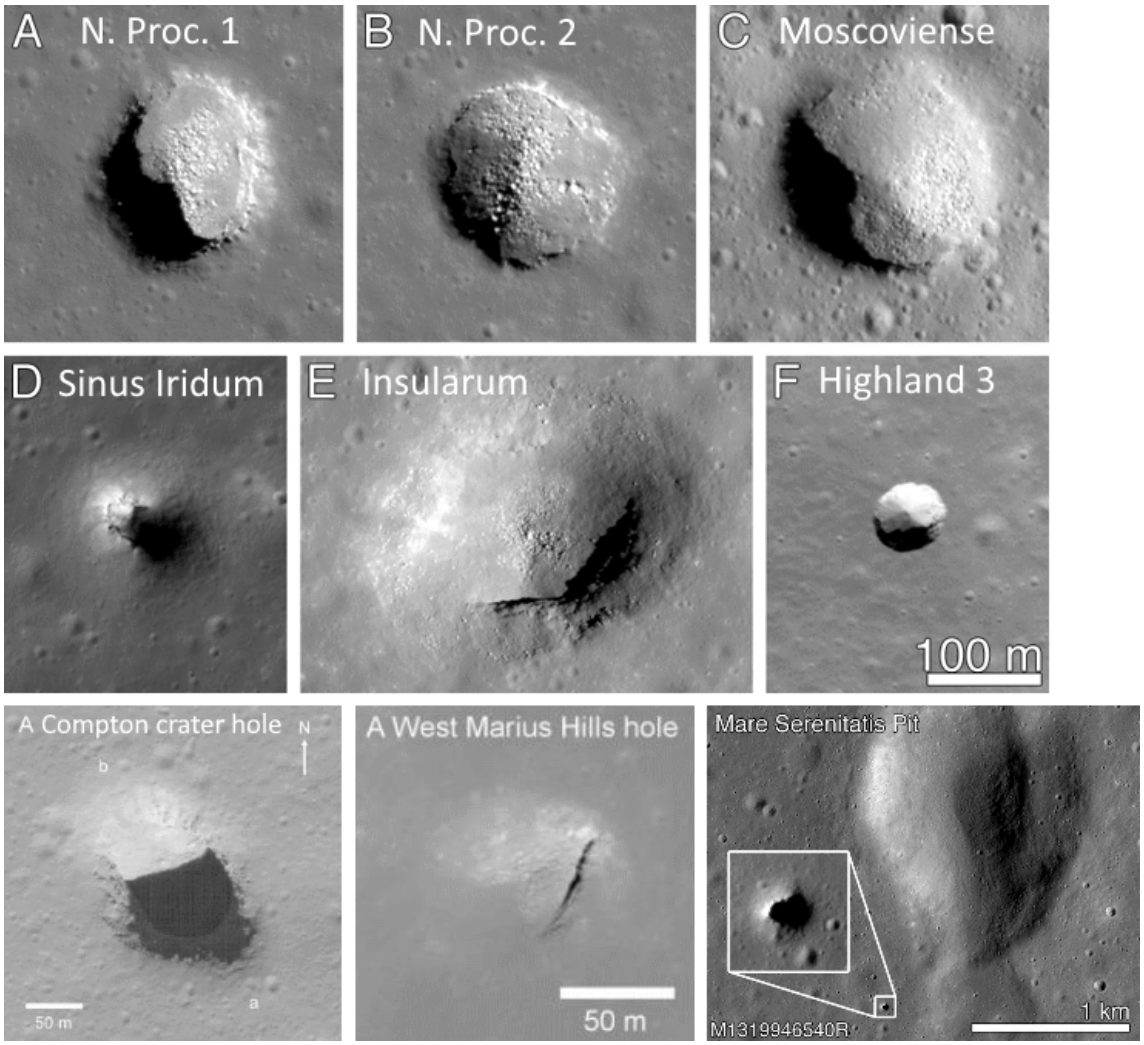


Figure 1.3: New mare and highlands pits. Credits: [12, 13, 14]

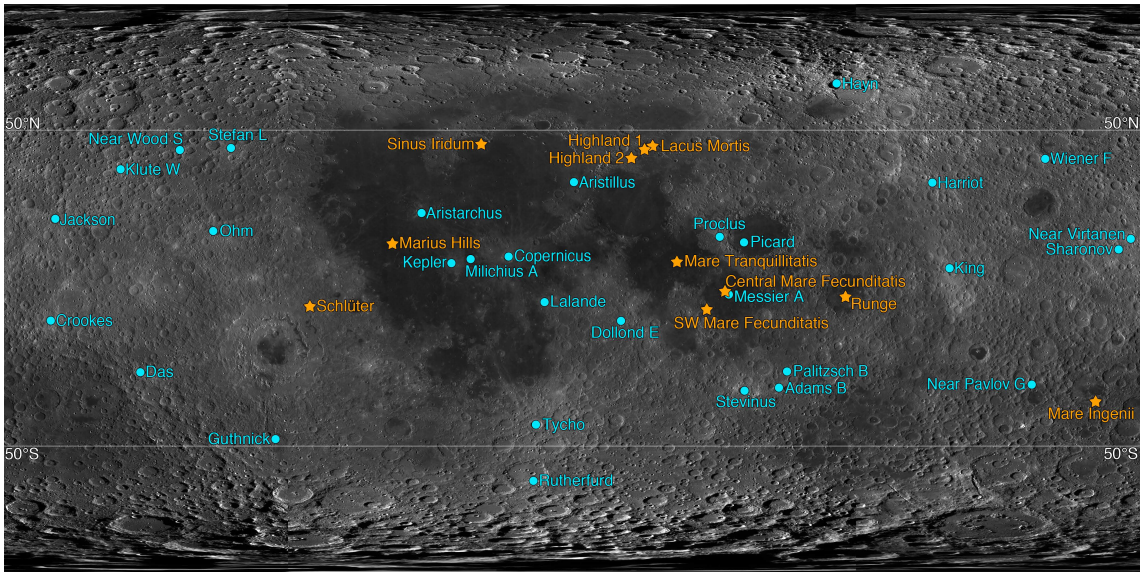


Figure 1.4: Locations of some of the pits. Credits: [11]

1.3 Mission site

A possible location for the mission must be selected in accordance with different needs, namely:

1. The scientific relevance of the site, primarily the possibility of the pit presenting an opening to a lava tube beneath the surface, also known as skylight.
2. The possibility of landing and delivering the spacecraft on the surface around the pit without excessive risks or technical challenges
3. Maximizing the probability of mission success by easing the task of entering and navigating to the pit

Given the first-of-its-kind quality of the LuNaDrone mission, the third criteria acquires a higher importance, since the goal for the first mission will be demonstrating the feasibility of the concept. Larger pits impose less stringent requirements on the precision of positional control of the spacecraft and regular round holes with sharp and short funnels and vertical walls, being in clear contrast with the rest of the surface, make it easier for the guidance system to identify the position and contour of the pit opening and reach the true bottom while avoiding collisions and the use of complex maneuvers.
A flat and regular terrain around the pit, free of dips and sharp edges, like a mare is preferred for an easier and safer landing and delivery of the LuNaDrone and a smaller latitude of the site is favored because it avoids complex and costly maneuvers to be reached. A location closer to the equator also offers a higher maximum elevation angle of the sun, allowing the interior of the pit to be illuminated, making visual navigation easier and allowing scientific observations in the visible region of the spectrum with reduced or no use of artificial light. A location on the near side of the moon could simplify the earth communication architecture of the mission and, in this regard, it is worth noting that there is a direct view from Earth from the bottom of Mare Tranquillitatis Pit. Positions of notable pits are in figure 1.4.
No pits have been definitively confirmed to be openings to extended lava tubes, but the best chance is represented by those with observed overhangs or cavities on their side and in proximity of features that indicate a volcanic origin. According to [25], the Marius Hills pit, being located within a lunar rille, represents the best candidate for a true skylight (collapsed lava tube ceiling), although the article is old and pre-date the discovery of new pits in [12, 13, 14].
[26] indicates Mare Tranquillitatis pit as the most compelling case for exploration from a geological standpoint, given the morphology of surrounding flood basalts, it's distance from the highlands and proximity to an internal mare boundary, the straightforward interpretation of its morphology and the likely exposed lava types being spectrally linked to basalt fragments in the Apollo 11 sample collection.

From the previous considerations, a reference site should be picked from the mare pits. In particular, three pits that present the most favorable features of size and shape (Mare Ingenii, Mare Tranquillitatis and Marius Hills) have confirmed overhangs of their side walls. The first of these three, Mare Ingenii pit, is located on the

far side of the moon and at a higher absolute latitude (although still at a modest 36°S), while the other two sites are at 8°N and 14°N, respectively. Mare Tranquillitatis and Marius Hills are also the most studied pits in the literature and the objectives of current mission proposals: they are among the oldest discovered and are in proximity to volcanic features that make them the best candidates for entries to lava tubes.
Marius Hills is the target of the five mission scenarios selected by ESA, one of which (DAEDALUS) [27] developed a hazard map of the area surrounding the pit and identified a possible landing ellipse, as seen in figure 1.5.
If Marius Hills pit is chosen as the reference site, being the smaller of the two and very similar to the other, a spacecraft capable of operating on Marius Hills would be able to do the same on Mare Tranquillitatis. Lacking better information specific to LuNaDrone mission, the landing ellipse from DAEDALUS [27] is taken as a reference indication of the distance to traverse between the landing site and the pit, about 300 m to 600 m.

Location	Latitude	Longitude	Central pit		Outer funnel (approx.)	
			Size [m]	Depth [m]	Size [m]	Depth [m]
Marius Hills	14.091	303.230	58x49	40	70x80	4-10
Mare Tranquillitatis	8.335	33.222	100x88	105	170x150	5-9

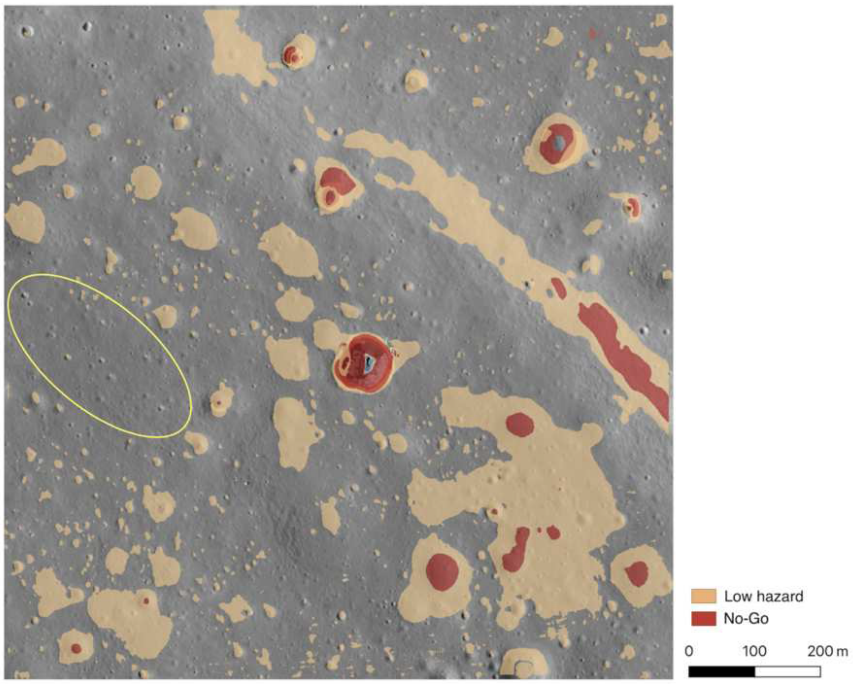


Figure 1.5: Marius Hills hazard map and landing ellipse from DAEDALUS study. Credits: [27]

Chapter 2

LuNaDrone mission

The LuNaDrone mission concept [28] was conceived in light of the Artemis Accords signed by Italy and USA in 2020. The work behind this project is associated with Italy's call to contribute to the development of enabling technologies for lunar surface exploration. In order to best meet these technological challenges, the Politecnico di Torino, which has been leading the feasibility and preliminary design studies since early 2020 [29, 30], has decided to collaborate with different SMEs operating in the most challenging fields of space industry. Indeed, as demonstrated in the past, universities, with the involvement of start-ups and SMEs, can effectively contribute through low-cost approaches to simpler missions in parallel to large institutional initiatives. The idea behind the LuNaDrone project is to follow the same approach adopted for CubeSats in terms of standardisation and modularity, i.e. to meet the need for standard, flexible and low-cost spacecraft for lunar exploration and for supporting future human infrastructure on the Moon.

The main mission objective is to explore the interior of a lunar pit to assess the existence of an opening to a lava tube and its accessibility for future robotic and human exploration missions. LuNaDrone is a compact spacecraft smaller than 12U and with a wet mass lower than 15 kg, delivered on the lunar surface by a host spacecraft (HSC). During those phases of the mission that precede the LuNaDrone operational phase (e.g., launch, orbital transfer and moon landing), LuNaDrone relies on the HSC to provide power, a communication link with Earth, shielding from micrometeoroids and a suitable thermal environment. The HSC is also responsible for deploying LuNaDrone at a minimum distance from the designated lunar pit. Once detached from the HSC, the drone is commanded via radio link. In particular, the communications architecture envisages the HSC to behave as a relay in communication between LuNaDrone and mission controllers. After the flight profile has been uploaded to the on-board computer, LuNaDrone performs an autonomous flight that takes it above the skylight. Then, the spacecraft starts its descent into the pit while collecting images of the bedrock lava layers in the pit wall. Once it has almost reached the bottom of the skylight, the drone hovers a few metres above the surface and begins scanning the surrounding environment with the aim of verifying the presence of an opening to a lava tube. Then, it egresses from the skylight and lands in the closest area assessed to be safe for landing and in line of sight to the

HSC, to which the drone transmits the mission data that will then be relayed to ground operators.

Although LuNaDrone was conceived for the exploration of lunar lava tubes, its inherent capabilities could lend themselves to other space exploration tasks. As NASA's helicopter Ingenuity did for Mars [31, 32], LuNaDrone could add an aerial dimension to the Moon exploration. The ability to quickly traverse difficult terrain that rovers can't roam and acquire close-ups that orbiters can't provide (e.g. craters, lunar pits and permanently shadowed regions) makes this type of spacecraft an ideal forward reconnaissance platform that would offer a broad range of scouting possibilities.

2.1 Challenges and related works

In [33, 34] several concepts for lunar pit exploration were considered, including: leaping into the pit with a robot, tethered rappel descent, robotic descent of a scree slope, descent from a Tyrolean line and precision landing on the bottom of the pit with subsequent deployment of an exploration rover. Of these solutions, the latter is the most similar to that of the LuNaDrone since in both cases a powered descent into the pit is required. However, LuNaDrone will not have to deploy any rover as it will perform scouting activities on its own. Furthermore, the drone will not land autonomously on the Moon surface from orbit, as it will be carried aboard an HSC, meaning that LuNaDrone's dimensions can be significantly smaller. Finally, since no mission requirements mandates a landing on the pit floor, which is classified as highly risky in [34], LuNaDrone will hover above the bottom of the pit, thus being able to scan the underground void without needing to land in it (the final landing will only take place once LuNaDrone has emerged from the skylight and is in a safer position outside the pit and in line of sight to the HSC).

Among the most studied approaches are tethered systems, such as the Moon Diver mission concept proposed in 2019 by NASA's JPL [35]. This approach relies on a 50 kg two-wheeled rover that would egress from the lander and, while anchored to it, would pay out its 300-metre tether as it traverses toward the pit, crosses the steep slopes of the pit funnel, and rappel down its vertical walls. Since LuNaDrone is not constrained by a tether, if compared to this solution it would have the significant advantage of imposing less stringent requirements on the landing ellipse. Indeed, the feasibility of the Moon Diver mission concept hinges on a critical space technology, i.e. pinpoint landing, that would enable the lander to land within less than 100 m from the edge of the MTP. Another critical aspect is related to the fact that lunar pits are generally characterised by a sloping funnel with loose regolith. Consequently, as reported in [36], any activity along this funnel would create mechanical instabilities in the regolith and underlying loose rock fragments, causing dust avalanches and rockfalls. In the case of LuNaDrone, by avoiding contact with the pit wall, this issue would be entirely prevented. On the other hand, by maintaining contact with the pit wall, the JPL rover would be able to perform otherwise impossible analyses such as collecting microscopic imagery for mineralogy and acquiring spectroscopic

measurements for elemental composition. Nevertheless, LuNaDrone would still be able to acquire context images of the pit wall to study the morphology of the exposed bedrock lava layers.

Following ESA's 2019 call for ideas to address the detection, mapping and exploration of lunar caves, five projects were selected [37]. Among them, two were further selected to take part in ESA CDF study, one concerning a spherical mapping unit, called DAEDALUS, to be deployed inside the lunar pit [27], and the other concerning a rover-mounted robotic crane that would be responsible for the deployment of DAEDALUS, or other exploring units [36]. In [36], the LuNaDrone project is also mentioned and in the comparison it is stated that only the crane would allow the controlled, slow descent needed to scientifically document the pit. Actually, LuNaDrone would be able to perform a controlled slow descent too, although propellant consumption would be higher the slower the descent, effectively placing a limit on the descent rate. Anyway, the LuNaDrone mission concept does not necessarily have to compete with systems such as RoboCrane, as it places itself in a different mission scenario. Since the mass of the rover-mounted crane would be more than 500 kg [36], this system will be compatible only with large lunar landers, such as the future European Large Logistics Lander. On the contrary, LuNaDrone's design is conceived to be compatible also with smaller landers, such as Astrobotic's Peregrine (whose first mission is planned for 2022 [38]) and other landers of NASA's CLPS program. This makes LuNaDrone suitable for a wider range of missions, including those planned in the near future. LuNaDrone could therefore be used to scout ahead and assess which lunar pits actually present an opening to a lava tube, thus being able to guide the more complex and expensive missions to sites that have proven to be worthy of interest.

2.2 Concept of operations

The LuNaDrone design is intended to make the spacecraft suitable for embarking on different missions aimed at distinct skylights (or even other lunar features), provided that the HSC deploys the drone at an acceptable distance from the mission target. If the lunar lander leverages terrain-relative navigation techniques, it may be possible to achieve a landing ellipse of less than 100 m [35, 39]. Nevertheless, with the goal of making LuNaDrone compatible with a broad spectrum of mission architectures and HSC designs, and at the same time not to oversize the spacecraft systems, a maximum of 1 km distance between the drone deployment site and the centre of the skylight has been considered. If the actual distance were to be significantly less than this maximum value, the extra propellant could be used to make additional flights inside the lunar pit, thus increasing the amount of scientific data that can be acquired. The depth the drone shall reach inside the pit can be up to 50 m as in the case of MHH or 120 m as in the case of MTP [11].

The LuNaDrone mission flight profile will be updated prior the drone's detachment from the HSC according to the landing site and its position relative to the mission target. The detachment will take place through a robotic deployment of the drone on

the lunar surface or through a controlled take-off directly from the HSC. LuNaDrone will fly on a fixed altitude trajectory, the height of which will be the result of a trade-off analysis taking into account the surface topography, the propellant consumption, and the performance and operational requirements of the GN&C system. Once above the skylight, the drone will starts a vertical descent during which images of the pit walls will be acquired. The drone will then hover above the pit floor at an appropriate height to avoid hazardous plume surface interactions and still manage to perform a detailed survey of the interior environment to identify a possible opening. Then, the drone will re-emerge from the lunar pit with a vertical ascent manoeuvre and fly up to the landing site. The latter will be chosen based on data provided by lunar satellite imagery and will be selected to be as close to the pit as safely possible and in line of sight to the HSC. After landing outside the lunar pit, LuNaDrone will have to survive in the lunar surface environment for a time sufficiently long to transmit to the HSC the scientific and telemetry data collected during the flight. This data will then be processed on ground and, if the status of the drone's on-board systems allows it, a second flight could be performed. Otherwise, the disposal phase will be initiated in order to ensure safety and not to impede any future missions that might head for that site of interest. If the amount of residual propellant allows it, the LuNaDrone may be commanded to perform one last flight that would take it away from the skylight. To maximise the range of this manoeuvre, it would be possible to command the LuNaDrone to fly until the propellant is completely drained. However, this would result in an uncontrolled crash of the drone on the lunar surface which, although occurring far from the skylight, would cause further concern. A safer alternative would involve the use of dedicated drain valves that would empty the pressurant gas and propellant tanks without the need for additional flights. Similar strategies would be adopted to make other safety-critical components (such as batteries) inert.

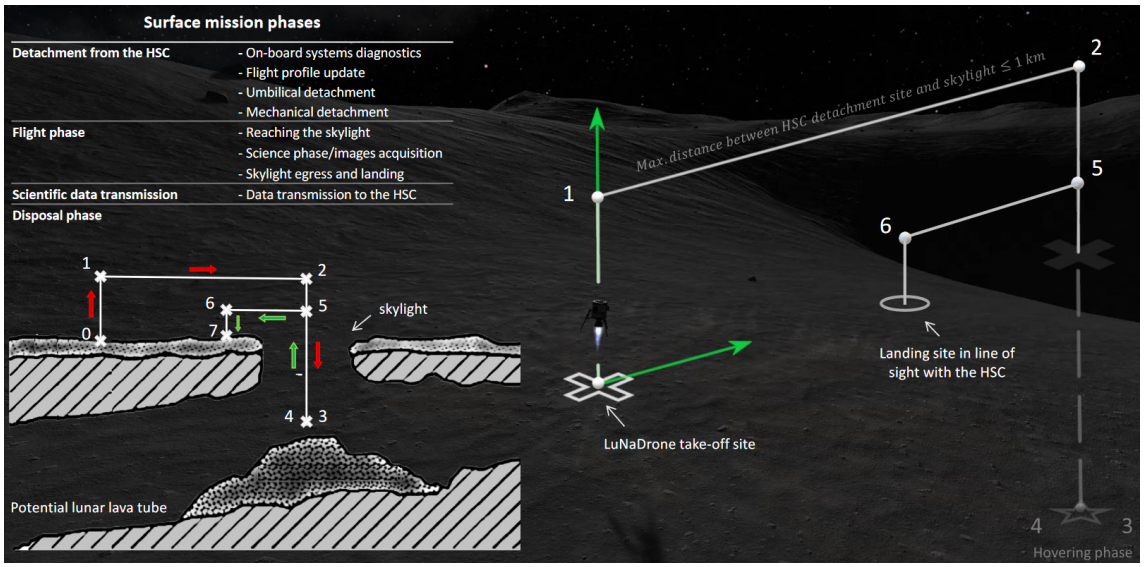


Figure 2.1: Flight profile of the LuNaDrone mission to lunar skylights

2.3 Propulsion system

The LuNaDrone propulsion system relies on a main hydrogen peroxide monopropellant thruster. This solution was the result of a trade-off analysis that saw 90% H_2O_2 prevail over potentially better contenders in terms of propulsive performance (such as Hydrazine), mainly due to its non-toxicity, which translates into easier and safer propellant handling in ground operations, and to its high volume specific impulse, which plays a significant role in stringent volume constraints applications such as that of the LuNaDrone. The main thruster has a nominal rating of 50 N which resulted to be the optimal value for the maximum continuous thrust. This value has been obtained by means of a careful analysis of the mission flight profile. This analysis was conducted on the basis of flight manoeuvre simulations that were developed in the MATLAB environment. The development of these scripts has also been essential for assessing the mission feasibility, defining the concept of operations and guiding the design choices about the LuNaDrone on-board systems.
The propulsion feeding system is pressure-regulated as it resulted to be the best design solution in terms of mass and footprint. The pressure regulator is the most critical fluidic element for which studies are underway to develop an optimal customized version of the valve. Thrust modulation is achieved through a bang-bang control technique, while the attitude of the spacecraft is controlled by small thrusters, powered by the same system as the one of the main thruster. Given the high propellant mass to spacecraft total mass ratio, the sloshing phenomenon needs to be properly managed. For this reason, a custom anti-sloshing system based on a piston separating the pressurant gas from the H_2O_2 has been designed.

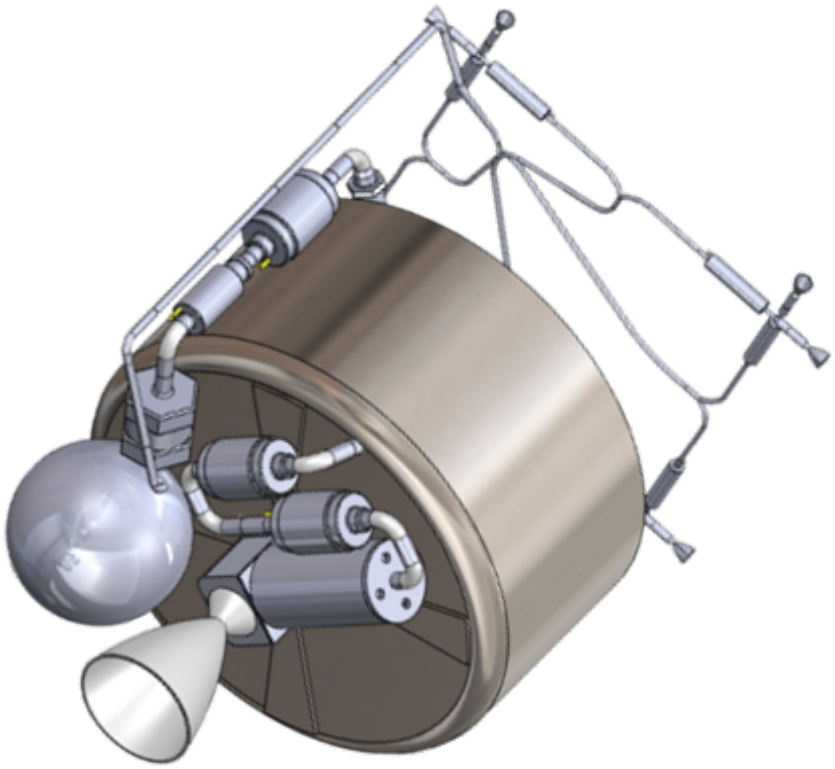


Figure 2.2: Representative CAD model of the LuNaDrone's propulsive system

Chapter 3

Navigation architecture

LuNaDrone requires its navigation system to be compact, light and able to offer precise estimation over the course of the flight, which is relatively short but includes critical phases like the descent into the pit. Sufficient accuracy is needed to avoid collisions and complete mission objectives in confined spaces as pits, which also make communication difficult or impossible, discouraging the use of external aids for navigation. Furthermore, reliance on external systems would increase the complexity of the mission and reduce the versatility of the LuNaDrone concept compared to fully self-contained architectures. A spacecraft of similar size and flight profile, although fundamentally different in the type of mission and mechanical architecture, is the Mars Helicopter *Ingenuity*, which was successfully flown with a vision based navigation system. This system is fully self contained and takes advantage of the recent technological advancements in miniaturization of integrated circuits and commercial electronic sensors to reduce mass and size to a minimum.

An array of approaches to vision-based navigation exist. The key concept is to recognize features in camera frames to recover information about the position and motion of the camera. Features can be pre-mapped landmarks, whose existence and position is known beforehand, or they can be generated on the fly. While the first approach can be very reliable, it cannot work when the area of operation is undefined or can't be mapped with sufficient accuracy. The second approach, generating features during navigation, is more generally known as SLAM (Simultaneous Localization And Mapping) and several algorithms exist which address the problem [40]. However, real time implementations of SLAM algorithms are challenging due to computational requirements in managing a large number of filter states that include the features themselves, each represented by at least three coordinates. Furthermore, as the number of features increase to improve the accuracy of the filter, the numerical stability decreases.

Before discussing LuNaDrone's navigation architecture, Ingenuity's solution is presented here as an insight into the state of the art of navigation systems of this type used for space exploration.

3.1 Mars Helicopter Ingenuity system

3.1.1 Hardware architecture

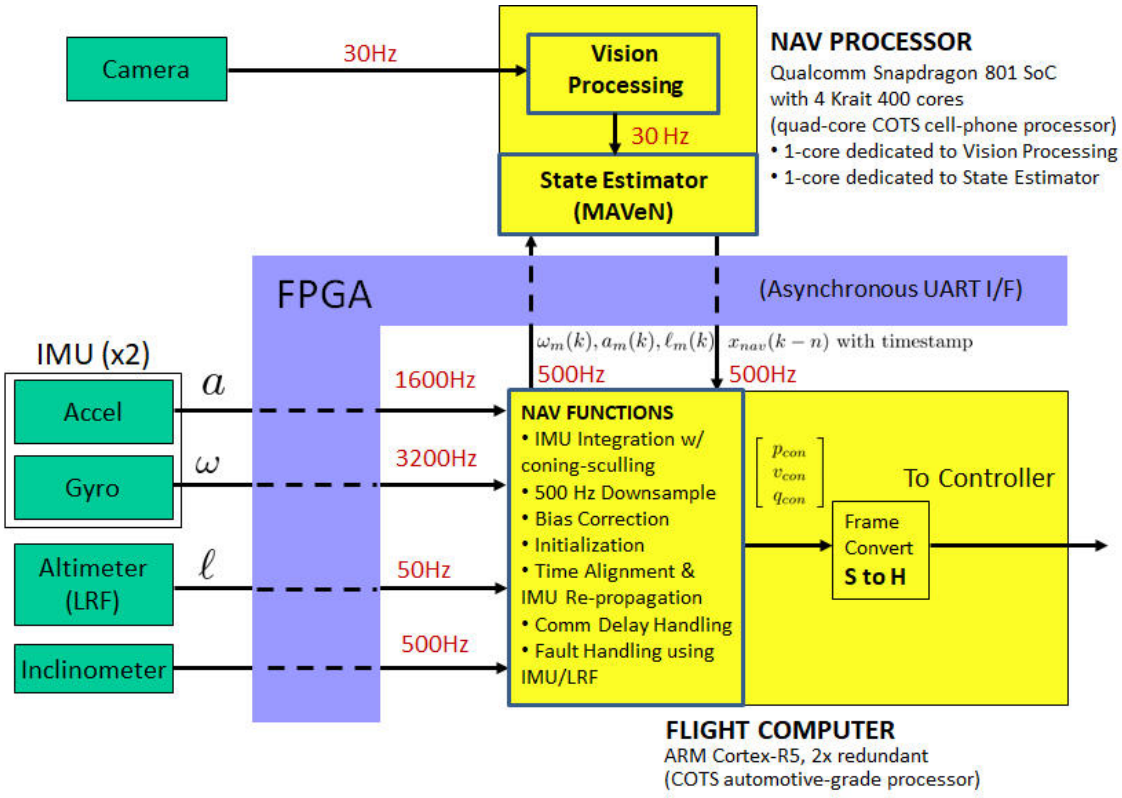


Figure 3.1: Ingenuity navigation architecture. Credits: [41]

Ingenuity employs a vision based system which makes use of the following sensors (an inclinometer is included for pre-flight calibration and initialization, but is not involved in the real-time navigation processing):

- A Bosch Sensortech BMI-160 inertial measurement unit, for measuring 3-axis accelerations and angular rates
- A Garmin Lidar-Lite-V3 laser rangefinder (LRF), for measuring distance to the ground
- A downward-looking 640x480 grayscale camera with an Omnivision OV7251 global-shutter sensor, providing images of the ground below the vehicle
- A muRata SCA100T-D02 inclinometer, for measuring roll and pitch attitude prior to flight

These are all commercial off-the-shelf (COTS) miniature sensors largely developed for the cell phone and lightweight drone markets. Camera images are read directly into the Nav Processor which also uses cell-phone technology and can directly ingest

video image sequences. The dedicated Nav Processor (Snapdragon 801) allows the CPU-intensive Vision Processing and State Estimation functions to be handled very efficiently using COTS cell phone technology. The Nav Processor serves to offload the FC and FPGA allowing them to more reliably handle important helicopter guidance and control functions. As seen in figure 3.1, the navigation processor and flight computer communicate asynchronously, so IMU propagation is intentionally made redundant between the two. The Flight Computer runs integration of IMU data at a fixed rate, propagating the state from the last navigation filter update, and receives new state and sensor bias updates from the Nav Processor when such are ready. This increases robustness in presence of dropouts and latency, allowing non-real-time operation of the navigation filter.
While using COTS components for most of the avionics poses additional risk, this choice was essential for meeting tight mission cost, mass, and power constraints. Moreover, all components underwent vibration, thermal, and radiation tests to make sure that risks are consistent with a NASA Class D technology demonstration [41].

Figure 3.2: Mars Helicopter Ingenuity

3.1.2 Algorithm

In the case of Ingenuity, Mars surface is observed predominantly from orbiting satellites which don't offer the necessary point of view or resolution for mapping landmarks to use in close ground proximity with visual navigation systems that rely on pre-mapped features. This and the challenges of pure real-time SLAM implementations motivated an alternative approach based on velocimetry called MAVeN. It avoids augmenting the state with feature vectors and uses a novel method of projecting image features onto a shape model of the ground surface, achieving full 6-DOF pose estimation with a relatively low-order 21-state filter. The inclusion of a rangefinder which directly measures altitude is critical to avoid ground collision and makes scale observable in camera frames, providing the ability to maintain a stable hover condition. In velocimetry, the vision system is used to characterize relative

motions of the vehicle from one image to the next, rather than to determine its absolute position. The navigation solution will drift with time but this is offset by the short (approximately 90 second), flights for the Mars Helicopter. For the Mars Helicopter application, a facet model of the terrain or digital elevation map (DEM) is not available, so that the shape model of the ground must be taken simply as a single facet. While this facet could be defined as a tangent plane approximation to local terrain as informed by satellite data, it is instead just assumed to be flat (i.e. normal to local gravity) for simplicity. This assumption is justifed by flying over terrain that is expected to have sustained slopes of only 1-3 degrees[41].

MAVeN is mechanized as an Extended Kalm Filter (EKF) which is updated by comparing a base image taken at time t_B with the current search image from camera. The state includes position vector p_S, velocity vector v_S and attitude quaternion q_S which form the search state, cloned position p_B vector and attitude quaternion q_B at time t_b which form the base state, and accelerometer b_a and gyroscope b_g biases. This process is briefly sketched here[41]

1. Identify the first image as a base image
2. Use the current estimate of base pose p_b, q_b to map features in the base image onto the planar surface model e.g., f_1, f_2, f_3 in figure 3.3. These feature positions will serve as pseudo-landmarks.
3. Identify the next image as a search image
4. Match search image features to the pseudo-landmarks mapped from most recent base image. Assume that there are m matches.
5. Combine the m pseudo-landmark matches with current geometry to form a measurement that is a function of both the current base and Search states

$$y_i = h_i\left(p_S, q_S, p_B, q_B\right) + v_i, \qquad i = 1, ..., m$$

   Perform Kalman filter measurement and time updates.
6. If the number of matched features drops below a threshold (or other relevant logic), declare the next image as a new base image and go to step 1. Otherwise declare the next image as a search image and go to step 3. In reality, the new base image in figure 3.3 is also used simultaneously as a search image associated with the previous base frame. This intentional overlap minimizes the drift incurred between base frames since it avoids a purely IMU-only period of integration.

In a motionless hover condition, MAVeN is capable of sitting on the same base image indefinitely which leads to very stable behavior. Algorithms based on the more computationally intensive EKF-SLAM are generally able to hover [42][43], while this property is usually not possible with non-SLAM approaches to velocimetry [41][44][45], at least those using few filter states like MAVeN. This capability follows directly from the novel approach of projecting image features onto a shape model

of the ground surface to use as pseudo-landmarks. During a test performing a 200 second hover, MAVeN accumulated only 0.6 m of position error[41].

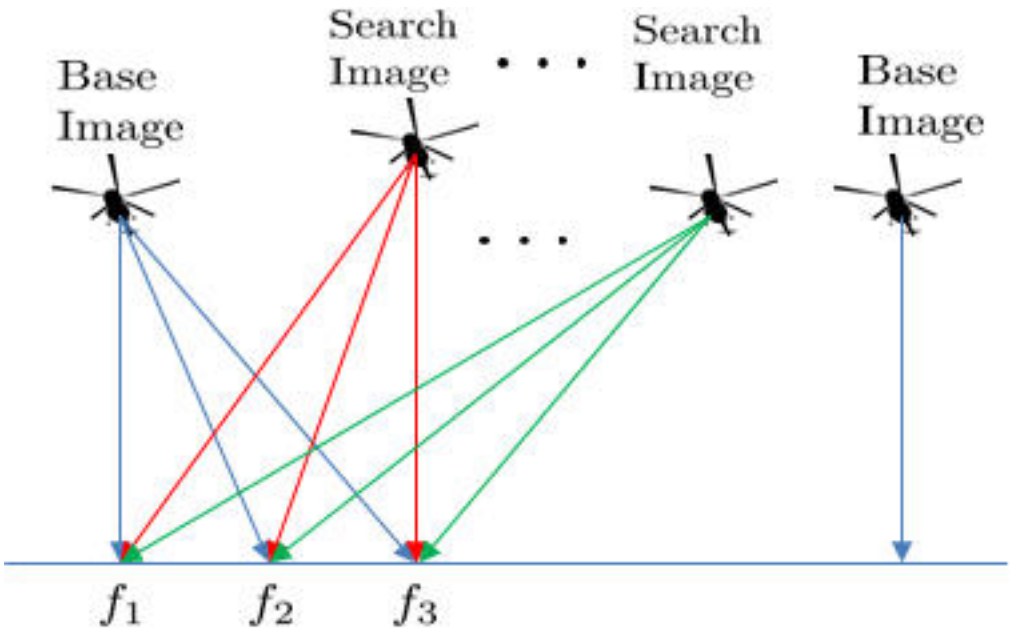


Figure 3.3: MAVeN base and Search frames. Credits: [41]

3.1.3 Vision processing

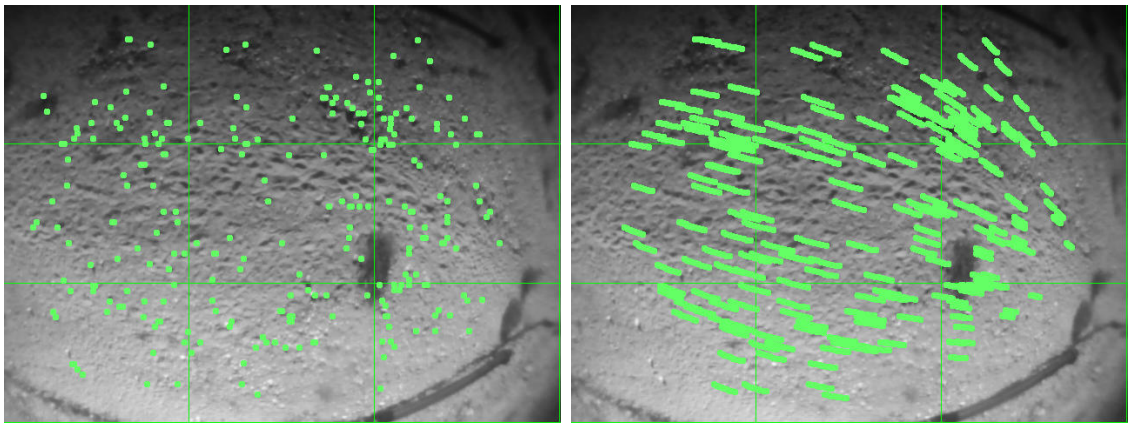
Vision processing involves: detection of features in a base frame, tracking of features in the following frames and outlier rejection to avoid passing invalid tracks to the navigation filter. For performance, processing is executed on distorted (non-rectified) images and a features are identified with a modified FAST (Features from Accelerated Segment Test) corner detector[46, 47]. FAST explores the differences in brightness between an evaluated center pixel and neighboring pixels that are located on a circle around the center pixel and achieves a significant speed-up compared to other detectors, being one of the fastest [48]. Tracking is done frame-to-frame with a Kanade-Lucas-Tomasi (KLT) tracking framework, which uses an iterative gradient-descent search algorithm. KLT could get caught in a local minimum, resulting in false matches, so an outlier rejection step is implemented through a homography-based RANSAC algorithm to identify the largest feature inlier set. This further enforces the ground plane assumption of the navigation filter since only features that are located on a common ground plane between a base frame and a Search frame survive the outlier rejection. Additionally, this scheme guarantees that all features that are detected on non-static texture like the helicopter shadow are also eliminated by the outlier rejection step under vehicle translation (but not pure rotation) [41].

After detection, all features pass through a final sorting step to only select the strongest features for ingestion into the navigation filter. The image gets divide in a 3x3 grid and the selected number of strongest features in each tile get chosen as features for the base frame. This distributes them over all the image to avoid

having all features clustered in few locations. When the total number of features drops below a parameter m or the number of tiles that do not contain any features exceeds a value k, a new base frame is requested. To help minimize drift induced by tracking over non-flat terrain, there is a maximum of n frames between two consecutive base frames. This follows from the fact that locally, over short distances, terrain tends to look planar. These numbers are chosen as $m = 40$ and $k = 3$ and the track lengths are limited to $n = 10$ frames. "Generally, larger values for n best support hovering since there will be fewer base-frame updates and therefore less drift, while lower values best support forward flight since shorter base-to-base intervals act to reduce navigation errors induced by traversing terrain that is not reasonably flat" [41].

To help deal with wind gusts, the Mars Helicopter was designed to handle up to 80 deg/s rotation rates. This ended up being a challenge for the navigation system, since the KLT tracker works best for small movements between frames. To achieve the desired performance, the software is sped up to process frames at 30Hz, but this resulted insufficient, so gyro derotation is also used. "Delta-angles from the gyro are incorporated into the vision tracking algorithms to predict future feature locations through large rotations. KLT linearizes the optical flow search around the initial feature position and becomes sensitive when the search position is initialized far from the true feature location. Gyro-based derotation has been found to significantly reduce the chance of tracking outliers and makes the vision processing more robust" [41].

The use of commercial electronics was critical to achieve the performance requested: "The Mars Helicopter would not have been possible without the low size, weight and power functionality offered by COTS hardware. The Snapdragon processor's four cores were assigned such that an entire core was dedicated to vision processing and an entire core to the navigation filter. This turned out to be essential for achieving the 30 Hz image frame rate needed to track features through wind-gust-induced vehicle rates of up to 80 deg/s" [41].



(a) (b)

Figure 3.4: New base frame with detected features on a 3x3 tile grid (a) and the surviving tracks after 10 frames (b). Credits: [41]

3.1.4 Limitations

The vertical altitude measured from an inertial frame of reference is of course different from the AGL altitude, and the two are the same only when the ground is perfectly flat. When traversing non-flat or irregular terrain, MAVeN can not differentiate between the two, so the filter is intentionally detuned to produce AGL estimation of vertical position and inertial estimation of vertical velocity. AGL altitude is critical to avoid collision with terrain, while inertial velocity is critical for controlling helicopter dynamics. Filter tuning is performed by increasing LRF and camera weighting in the vertical position, and IMU weighting in the vertical velocity. However, this compromise still produces noticeable errors in vertical velocity estimates when flying over rough terrain. Furthermore, since there is no absolute attitude sensors and the gyro error is too big to obtain precise attitude using only IMU integration, it was necessary to augment the state with attitude states. In doing so the attitude estimate becomes sensitive to non-flat terrain. For example, when flying forward over a long uphill stretch, the pitch error converges to the slope of the hill [41].
Everything considered, the MAVeN algorithm is limited to fairly flat terrain and, as a matter of fact, the operational terrain of Mars helicopter Ingenuity is expected to have sustained slopes no larger than 1-3 deg.

3.2 LuNaDrone system

Given the level of miniaturization of the sensors used for Ingenuity, which reduces total mass and volume of the spacecraft; the availability of sensors as COTS hardware, which reduces costs and time of development; the self-contained nature of the system and the non-necessity of pre-mapped landmarks, which offers maximum flexibility of the mission, it follows that a navigation system like the one of Ingenuity would be ideal also for LuNaDrone. Unfortunately, LuNaDrone will not operate in an environment where the flat terrain assumption is respected. Indeed, LuNaDrone's goal entails entering pits with funnels and vertical walls taller than 100 m [13, 11]. A faceted terrain model, known before the flight, could be used instead of a flat plane (in fact, this was in the original concept behind MAVeN, which was originally developed for comet exploration [49]) but such model is not available for LuNaDrone mission, given the highly irregular and often unknown morphology of pits and skylights.

With future mars helicopters missions in mind, JPL developed a new navigation framework called xVIO that, using the same type of sensors of Ingenuity, drops the assumption of a flat terrain among other advantages [50, 51, 52]. It is stated that this new system has been tested in real-time operations on 1 core of a Qualcomm Snapdragon 820, which is the successor of the Snapdragon 801 used on Ingenuity [50, 52]. Instead of lying on a plane, the visual features in xVIO form a faceted geometry over the terrain. IMU, camera and rangefinder data is fused together by an extended Kalman filter (EKF). During the prediction step of the filter, the state of the system is propagated forward in time by integration of accelerometer

and gyroscope data at high frequency (hundreds of Hz). In the update steps of the EKF, camera frames and rangefinder data are processed at a lower frequency to correct this estimate. Features are tracked between frames and used for the visual update through a hybrid approach that uses both *multi-state constraint kalman filter* (MSCKF) [45] and *simultaneous localization and mapping* (SLAM) paradigms to optimize computational cost and accuracy [53][52]. Like in Ingenuity, the presence of the rangefinder allows state estimation in absence of accelerated motion, as in a steady hover or constant speed traverse.

JPL's xVIO framework represent the optimal solution also for LuNaDrone mission and as such in the rest of this thesis the framework is studied and a MATLAB implementation is developed for initial validation and testing.

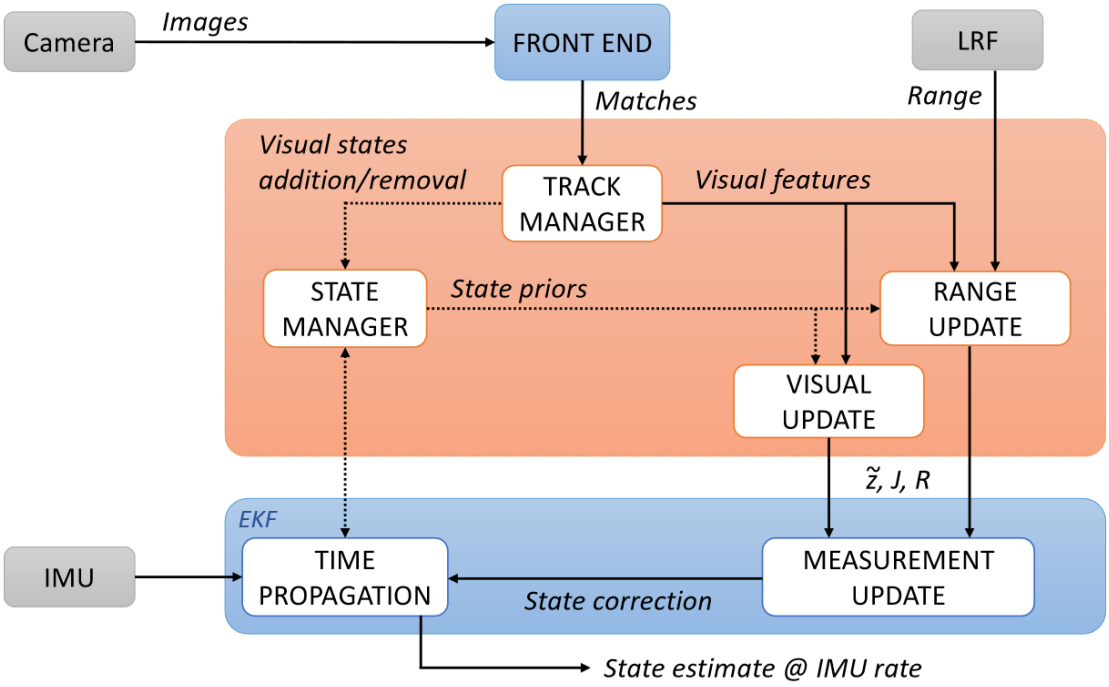


Figure 3.5: xVIO architecture. Credits: [51]

Chapter 4

Filter basics

In robotics, localization problems at their core are problems of state estimation. The robot maintains an internal belief with regards to the state of its environment and can sense it through its sensor and influence it through its actuators. Sensors carry only partial information, and their measurements are corrupted by noise, while the actuators can be influenced by several and often random factors in the environment. State estimation seeks to recover the state from this data
The state is the collection of all the variables describing the robot and the environment which can influence the robot in the future and is called complete if it is the best predictor of the future. Completeness entails that knowledge of past states, measurements, or controls carry no additional information that would help us to predict the future more accurately. This doesn't imply that the future is deterministically derived from the state, it could be stochastic but with no variables prior to the current state that influence the stochastic evolution. This property is called Markov assumption and processes that meet these conditions are known as Markov chains. In reality, it is impossible to specify a complete state for a robot, since this would include all aspects of the environment, including the internal dynamics and data of the robot itself and of external actors (natural or artificial) operating in the same environment. Practical implementations therefore use a subset of the variables, an incomplete state, which is enough for state estimation in the required operating conditions.

A robot interacts with the environment through sensor measurements, often called *observations*, which provide information on a momentary state of the environment, or trough control actions (often called *process*), like motion, which change the state of the environment. These interactions carry a certain grade of unpredictability. Probabilistic algorithms model this uncertainty explicitly and for this reason are more robust in the face of sensor limitations, sensor noise, environment dynamics, often scaling much better to complex and unstructured environments [54].
In a discrete representation, at the time t, let's call the state of the system x_t, the measurements z_t and the control actions u_t. The probabilistic distribution of the state x_t can be expressed in terms of all the past states, controls and measurements but, if the state is complete (the Markov assumption is true), it is dependent only

on the state x_{t-1} and the control actions u_t, in mathematical terms:

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

Also if the state is complete, x_t is sufficient to predict the measurement z_t:

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

$p\left(x_t|x_{t-1}, u_t\right)$ and $p\left(z_t|x_t\right)$ represent the *state transition probability* and the *measurement probability*, respectively.

A distinction has to be made between the true state x_t and the *belief* $\hat{x}_t$.The belief is the robot's internal knowledge of the environment state and it is expressed as a probability distribution conditioned on all the past measurements $z_{1:t}$ and controls $u_{1:t}$:

$$\hat{x}_t = p\left(x_t|z_{1:t}, u_{1:t}\right)$$

It assign a probability to every possible hypothesis regarding the true state after all the measurements and controls are known and is also called the *posterior* distribution. If it is calculated before incorporating the last measurement z_t, it is referred to as the *prediction* or *prior*:

$$\hat{x}_t^- = p\left(x_t|z_{1:t-1}, u_{1:t}\right)$$

Calculating the posterior $\hat{x}_t$ from the prior $\hat{x}_t^-$ is called *correction* or *measurement update*.

The more general algorithm for calculating beliefs from control and measurements is the Bayes filter. It is a recursive estimator that calculates the belief at time t from the belief at time $t-1$ and the most recent control u_t and measurement z_t. The Bayes filter is based on the Markov assumption. However, this category of filters has been found to be surprisingly robust to violations of the assumption in practical applications with incomplete state representations, particularly so if the effects of unmodeled variables are close to random.
The most popular family of recursive state estimators are the *gaussian* filters, a type of Bayes filter in which beliefs are represented by a multivariate normal distribution:

$$p(x) = \det\left(2\pi\Sigma\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{4.1}$$

Like in a single variable normal distribution there is a scalar mean value and a scalar variance, in a multivariate normal distribution there is a mean vector μ and a covariance matrix Σ. The representation of a gaussian by these two parameters is called the *moments representation*. The covariance matrix between the components of a vector x is defined as:

$$\Sigma(x,x) = \Sigma_{xx} = E\left[(x - E[x])(x - E[x])^T\right] \tag{4.2}$$

where E denotes the *expected value*. The covariance matrix is square, symmetric, positive-semidefinite and its dimensionality is the square of the dimension of x.

4.1 Kalman filter

Perhaps the best studied among Bayes filters is the Kalman filter (KF) [55], a linear gaussian estimator. In addition to the Markov assumption of the Bayes filter, the beliefs are gaussians if the following properties are true [54]:

- The state transition probability $p(x_t|x_{t-1}, u_t)$ is a linear function of its arguments, plus gaussian noise with zero mean and covariance Q.

$$x_t = Ax_{t-1} + Bu_t + w_t \tag{4.3}$$

$$p(w_t) = N(0, Q) \tag{4.4}$$

  This means the system dynamics are linear and the randomness in the state transition is modeled by the gaussian term. The state transition probability is given by substituting (4.3) as the mean term in the definition of the multivariate normal distribution (4.1), and R as Σ:

$$p(x_t|u_t, x_{t-1}) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp(-\frac{1}{2}(x_t - Ax_{t-1} + Bu_t)^T Q_t^{-1}(x_t - Ax_{t-1} + Bu_t))$$

- The measurement probability $p(z_t|x_t)$ is a linear function of its arguments, plus gaussian noise with zero mean and covariance R.

$$z_t = Hx_t + v_t \tag{4.5}$$

$$p(v_t) = N(0, R) \tag{4.6}$$

  This means the measurements are linear with respect to the state and the noise of the sensor is modeled by a gaussian. The measurement probability is given by substituting in (4.1):

$$p(z_t|x_t) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(z_t - Hx_t)^T R_t^{-1}(z_t - Hx_t)\right)$$

- The initial belief at $t = 0$ is normally distributed:

$$p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\right)$$

Under these assumptions, the Kalman Filter is optimal. If the noise is not gaussian, the KF is the optimal *linear* estimator in the minimum mean-square-error sense [56].
The KF algorithm is an iterative method which alternates between a *prediction* step and a *correction* step. In the former, the filter projects forward in time the current state of the system to obtain a prior estimate of the state, along with the uncertainties, while in the latter a feedback is obtained in the form of noisy measurements and is incorporated in the prior estimate to obtain an improved posterior estimate of the state and uncertainties. Being a recursive algorithm, it is suitable for real time applications.

Prediction step

The discrete *time update* (or *propagation*) equations are:

$$\hat{x}_t^- = Ax_{t-1} + Bu_t \qquad \text{(State prediction)}$$

$$P_t^- = AP_{t-1}A^T + Q \qquad \text{(Covariance prediction)}$$

A, B are the matrices that describe the dynamics of the process in (4.3) while Q is the process noise covariance matrix in (4.4). In general, these matrices do not have to be constant on each step. These equations bring the system forward in time from $t-1$ to t.

Correction step

The discrete *measurement update* (or simply *update*) equations are:

$$\tilde{y}_t = z_t - H\hat{x}_t^- \qquad \text{(Innovation or residual)}$$

$$S_t = HP_t^-H^T + R \qquad \text{(Innovation covariance)}$$

$$K_t = P_t^-H^TS_t^{-1} \qquad \text{(Kalman gain)}$$

$$\hat{x}_t = \hat{x}_t^- + K_t\tilde{y}_t \qquad \text{(State update)}$$

$$P_t = (I - K_tH)\,P_t^- \qquad \text{(Covariance update)}$$

First, the real measurements z_t are compared with the expected value from the measurement model in (4.5) to obtain the residuals, and the covariance of the innovation is calculated taking into consideration the covariance R of the noise of the sensors in (4.6). To obtain a posterior estimate of the state which weights the prediction against the measurement, the Kalman gain is calculated, which takes into account both the process noise (through the effect of Q on P_t^-) and the measurement noise (through the effect of R on S_t). For example, in the presence of sensors which are very noisy, the terms in the covariance of the measurements R are very large, which results in large terms in S and consequentially a small gain K which gives a small weight to the measurement innovation in the state update equation.

While the covariance of the measurements R is of easy definition, since it comes directly from the noise variances of the sensors which can be known through testing, the covariance of the process noise Q often changes in time and depends on a process that we usually cannot be observed directly. As a result, these matrices are often obtained by a process of tuning. In some cases a poor model of the process can be used in a Kalman filter and produce acceptable results if the shortcomings of the model are "masked" by large values in Q (high uncertainty).

4.2 Extended Kalman Filter

the Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by a **linear** stochastic difference equation. Some of the most successful applications of the filter, however, have been in situations where the process to be estimated and/or the measurement model is not linear. A Kalman filter that linearizes about the current mean and covariance is referred to as an *Extended Kalman filter* or EKF. This time, the state changes by a non-linear process:

$$x_t = f\left(x_{t-1}, u_t, w_t\right)$$

with measurement:

$$z_t = h\left(x_t, v_t\right)$$

where w_t and v_t again represent the process and measurement noise. When calculating the predicted state $\hat{x}_t$ and measurement $\hat{z}_t$, the noise term is obviously assumed zero, since it is unknown:

$$\hat{x}_t^- = f\left(\hat{x}_{t-1}, u_t, 0\right) \tag{4.7}$$

$$\hat{z}_t = h\left(\hat{x}_t^-, 0\right) \tag{4.8}$$

The fundamental flaw of the Extended Kalman Filter lies in the fact that the normal distribution of the variables is no longer normal after undergoing a non-linear transformation. Therefore, EKF employs linearization, which approximates a function by a linear function that is tangent to it at the mean of the Gaussian. By projecting the Gaussian through this linear approximation, the posterior is Gaussian. In fact, once g and f are linearized, the mechanics of belief propagation are equivalent to those of the Kalman filter [54]. EKF uses a first order Taylor expansion which linearize an estimate about (4.7) and (4.8):

$$x_t \approx \hat{x}_t^- + F\left(x_{t-1} - \hat{x}_{t-1}\right) + W w_t$$

$$z_t \approx \hat{z}_t + H\left(x_t - \hat{x}_t^-\right) + V v_t$$

In this model, the true state x_t differs from the estimated prior $\hat{x}_t^-$ by a term which is linear with respect to the error at the previous step (the difference between the true state at the previous step x_{t-1} and the best estimate at the previous step $\hat{x}_{t-1}$) and whose slope is defined by the jacobian F. The measurement z_t differs from the estimated $\hat{z}_t$ by a term which is linear with respect to the prediction error at the current step (the difference between the true state x_t and the predicted state at the current step $\hat{x}_t^-$) and whose slope is defined by the jacobian H. In both there is an added noise term which is the process noise w_t or measurement noise v_t multiplied by their respective jacobians W or V.

The difference between the true state and the estimated state is called the error state and, depending on whether it is relative to a posterior or a prior estimate, is indicated as:

$$\delta x_t = x_t - \hat{x}_t$$

$$\delta x_t^- = x_t - \hat{x}_t^-$$

The difference between measurement and prediction of the measurement (the residual) is indicated as:

$$\delta z_t = z_t - \hat{z}_t$$

so the model can be written as:

$$\delta x_t^- \approx F\delta x_{t-1} + Ww_t \tag{4.9}$$

$$\delta z_t \approx H\delta x_t^- + Vv_t \tag{4.10}$$

In the first equation, the model describes he evolution of the error state as a linear function of the previous error state and the process noise. In the second equation it describes the residual as a linear function of the current error state and the measurement noise.
The rationale behind writing (4.10) is that the state error is not known but the residual is, which means information about the state can be recovered (probabilistically because of the presence of noise).

Prediction step

The discrete *time update* (or *propagation*) equations become:

$$\hat{x}_t^- = f\left(\hat{x}_{t-1}, u_t, 0\right) \qquad \text{(State prediction)}$$

$$P_t^- = FP_{t-1}F^T + WQW^T \qquad \text{(Covariance prediction)}$$

In general, the matrices F, H, W and Q are not constant on each step. The subscript t is dropped to maintain a lighter notation.

Correction step

The discrete *measurement update* (or simply *update*) equations become:

$$\tilde{y}_t = z_t - h\left(\hat{x}_t^-, 0\right) \qquad \text{(Innovation or residual)}$$

$$S_t = HP_t^-H^T + VRV^T \qquad \text{(Innovation covariance)}$$

$$K_t = P_t^-H^TS_t^{-1} \qquad \text{(Kalman gain)}$$

$$\hat{x}_t = \hat{x}_t^- + K_t\tilde{y}_t \qquad \text{(State update)}$$

$$P_t = \left(I - K_tH\right)P_t^- \qquad \text{(Covariance update)}$$

The matrices V and R are also in general not constant on each step, but the subscript t is dropped to maintain a lighter notation.

The EKF strength is its simplicity and computational efficiency (each update requires time $O\left(k^{2.8} + n^2\right)$ where k and n are the dimensions of the measurement and the state vector, respectively) and EKFs have been applied with great success to a

number of state estimation problems that violate the underlying assumptions [54]. The advantages come from the representation of beliefs through multivariate gaussian distributions. However, this is also the limit of this type of filters in applications where the real world distributions are multimodal, for example when two distinct hypotesis of the machine state exist whose mean is not a likely candidate.

Chapter 5

State definition and time propagation

5.1 Notation

p_a^b and v_a^b are the position and acceleration of frame of reference $\{b\}$ with respect to frame of reference $\{a\}$. When the coordinates are expressed in the axis of $\{a\}$, for example, they are written as ${}^a p_a^b$ and ${}^a v_a^b$. If not indicated, they are represented by default by their coordinates in the origin frame:

$$p_a^b = {}^a p_a^b$$

$$v_a^b = {}^a v_a^b$$

q_a^b is the quaternion describing the rotation from $\{a\}$ to $\{b\}$ and $C(q_a^b)$ is the coordinate change matrix associated to that quaternion, such that:

$${}^b x = C(q_a^b) {}^a x$$

The coordinate change matrix is the transpose of what is commonly called the rotation matrix.

5.2 State definition

In xVIO, the state vector x is divided between the IMU states x_I and the vision states x_V:

$$x = \begin{bmatrix} x_I \\ x_V \end{bmatrix}$$

The IMU state contains information about the position p_w^i, velocity v_w^i and orientation q_w^i of the system, and the bias terms of the gyroscope b_g and accelerometer

b_a.

$$x_I = \begin{bmatrix} p_w^i \\ v_w^i \\ q_w^i \\ b_g \\ b_a \end{bmatrix}$$

The notation used in p_w^i, as an example, means that the variable represents the position of the IMU frame $\{i\}$ with respect to the world frame $\{w\}$. The world frame is the inertial frame fixed to the terrain, with the z axis pointing upward. The IMU frame is the reference frame fixed to the body of the system, its axis are the same as the accelerometer and gyroscope measurements.
The position, velocity and the bias terms are each represented by a 3×1 vector, while the orientation is a 4×1 quaternion, for a total IMU state size of 16×1. It might seems strange that biases are introduced into the state vector when their estimation is not requested as an output of the navigation system, but remember from the previous chapter that the state has to include ideally all variables that can influence it in the future and, sure enough, the biases have a relevant effect on the system and can be quite large for commercial MEMS IMUs.

The vision states include N *feature states* in inverse depth parametrization and the camera *poses* (position and orientation) during the last M camera frames, called the *sliding window states*. For every new frame, the oldest camera pose is removed from the state vector, the remaining poses are slid one position and a new pose is added to the state. Poses are used for both the MSCKF update and the SLAM update, while features are only used for SLAM.

$$x_V = \begin{bmatrix} p_w^{c_1} \\ \vdots \\ p_w^{c_M} \\ \\ q_w^{c_1} \\ \vdots \\ q_w^{c_M} \\ \\ f_1 \\ \vdots \\ f_N \end{bmatrix}$$

$p_w^{C_i}$ and $q_w^{C_i}$ are the position and orientation of the i-th most recent camera frame of reference $\{C_i\}$ with respect to the IMU frame $\{i\}$. The origin of the camera frame is in the optical center of the lens with the z axis pointing outward along the optical axis, the x axis pointing right in the image plane and the y axis pointing down in the image plane. A pose necessitates of 7 states (3 for the position vector and 4 for the orientation quaternion), while the features in inverse depth parametrization need 3, so the total size of the vision states is $(7M + 3N) \times 1$.

5.2.1 Inverse depth parametrization

Inverse depth parametrization is a way to represent point features that was introduced in [57, 58]. It offers improved convergence properties, can cope with features over a huge range of depths and even at infinity and permits efficient and accurate representation of uncertainty during undelayed initialization (when features are added to the state vector after a single observation).
Applied to xVIO, the feature point j is described by the *anchor* pose (position ${}^{c_{i_j}}p_w^w$ and orientation ${}^{c_{i_j}}p_w^w$) of the camera when the feature was first observed, two parameters α_j and β_j which are the coordinates of the projection of the feature in the image plane, and ρ_j which is the inverse of the distance between the optical center and the the plane, parallel to the image plane, which contains the feature. This way, the point is defined by the ray that starts from the optical center and goes through the projection of the point on the image plane and from the inverse of the distance along this ray, as in figure 5.1. The cartesian coordinates in world frame become:

$$ {}^{w}p_j = \begin{bmatrix} {}^{w}x_j \\ {}^{w}y_j \\ {}^{w}z_j \end{bmatrix} = {}^{c_{i_j}}p_w^w + \frac{1}{\rho_j} C({}^{c_{i_j}}q_w^w)^T \begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} $$

As it is apparent, a point at infinite distance has $\rho_j = 0$, a property that will come useful later. The anchor pose for each feature in the state vector is one of the sliding window states; for this reason when a pose that is an anchor for some features is about to leave the state vector, the associated features need to be reparametrized with respect to a more recent pose, a procedure that is detailed in section 7.1.1.

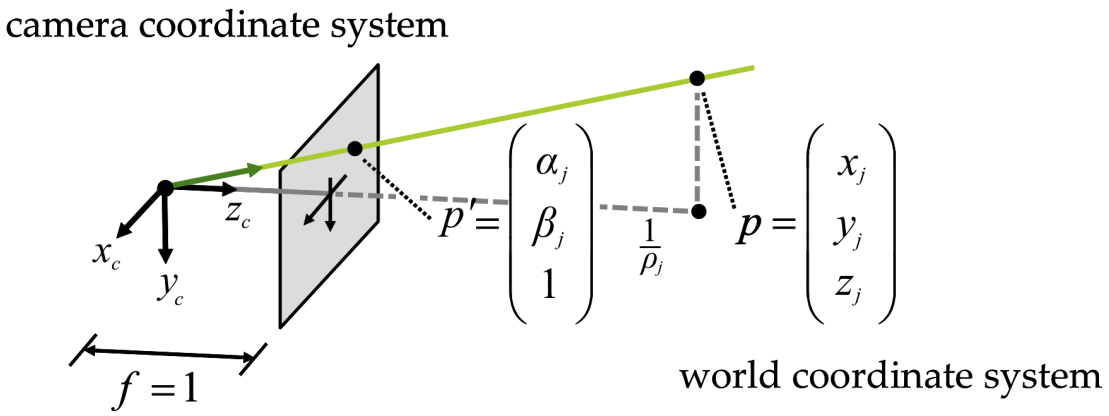


Figure 5.1: Inverse depth parametrization in the pinhole camera model

5.2.2 Error states

The model of the EKF in equations (4.9) and (4.10) is linearized with respect to the **error states**, which in this case are defined with a peculiarity. The error states are:

$$ \delta x = \begin{bmatrix} \delta x_I \\ \delta x_V \end{bmatrix} $$

where the inertial error states are:

$$\delta x_I = \begin{bmatrix} \delta p_w^i \\ \delta v_w^i \\ \delta \theta_w^i \\ \delta b_g \\ \delta b_a \end{bmatrix}$$

Position, velocity and biases error states are simply the difference between the true and estimated vector, for example:

$$\delta p_w^i = p_w^i - \hat{p}_w^i$$

The quaternion error could be δq, defined such that the quaternion product between the estimate and the error gives the true orientation quaternion:

$$q = \hat{q} \otimes \delta q$$

However, using the small angle approximation (in Hamilton notation):

$$\delta q \simeq \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta \end{bmatrix}$$

Therefore, $\delta\theta$ can be used as a correct minimal representation of the error quaternion which also increases numerical stability [59]. The size of the inertial error states is then reduced to 15×1.

Likewise, for the vision error states:

$$\delta x_V = \begin{bmatrix} \delta p_w^{c_1} \\ \vdots \\ \delta p_w^{c_M} \\ \\ \delta \theta_w^{c_1} \\ \vdots \\ \delta \theta_w^{c_M} \\ \\ f_1 \\ \vdots \\ f_N \end{bmatrix}$$

whose size becomes $(6M + 3N) \times 1$.

To avoid any confusion, let's specify that error states here presented are a mathematical definition and, unlike system states, do not correspond to actual variables in memory in the filter implementation.

5.3 Time propagation

Referring to the KF and EKF models presented before, the "control term" in this case is represented by the angular rates and linear accelerations on the spacecraft,

as measured by the IMU. Therefore, the time propagation happens according to an inertial model, which is the same as in [60]:

$$\begin{cases} \dot{p}_w^i = v_w^i \\ \dot{v}_w^i = a_w^i \\ \dot{q}_w^i = \frac{1}{2}\Omega({}^i\omega_w^i)q_w^i \end{cases}$$

where:

$$\Omega(\omega) = \begin{bmatrix} 0 & -\omega^T \\ \omega & -\lfloor\omega\times\rfloor \end{bmatrix}$$

$$\lfloor\omega\times\rfloor = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

$\lfloor\omega\times\rfloor$ is the skew-symmetrix matrix which represent the cross product as a matrix multiplication.

The acceleration a_w^i and angular rates v_w^i are not directly known, but are related to the measured values from the IMU:

$$\omega_{\text{IMU}} = {}^i\,\omega_w^i + b_g + n_g$$

$$a_{\text{IMU}} = C(q_w^i)(a_w^i - {}^w\,g) + b_a + n_a$$

g is the gravity vector, n_g and n_a are zero-mean gaussian white noises.
The biases do not change if not for a random walk process driven by zero mean gaussian white noises n_{b_g} and n_{b_a}:

$$\begin{cases} \dot{b}_g = n_{b_g} \\ \dot{b}_a = n_{b_a} \end{cases}$$

5.3.1 State time propagation

The final inertial state propagation model in continuous time is (note: $C(q_w^i)^T = C(q_i^w)$):

$$\begin{cases} \dot{p}_w^i = v_w^i \\ \dot{v}_w^i = C(q_w^i)^T(a_{\text{IMU}} - b_a - n_a) + {}^w\,g \\ \dot{q}_w^i = \frac{1}{2}\Omega(\omega_{\text{IMU}} - b_g - n_g)q_w^i \\ \dot{b}_g = n_{b_g} \\ \dot{b}_a = n_{b_a} \end{cases} \tag{5.1}$$

The vision states do not have any dynamics since they refer to static poses or features on the terrain, so:

$$\dot{x}_V = 0$$

The state is propagated forward in time by taking the expected value (i.e. removing the noise terms) and integrating in time at first order as detailed in [59]. The discrete time update becomes, for the inertial states:

$$\begin{cases} \hat{a}_t = a_{\text{IMU}} - \hat{b}_{a,t-1} \\ \hat{\omega}_t = \omega_{\text{IMU}} - \hat{b}_{g,t-1} \\ \hat{q}_t^- = \left(\exp\left(\frac{1}{2}\Omega\left(\frac{\hat{\omega}_t + \hat{\omega}_{t-1}}{2} \right) \Delta t \right) + \frac{1}{48}\Delta t^2 \left(\Omega(\hat{\omega}_t)\Omega(\hat{\omega}_{t-1}) - \Omega(\hat{\omega}_{t-1})\Omega(\hat{\omega}_t) \right) \right) \hat{q}_{t-1} \\ \hat{v}_t^- = \hat{v}_{t-1} + \left(\frac{1}{2}\left(C(\hat{q}_t^-)^T \hat{a}_t + C(\hat{q}_{t-1})^T \hat{a}_{t-1} \right) +^w g \right) \Delta t \\ \hat{p}_t^- = \hat{p}_{t-1} + \frac{1}{2}\left(\hat{v}_t^- + \hat{v}_{t-1} \right) \\ \hat{b}_{a,t}^- = \hat{b}_{a,t-1} \\ \hat{b}_{g,t}^- = \hat{b}_{g,t-1} \end{cases}$$

and for vision states:

$$\hat{x}_{V,t}^- = \hat{x}_{V,t-1}$$

5.3.2 Covariance time propagation

In continuous time, the inertial error states evolve according to:

$$\dot{\delta x_I} = F_c \delta x_I + G_c n_{\text{IMU}} \tag{5.2}$$

$$n_{\text{IMU}} = \begin{bmatrix} n_a^T \\ n_{b_a}^T \\ n_g^T \\ n_{b_g}^T \end{bmatrix}$$

whose matrices are obtained from the inertial propagation model in continuous time (5.1) as such:

$$F_c = \begin{bmatrix} 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C(\hat{q}_t^-)^T \lfloor \hat{a}_t \times \rfloor & 0_{3\times3} & -C(\hat{q}_t^-)^T \\ 0_{3\times3} & 0_{3\times3} & -\lfloor \hat{w}_t \times \rfloor & -I_3 & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

$$G_c = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C(\hat{q}_t^-)^T & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -I_3 & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

Considering only the portion of the covariance relative to the inertial states, the discretization follows the method in [60]. The discrete time update step is:

$$P_t^- = F_d P_{t-1} F_d^T + Q_d \tag{5.3}$$

with the matrix F_d obtained by solving the ordinary matrix differential equation (5.2) and discretizing assuming F_c and G_c as constant over Δt:

$$F_d = \exp\left(F_c \Delta t\right) = I_d + F_c \Delta t + \frac{1}{2!} F_c^2 \Delta t^2 + \dots$$

However, for small values of $|\omega|$, the expression leads to numerical instability [59]. By using the small angle approximation for which $|\omega| \to 0$ and applying L'Hopital's rule, the following solution is obtained:

$$F_d = \begin{bmatrix} I_3 & \Delta t & A & B & -C(\hat{q}_t^-)^T \frac{\Delta t^2}{2} & 0_{3\times 7} \\ 0_{3\times 3} & I_3 & C & D & -C(\hat{q}_t^-)^T \Delta t & 0_{3\times 7} \\ 0_{3\times 3} & 0_{3\times 3} & E & F & 0_{3\times 3} & 0_{3\times 7} \\ 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 3} & I_3 & 0_{3\times 3} & 0_{3\times 7} \\ 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 3} & I_3 & 0_{3\times 7} \\ 0_{7\times 3} & 0_{7\times 3} & 0_{7\times 3} & 0_{7\times 3} & 0_{7\times 3} & I_7 \end{bmatrix}$$

$$A = -C(\hat{q}_t^-)^T \lfloor \hat{a}_t \times \rfloor \left(\frac{\Delta t^2}{2} - \frac{\Delta t^3}{3!} \lfloor \omega \times \rfloor + \frac{\Delta t^4}{4!} \lfloor \omega \times \rfloor^2 \right)$$

$$B = -C(\hat{q}_t^-)^T \lfloor \hat{a}_t \times \rfloor \left(-\frac{\Delta t^3}{3!} + \frac{\Delta t^4}{4!} \lfloor \omega \times \rfloor - \frac{\Delta t^5}{5!} \lfloor \omega \times \rfloor^2 \right)$$

$$A = -C(\hat{q}_t^-)^T \lfloor \hat{a}_t \times \rfloor \left(\Delta t \cdot I_3 - \frac{\Delta t^2}{2!} \lfloor \omega \times \rfloor + \frac{\Delta t^3}{3!} \lfloor \omega \times \rfloor^2 \right)$$

$$D = -A$$

$$E = I_3 - \Delta t \lfloor \omega \times \rfloor + \frac{\Delta t^2}{2!} \lfloor \omega \times \rfloor^2$$

$$F = -\Delta t \cdot I_3 + \frac{\Delta t^2}{2!} - \frac{\Delta t^3}{3!} \lfloor \omega \times \rfloor^2$$

The discrete time noise covariance matrix can then be calculated as:

$$Q_d = \int_{\Delta t} F_d(\tau) G_c Q_c G_c^T F_d(\tau)^T d\tau$$

where the covariance matrix of the noise in continuous time is:

$$Q_c = \begin{bmatrix} \sigma_{n_a}^2 & & & \\ & \sigma_{n_{b_a}}^2 & & \\ & & \sigma_{n_g}^2 & \\ & & & \sigma_{n_{b_g}}^2 \end{bmatrix}$$

The practical algorithm and code for calculating of Q_d is obtained from the work of [61, 62].

Let's remember that (5.3) is only relative to the propagation of the covariance of inertial states. Vision error states have no dynamics just like vision states:

$$\dot{\delta x_V} = 0$$

For this reason, the covariance of the whole state can be easily calculated by subdividing it into four parts:

$$P = \begin{bmatrix} P_{II} & P_{IV} \\ P_{VI} & P_{VV} \end{bmatrix}$$

P_{II} corresponds to the covariance of inertial error states only, as in equation (5.3), and has size 15×15. P_{VV} is the covariance of vision states only and has size $(6M + 3N) \times (6M + 3N)$. The time update of the covariance blocks is:

$$\begin{cases} P^-_{II,t} = F_d P_{II,t-1} F_d^T + Q_d \\ P^-_{IV,t} = F_d P_{IV,t-1} \\ P^-_{VI,t} = \left(P^-_{IV,t}\right)^T \\ P^-_{VV,t} = P_{VV,t-1} \end{cases}$$

Chapter 6

Measurement Update

6.1 The hybrid approach

Simultaneous Localization And Mapping within an EKF provides a filter update when it observes a feature in the world that corresponds to one of the features included in the state vector. Due to the augmentation with these feature states, not only an estimate of the position of the vehicle (localization) is recovered from the observations, but also the position of the features themselves (mapping), hence the name. This is one of the most used and studied method for visual navigation [42], but it has been found challenging for real time application due to a computational cost which scales as the cube of the number of features N in the state. While SLAM algorithms have been indeed used in applications where obtaining a map of the environment is part of the goals, in xVIO the ultimate objective is limited to the current estimation of the vehicle position, velocity and attitude. Because of this and to keep the computation bounded, the algorithm removes the features from the state as soon as they are lost by the visual tracker, rendering the mapping only temporary and limited to the part of the scene which is currently in the field of view. Obviously, without a persistent map, no loop closure is considered.

The Multi-State Constraint Kalman Filter is a paradigm for visual navigation which makes use of the geometric constraints that arise from the observation of features from multiple camera poses. It was originally presented in [45] and it has since gained major importance in the field of visual navigation due to its advantages, most importantly that features do not have to be included in the states (only camera poses have to), which allows efficient tracking of a large number of features without the computational costs associated with large filters states of different paradigms such as EKF-SLAM. In MSCKF, the cost is only linear with respect to the number of features N, and cubical in the number of poses M in the sliding window[45].
The measurements of a single feature in consecutive camera poses (the *track*) are processed as a batch either when the feature is lost or after the length of the track exceeds the length of the sliding window of poses included in the state vector. This is one of the shortcomings of MSCKF, since it introduces a delay from the first appearance of a feature to the processing of its track, the other being the necessity

of a minimum camera translation between poses, which means that MSCKF can't constrain the state when the vehicle is not moving. On the contrary, SLAM can provide an update at image rate and in hovering. In addition, the mapping of features in SLAM is also essential for performing the range measurement update because the SLAM features being estimated constitute the vertices of triangular facets which form a polygonal surface approximating the shape of the terrain. The measurement model of the range update involves estimation of the impact point of the ray coming from the laser rangefinder onto these facets.

MSCKF vs SLAM then becomes a trade-off problem, hence both paradigms are used in a hybrid fashion with the logic described in [53][63]. Since MSCKF has a computational cost which is cubic in the number of poses M, while EKF-SLAM is cubic in the number of features M, it follows that if a small number of features are tracked for a large number of frames, it is convenient to use EKF-SLAM, while if a large number of features are tracked for a short time it is more convenient to employ MSCKF. In real world scenarios most of the features are tracked only for short times, with only few surviving over a large number of frames. The optimal strategy in terms of computational requirements, according to the results of [53], is to process a feature using MSCKF if it is lost after fewer than M frames while, instead, if it is still being tracked after M images, the M measurements are used to triangulate it, compute its initial covariance, compute the cross-correlation with other filter states and initialize it into the state vector, for subsequent use with SLAM. The only parameter to optimize is the size of the sliding window, M. The best value at any given moment depends on the future behavior of the feature tracks. Clearly, it is impossible to obtain such information, but it is possible to collect statistical data about tracks during runtime and to use this information to minimize the expected cost of the EKF updates by actively adjusting the parameter M. In two test with synthetic datasets, [53] claims that the optimal hybrid filter has a runtime 37.17% smaller than that of the MSCKF, and 72.8% smaller than EKF-SLAM in one case and a runtime 45.8% smaller than the MSCKF, and 55.6% smaller than SLAM in the other. As a simpler, but clearly sub-optimal alternative, M can also be a fixed value of compromise chosen beforehand by a process of tuning.

6.1.1 Image measurement model

To construct an update, we have to formulate the model of the EKF in equation (4.10) for the specific system. The matrix H is the jacobian of the function h in equation (4.8), a non-linear function which expresses the measurement z_t as a function of the state x_t. In both measurement model of SLAM and MSCKF, the observation of a single feature corresponds to the (x, y) coordinates of the normalized projection in the plane which sits at $z = 1$ in the camera frame of reference. The position of a feature j in Cartesian coordinates in the frame of reference of the camera frame i is:

$$ {}^{c_i}p_j = \begin{bmatrix} {}^{c_i}x_j \\ {}^{c_i}y_j \\ {}^{c_i}z_j \end{bmatrix} $$

which can be recovered from the position in world frame and the current pose of the camera:

$$^{c_i}p_j = C(q_w^{c_i})\,(^w p_j - p_w^{c_i}) \tag{6.1}$$

so the observation is:

$$^i z_j = \frac{1}{^{c_i}z_j}\begin{bmatrix} ^{c_i}x_j \\ ^{c_i}y_j \end{bmatrix} + ^i n_j \tag{6.2}$$

with $^i n_j$ being the noise term which, by hypothesis of the EKF, is gaussian with zero-mean.

$$^i n_j \sim \mathcal{N}(0, ^i R_j)$$

The covariance of this noise is a matrix:

$$^i R_j = \sigma_V^2 \cdot I_2 \tag{6.3}$$

meaning that the noise in the x and y direction of the image is uncorrelated and with the same standard deviation σ_V, which depends on the performance of the visual front-end. The noise is also assumed to be uniform in all the image.

The feature position in Cartesian coordinates in world frame $^w p_j$, however, is not part of the state vector. To express the measurement as a function of only states and noise, SLAM and MSCKF use different strategies.

6.2 SLAM Update

6.2.1 Model

In (6.1) the feature appears in Cartesian coordinates, but the feature in the states is represented by inverse-depth parameters, as described in section 5.2.1, so there's need to translate from one representation to another tanks to:

$$^w p_j = p_w^{c_{i_j}} + \frac{1}{\rho_j} C(q_w^{c_{i_j}})^T \begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} \tag{6.4}$$

where c_{i_j} refers to the anchor pose for the feature in inverse-depth parameters f_j:

$$f_j = \begin{bmatrix} \alpha_j \\ \beta_j \\ \rho_j \end{bmatrix}$$

By substituting (6.4) into (6.1):

$$^{c_i}p_j = C(q_w^{c_i}) \left(p_w^{c_{i_j}} + \frac{1}{\rho_j} C(q_w^{c_{i_j}})^T \begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} - p_w^{c_i} \right) \tag{6.5}$$

The coordinates of $^{c_i}p_j$ are substituted in (6.2) to estimate the observation. This defines a non-linear measurement model which is a function of the state only (plus noise).

$$^i z_j = h\left(p_w^{c_i}, q_w^{c_i}, p_w^{c_{i_j}}, q_w^{c_{i_j}}, f_j, ^i n_j\right) = h\left(x, ^i n_j\right)$$

This model is linearized to write the residuals as a linear combination of the error states:

$$ {}^{i}\delta z_j \approx \frac{\partial h}{\partial p_w^{c_i}}\delta p_w^{c_i} + \frac{\partial h}{\partial p_w^{c_{i_j}}}\delta p_w^{c_{i_j}} + \frac{\partial h}{\partial (\delta\theta_w^{c_i})}\delta\theta_w^{c_i} + \frac{\partial h}{\partial (\delta\theta_w^{c_{i_j}})}\delta\theta_w^{c_{i_j}} + \frac{\partial h}{\partial f_j}\delta f_j + \frac{\partial h}{\partial {}^{i}n_j}{}^{i}n_j \quad (6.6) $$

$$
\begin{aligned}
\frac{\partial h}{\partial p_w^{c_i}} &= \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial p_w^{c_i}} = J_j \cdot J_{p_i} = H_{p_i} \\
\frac{\partial h}{\partial p_w^{c_{i_j}}} &= \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial p_w^{c_{i_j}}} = J_j \cdot J_{p_{i_j}} = H_{p_{i_j}} \\
\frac{\partial h}{\partial (\delta\theta_w^{c_i})} &= \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial (\delta\theta_w^{c_i})} = J_j \cdot J_{\theta_i} = H_{\theta_i} \\
\frac{\partial h}{\partial (\delta\theta_w^{c_{i_j}})} &= \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial (\delta\theta_w^{c_{i_j}})} = J_j \cdot J_{\theta_{i_j}} = H_{\theta_{i_j}} \\
\frac{\partial h}{\partial f_j} &= \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial f_j} = J_j \cdot J_{f_j} = H_{f_j} \\
\frac{\partial h}{\partial {}^{i}n_j} &= \frac{\partial {}^{i}z_j}{\partial {}^{i}n_j} = V_j
\end{aligned}
\quad (6.7)
$$

The demonstration of the expressions of these jacobians can be found in [51], while the results are:

$$ J_j = \frac{\partial {}^{i}z_j}{\partial {}^{c_i}p_j} = \begin{bmatrix} 1/{}^{c_i}\hat{z}_j & 0 & -{}^{c_i}\hat{x}_j/{}^{c_i}\hat{z}_j^2 \\ 0 & 1/{}^{c_i}\hat{z}_j & -{}^{c_i}\hat{y}_j/{}^{c_i}\hat{z}_j^2 \end{bmatrix} \quad (6.8) $$

$$ J_{p_i} = \frac{\partial {}^{c_i}p_j}{\partial p_w^{c_i}} = -C(\hat{q}_w^{c_i}) \quad (6.9) $$

$$ J_{p_{i_j}} = \frac{\partial {}^{c_i}p_j}{\partial p_w^{c_{i_j}}} = C(\hat{q}_w^{c_i}) \quad (6.10) $$

$$ J_{\theta_i} = \frac{\partial {}^{c_i}p_j}{\partial (\delta\theta_w^{c_i})} = \lfloor {}^{c_i}\hat{p}_j \times \rfloor \quad (6.11) $$

$$ J_{\theta_{i_j}} = \frac{\partial {}^{c_i}p_j}{\partial (\delta\theta_w^{c_{i_j}})} = -\frac{1}{\hat{\rho}_j} C(\hat{q}_w^{c_i}) C(\hat{q}_w^{c_{i_j}})^T \left\lfloor \begin{bmatrix} \hat{\alpha}_j \\ \hat{\beta}_j \\ 1 \end{bmatrix} \times \right\rfloor \quad (6.12) $$

$$ J_{f_j} = \frac{\partial {}^{c_i}p_j}{\partial f_j} = \frac{1}{\hat{\rho}_j} C(\hat{q}_w^{c_i}) C(\hat{q}_w^{c_{i_j}})^T \begin{bmatrix} 1 & 0 & -\hat{\alpha}_j/\hat{\rho}_j \\ 0 & 1 & -\hat{\beta}_j/\hat{\rho}_j \\ 0 & 0 & -1/\hat{\rho}_j \end{bmatrix} \quad (6.13) $$

$$ V_j = \frac{\partial {}^{i}z_j}{\partial {}^{i}n_j} = I_2 $$

The model (6.6) can be rewritten in the form of (4.10):

$$^i\delta z_j \approx \frac{\partial h}{\partial(\delta x)}\delta x + \frac{\partial h}{\partial {}^i n_j}{}^i n_j$$

$$^i\delta z_j \approx H_j \delta x + V_j {}^i n_j$$

or, given that V_j is the identity matrix:

$$^i\delta z_j \approx H_j \delta x + {}^i n_j$$

by assembling the jacobians relative to the different error states ($\delta p_w^{c_i}$, $\delta q_w^{c_i}$, $\delta p_w^{c_{i_j}}$, $\delta q_w^{c_{i_j}}$, δf_j) into a single jacobian H_j, which multiplies the whole error state δx. Assuming M camera poses and N features in the state:

$$H_j = \frac{\partial h}{\partial(\delta x)} = J_j \cdot \frac{\partial {}^{c_i}p_j}{\partial(\delta x)} = \tag{6.14}$$

$$J_j \cdot [\ 0_{2\times 15},\ J_{p_i},\ 0_{2\times 3(i_j-2)},\ J_{p_{i_j}},\ 0_{2\times 3(M-i_j)},\ J_{\theta_i},$$

$$0_{2\times 3(i_j-2)},\ J_{\theta_{i_j}},\ 0_{2\times 3(M-i_j)}, 0_{2\times 3(j-1)},\ J_{f_j},\ 0_{2\times 3(N-j)}\ ]$$

Batch update

The equations described so far are relative to a filter update regarding a single observation of a single feature j. However, N observations of N features can be processed as a batch with measurement model:

$$^i\delta z \approx H\delta x + {}^i n \tag{6.15}$$

by *stacking* the residuals, jacobians and noises of every feature j:

$$^i\delta z = \begin{bmatrix} \delta z_1 \\ \delta z_2 \\ \vdots \\ \delta z_N \end{bmatrix} \qquad H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_N \end{bmatrix} \quad ^i n = \begin{bmatrix} ^i n_1 \\ ^i n_2 \\ \vdots \\ ^i n_N \end{bmatrix}$$

The covariance of the whole noise vector is now:

$$R = E\left[{}^i n {}^i n^T\right] = \sigma_V^2 \cdot I_{2N} \tag{6.16}$$

since the noise variance is the same for every feature.

6.2.2 Method

In practice, to construct an update:

1. For each feature in the state:

(a) Estimate the feature position ${}^{c_i}\hat{p}_j$ in camera frame with (6.5)

(b) Estimate the observation with (6.2)

(c) Compute the residual ${}^{i}\delta z_j$ between the real observation of the feature and the estimation of in the previous step

(d) Calculate the jacobian H_j in (6.14)

(e) Create R_j with (6.3)

(f) Check if the observation is an outlier with a validation gate based on the Mahalanobis distance, as described in section 6.5, with degrees of freedom equal to the size of the residual (2 DOF)

2. Stack the residuals and jacobians of all features that passed the outlier check into ${}^{i}\delta z$ and H of model (6.15)

3. Create the noise covariance matrix R with (6.16)

4. Perform the update of the EKF according to the correction step equations described in 4.2, but applying the Joseph form explained in section 6.6.

6.3 MSCKF update

While in SLAM the measurement prediction is computed starting from the feature parameters which are the state (6.5), in MSCKF all measurements from several frames are processed together to obtain a first estimate of the feature position ${}^{w}p_j$, which is then used to evaluate the residual in each frame.

6.3.1 Feature estimation

To obtain a feature estimate, the Gauss-Newton least-squares minimization algorithm is employed, starting from an initial value found by triangulation with the techniques of Direct Linear Transformation.

Direct Linear Transformation

In a pinhole camera model with camera matrix P, the world point X gets projected onto the image plane as point x. Using homogeneous coordinates for both points, the relationship is:

$$x = PX$$

$$P = K^T C(q_w^{c_i}) \begin{bmatrix} 1 & 0 & 0 & -x_w^{c_i} \\ 0 & 1 & 0 & -y_w^{c_i} \\ 0 & 0 & 1 & -z_w^{c_i} \end{bmatrix}$$

with K being the camera intrinisc matrix. It follows that:

$$x \times PX = 0$$

or:

$$\lfloor x \times \rfloor PX = 0$$

The matrix that results from the cross product $x \times P$ does not have full row rank, so the previous system is equivalent to

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \lfloor x \times \rfloor PX = 0$$

or:

$$A_i X = 0$$

$$A_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \lfloor x \times \rfloor P$$

The same world point X viewed by two distinct camera poses projects two points x_1 and x_2 on the image plane. If the measurement were perfect with zero noise, the rays passing through x_1 and x_2 would intersect at X, which could be found by solving:

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} X = 0 \qquad \rightarrow \qquad AX = 0$$

In real scenarios the rays do not intersect, so the system does not have an exact solution, but the point X can be found as the linear least square solution by calculating the Singular Valued Decomposition of A:

$$U \Sigma V^* = A$$

and taking the column of V that corresponds to the smallest singular value. Then, to obtain Cartesian coordinates:

$$x = \frac{1}{X_4} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

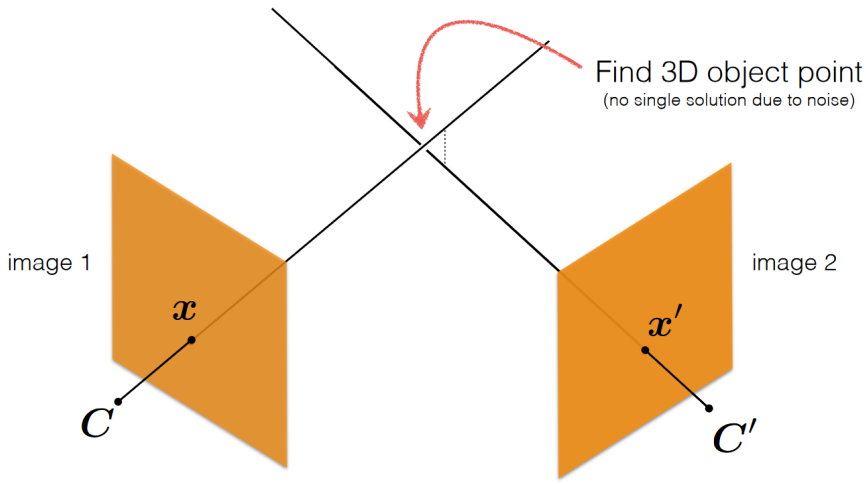


Figure 6.1: Noisy measurements of the same point in two camera poses. Credits: [64]

Gauss-Newton method

While DLT solves the **linear** least-squares problem, Gauss-Newton method is employed to solve the **non-linear** problem by iteration, starting from the linear solution. For better convergence properties [43], the DLT result is translated to inverse-depth parameters f anchored to the last camera pose, then:

1. For each camera frame i in the track of a feature j:
   (a) Estimate the feature position ${}^{c_i}\hat{p}_j$ with (6.5)
   (b) Estimate the observation with (6.2)
   (c) Calculate the residual ${}^{i}\delta z_j$ between the real observation of the feature and the estimation in the previous step
   (d) Calculate the jacobian of the residual with respect to the inverse-depth parameters $f_j = [\alpha\ \beta\ \rho]^T$, which is identical to (6.7):

$$ {}^{i}J_{r,j} = \frac{\partial^{i}\delta z_j}{\partial f_j} = {}^{i}J_j \cdot {}^{i}J_{f_j} $$

2. Stack the residuals and the jacobians of all m observations:

$$ \delta z_j = \begin{bmatrix} {}^{1}\delta z_j \\ {}^{2}\delta z_j \\ \vdots \\ {}^{m}\delta z_j \end{bmatrix} \qquad J_{gn,j} = \begin{bmatrix} {}^{1}J_{r,j} \\ {}^{2}J_{r,j} \\ \vdots \\ {}^{m}J_{r,j} \end{bmatrix} $$

3. Apply the update to f_j:

$$ f_{j,\text{new}} \quad \leftarrow \quad f_j - \left(J_{r,j}^T \cdot J_{r,j}\right)^{-1} J_{r,j}^T \cdot \delta z_j $$

4. If either the difference between the norm of the residuals at the previous step and the current step is smaller than a specific value or the number of iterations reached the maximum, stop the algorithm and return $f_{j,\text{new}}$, otherwise repeat.

6.3.2 Model

The observation model is a function of type:

$$ {}^{i}z_j = h\left(p_w^{c_i}, q_w^{c_i}, {}^{w}p_j, {}^{i}n_j\right) $$

which can be linearized to express the residuals as:

$$ {}^{i}\delta z_j \approx \frac{\partial h}{\partial p_w^{c_i}}\delta p_w^{c_i} + \frac{\partial h}{\partial(\delta\theta_w^{c_i})}\delta\theta_w^{c_i} + \frac{\partial h}{\partial {}^{w}p_j}\delta {}^{w}p_j + \frac{\partial h}{\partial {}^{i}n_j}{}^{i}n_j \tag{6.17} $$

$$\frac{\partial h}{\partial p_w^{c_i}} = \frac{\partial {}^i z_j}{\partial {}^{c_i} p_j} \cdot \frac{\partial {}^{c_i} p_j}{\partial p_w^{c_i}} = {}^i J_j \cdot {}^i J_{p_i} = {}^i H_{p_i}$$

$$\frac{\partial h}{\partial (\delta\theta_w^{c_i})} = \frac{\partial {}^i z_j}{\partial {}^{c_i} p_j} \cdot \frac{\partial {}^{c_i} p_j}{\partial (\delta\theta_w^{c_i})} = {}^i J_j \cdot {}^i J_{\theta_i} = {}^i H_{\theta_i}$$

$$\frac{\partial h}{\partial {}^w p_j} = \frac{\partial {}^i z_j}{\partial {}^{c_i} p_j} \cdot \frac{\partial {}^{c_i} p_j}{\partial {}^w p_j} = {}^i J_j \cdot {}^i J_{p_j} = {}^i H_{p_j}$$

$$\frac{\partial h}{\partial {}^i n_j} = \frac{\partial {}^i z_j}{\partial {}^i n_j} = {}^i V_j$$

The demonstration of the expressions of these jacobians can be found in [51]. J_j, J_{p_i} and J_{θ_i} are the same of (6.8), (6.9) and (6.11):

$${}^i J_j = \frac{\partial {}^i z_j}{\partial {}^{c_i} p_j} = \begin{bmatrix} 1/{}^{c_i}\hat{z}_j & 0 & -{}^{c_i}\hat{x}_j/{}^{c_i}\hat{z}_j^2 \\ 0 & 1/{}^{c_i}\hat{z}_j & -{}^{c_i}\hat{y}_j/{}^{c_i}\hat{z}_j^2 \end{bmatrix}$$

$${}^i J_{p_i} = \frac{\partial {}^{c_i} p_j}{\partial p_w^{c_i}} = -C(\hat{q}_w^{c_i})$$

$${}^i J_{\theta_i} = \frac{\partial {}^{c_i} p_j}{\partial (\delta\theta_w^{c_i})} = \lfloor {}^{c_i}\hat{p}_j \times \rfloor$$

$${}^i J_{p_j} = \frac{\partial {}^{c_i} p_j}{\partial {}^w p_j} = C(\hat{q}_w^{c_i})$$

$${}^i V_j = \frac{\partial z_j}{\partial {}^i n_j} = I_2$$

The model (6.17) **cannot** be expressed in the form of (4.10), because ${}^w p_j$ is not part of the state. First, by assembling the jacobians ${}^i H_{p_i}$, ${}^i H_{\theta_i}$, into a single jacobian H_j which multiplies the whole error state δx, the model is rewritten as:

$${}^i \delta z_j \approx \frac{\partial h}{\partial (\delta x)} \delta x + \frac{\partial h}{\partial {}^w p_j} \delta {}^w p_j + \frac{\partial h}{\partial {}^i n_j} {}^i n_j$$

$${}^i \delta z_j \approx {}^i H_j \delta x + {}^i H_{p_j} \delta {}^w p_j + {}^i V_j {}^i n_j$$

or, given that ${}^i V_j$ is the identity matrix:

$${}^i \delta z_j \approx {}^i H_j \delta x + {}^i H_{p_j} \delta {}^w p_j + {}^i n_j \tag{6.18}$$

where, if there are M camera poses and N features in the state:

$${}^i H_j = \frac{\partial h}{\partial (\delta x)} = {}^i J_j \cdot \frac{\partial {}^{c_i} p_j}{\partial \delta x} = {}^i J_j \cdot [\ 0_{2\times 15},\ {}^i J_{p_i},\ 0_{2\times 3(M-1)},\ {}^i J_{\theta_i},\ 0_{2\times 3(M-1+N)}\ ]$$

So far, the model is valid for a single observation in frame i of a single feature j, but all residuals, matrices and noise terms of a single MSCKF track can be stacked to create the model relative to all observations of a single feature j in m_j frames:

$$\delta z_j \approx H_j \delta x + H_{p_j} {}^w \delta p_j + n_j \tag{6.19}$$

with:

$$\delta z_j = \begin{bmatrix} {}^1\delta z_j \\ {}^2\delta z_j \\ \vdots \\ {}^{m_j}\delta z_j \end{bmatrix} \qquad H_j = \begin{bmatrix} {}^1H_j \\ {}^2H_j \\ \vdots \\ {}^{m}H_j \end{bmatrix} \qquad H_{p_j} = \begin{bmatrix} {}^1H_{p_j} \\ {}^2H_{p_j} \\ \vdots \\ {}^{m_j}H_{p_j} \end{bmatrix} \qquad n_j = \begin{bmatrix} {}^1n_j \\ {}^2n_j \\ \vdots \\ {}^{m_j}n_j \end{bmatrix}$$

Since the residuals are not expressed as a function of only error states and noise, an EKF update cannot be applied as is. To get to a model in a usable form, both sides of (6.19) are multiplied by a basis of the left null-space (or cokernel) of H_{p_j}, indicated as A_j. By definition:

$$A_j^T H_{p_j} = 0$$

so

$$A_j^T \delta z_j \approx A_j^T H_j \delta x + A_j^T n_j$$

which can be written as

$$\delta z_{0,j} \approx H_{0,j} \delta x + n_{0,j} \tag{6.20}$$

where:

$$\delta z_{0,j} = A_j^T \delta z_j \qquad H_{0,j} = A_j^T H_j \qquad n_{0,j} = A_j^T n_j$$

The dependency from ${}^w\delta p_j$ has disappeared, so this new model can be used to update the EKF.

H_{p_j} has size $2m_j \times 3$ and has full rank 3, so A_j has rank $2m_j - 3$ and size $2m_j \times (2m_j - 3)$ per rank nullity theorem. This means by the multiplication with A_j^T the number of rows of $\delta z_{0,j}$, $H_{0,j}$ and $n_{0,j}$ has been reduced to $2m_j - 3$. To find A_j, it's possible to perform a QR decomposition on H_{p_j} and then take the last $(2m_j - 3)$ columns of Q.

$$H_{p_j} = QR$$

$$Q = \begin{bmatrix} U & A^T \end{bmatrix}$$

$$A \in \mathbb{R}^{2m_j \times (2m_j - 3)}$$

The covariance of noise vector n_j is:

$$R_j = E\left[n_j n_j^T\right] = \sigma_V^2 \cdot I_{m_j}$$

and of noise vector $n_{0,j}$ is:

$$R_{0,j} = E\left[A_j^T n_j n_j^T A_j\right] = A_j^T E\left[n_j n_j^T\right] A_j = A_j^T R_j A_j \tag{6.21}$$

but since A_j is unitary:

$$R_{0,j} = \sigma_V^2 \cdot I_{2m_j - 3}$$

Batch update

N_t features can be initialized together as a batch by stacking residuals, jacobians and noise terms of each track j into:

$$\delta z_0 = \begin{bmatrix} \delta z_{0,1} \\ \delta z_{0,2} \\ \vdots \\ \delta z_{0,N_t} \end{bmatrix} \qquad H_0 = \begin{bmatrix} H_{0,1} \\ H_{0,2} \\ \vdots \\ H_{0,N_t} \end{bmatrix} \qquad n = \begin{bmatrix} n_{0,1} \\ n_{0,2} \\ \vdots \\ n_{0,N_t} \end{bmatrix}$$

and using the model:

$$\delta z_0 \approx H_0 \delta x + n \tag{6.22}$$

The covariance of the noise vector n is:

$$R = E\left[nn^T\right] = \sigma_V^2 \cdot I_{[2(m_1+\cdots+m_{N_t})-3N_t]} \tag{6.23}$$

6.3.3 Method

1. For each feature track j:
   (a) Obtain a linear least square estimate of the feature position by the DLT method using two observations in two poses, for example the first and the last in the track.
   (b) Obtain a non-linear least square estimate of the feature by the Gauss-Newton method using all the observations
   (c) For each camera frame and observation i in the track:
      i. Estimate the feature position ${}^{c_i}\hat{p}_j$ with (6.5)
      ii. Estimate the observation with (6.2)
      iii. Compute the residual ${}^{i}\delta z_j$ between the real observation of the feature and the estimation in the previous step
      iv. Calculate the jacobians ${}^{i}H_j$ and ${}^{i}H_{p_j}$ in (6.18)
   (d) Stack residuals and jacobians ${}^{i}\delta z_j$, ${}^{i}H_j$, ${}^{i}H_{p_j}$ of all observations i into δz_j, H_j, H_{p_j} of model (6.19)
   (e) Find the left nullspace A_j of H_{p_j}
   (f) Calculate the new residuals and jacobians $\delta z_{0,j}$ and $H_{0,j}$ of model (6.20)
   (g) Create the noise covariance matrix $R_{0,j}$ with (6.21)
   (h) Check if the feature estimate is an outlier with a validation gate based on the Mahalanobis distance, as described in section 6.5, with degrees of freedom equal to the size of the residual ($(2 * m_j - 3)$ DOF)
2. Stack the residuals and jacobians of all features that passed the outlier check into δz_0 and H_0 of model (6.22)
3. Create the noise covariance matrix R with (6.23)
4. Perform the update of the EKF according to the correction step equations described in 4.2, but applying the Joseph form explained in section 6.6.

6.4 Range Update

The measurement model considers a rangefinder which is projecting a ray to a triangular surface in front of the camera, defined by three SLAM features as vertices. The terrain is assumed locally flat along this surface, but the different triangular facets between all features in view form a 3D mesh which approximates the shape of the terrain. There is no assumption of a globally flat terrain like in the case of Mars Helicopter Ingenuity's navigation system [41].

6.4.1 Model

The ray coming from the rangefinder is assumed to originate from the optical center of the camera and is oriented along the unit vector u_r parallel to the camera optical axis. It intersect in p_I the triangle with features p_{j_1}, p_{j_2} e p_{j_3} as vertices. In inverse depth parametrization, the features are f_{j_1}, f_{j_2} and f_{j_3} and anchored to camera poses i_1, i_2 and i_3. A vector normal to the triangle facet can be found by taking the cross product of two vectors between any two pairs of vertices in Cartesian coordinates:

$$ {}^{w}n = ({}^{w}p_{j_1} - {}^{w}p_{j_2}) \times ({}^{w}p_{j_3} - {}^{w}p_{j_2}) \tag{6.24} $$

This normal vector is not generally a unit vector. The distance between the intersection point p_I and the camera can be formulated using any pair out of the tree vertices:

$$ z_r = \frac{({}^{w}p_I - p_w^{c_i})^T \cdot {}^{w}n}{{}^{w}u_r \cdot {}^{w}n} = \frac{({}^{w}p_I - {}^{w}p_{j_2} + {}^{w}p_{j_2} - p_w^{c_i})^T \cdot {}^{w}n}{{}^{w}u_r \cdot {}^{w}n} $$

$$ \downarrow $$

$$ z_r = \frac{({}^{w}p_{j_2} - p_w^{c_i})^T \cdot {}^{w}n}{{}^{w}u_r \cdot {}^{w}n} \tag{6.25} $$

The position of features ${}^{w}p_j$ can be retrieved from the state thanks to (6.4), which means the model is in the form

$$ z_r = h\,(x, n) $$

as a function of states and noise only, and can be employed for an EKF update. This model can be linearized to write the residuals as a linear combination of the error states and noise:

$$ {}^{i}\delta z_r \approx \frac{\partial h}{\partial p_w^{c_i}}\delta p_w^{c_i} + \frac{\partial h}{\partial(\delta\theta_w^{c_i})}\delta\theta_w^{c_i} + \frac{\partial h}{\partial p_w^{c_{i_1}}}\delta p_w^{c_{i_1}} + \frac{\partial h}{\partial p_w^{c_{i_2}}}\delta p_w^{c_{i_2}} + \frac{\partial h}{\partial p_w^{c_{i_3}}}\delta p_w^{c_{i_3}} + \frac{\partial h}{\partial(\delta\theta_w^{c_{i_1}})}\delta\theta_w^{c_{i_1}} + \tag{6.26} $$

$$ + \frac{\partial h}{\partial(\delta\theta_w^{c_{i_2}})}\delta\theta_w^{c_{i_2}} + \frac{\partial h}{\partial(\delta\theta_w^{c_{i_3}})}\delta\theta_w^{c_{i_3}} + \frac{\partial h}{\partial f_{j_1}}\delta f_{j_1} + \frac{\partial h}{\partial f_{j_2}}\delta f_{j_2} + \frac{\partial h}{\partial f_{j_3}}\delta f_{j_3} + \frac{\partial h}{\partial {}^{i}n_r}{}^{i}n_r $$

where:

$$\frac{\partial h}{\partial p_w^{c_i}} = \frac{\partial^i z_r}{\partial p_w^{c_i}} = H_{p_i}$$

$$\frac{\partial h}{\partial (\delta\theta_w^{c_i})} = \frac{\partial^i z_r}{\partial (\delta\theta_w^{c_i})} = H_{\theta_i}$$

$$\frac{\partial h}{\partial p_w^{c_{i_1}}} = \frac{\partial^i z_r}{\partial p_w^{c_{i_1}}} = \frac{\partial^i z_r}{\partial p_w^{c_{j_1}}} = J_{p_{j_1}} = H_{p_{i_1}}$$

$$\frac{\partial h}{\partial p_w^{c_{i_2}}} = \frac{\partial^i z_r}{\partial p_w^{c_{i_2}}} = \frac{\partial^i z_r}{\partial p_w^{c_{j_2}}} = J_{p_{j_2}} = H_{p_{i_2}}$$

$$\frac{\partial h}{\partial p_w^{c_{i_3}}} = \frac{\partial^i z_r}{\partial p_w^{c_{i_3}}} = \frac{\partial^i z_r}{\partial p_w^{c_{j_3}}} = J_{p_{j_3}} = H_{p_{i_3}}$$

$$\frac{\partial h}{\partial (\delta\theta_w^{c_{i_1}})} = \frac{\partial^i z_r}{\partial^w p_{j_1}} \cdot \frac{\partial^w p_{j_1}}{\partial (\delta\theta_w^{c_{i_1}})} = J_{p_{j_1}} \cdot J_{\theta_{i_1}} = H_{\theta_{i_1}}$$

$$\frac{\partial h}{\partial (\delta\theta_w^{c_{i_2}})} = \frac{\partial^i z_r}{\partial^w p_{j_2}} \cdot \frac{\partial^w p_{j_2}}{\partial (\delta\theta_w^{c_{i_2}})} = J_{p_{j_2}} \cdot J_{\theta_{i_2}} = H_{\theta_{i_2}}$$

$$\frac{\partial h}{\partial (\delta\theta_w^{c_{i_3}})} = \frac{\partial^i z_r}{\partial^w p_{j_3}} \cdot \frac{\partial^w p_{j_3}}{\partial (\delta\theta_w^{c_{i_3}})} = J_{p_{j_3}} \cdot J_{\theta_{i_3}} = H_{\theta_{i_3}}$$

$$\frac{\partial h}{\partial f_{j_1}} = \frac{\partial^i z_r}{\partial^w p_{j_1}} \cdot \frac{\partial^w p_{j_1}}{\partial f_{j_1}} = J_{p_{j_1}} \cdot J_{f_{j_1}} = H_{f_{j_1}}$$

$$\frac{\partial h}{\partial f_{j_2}} = \frac{\partial^i z_r}{\partial^w p_{j_2}} \cdot \frac{\partial^w p_{j_2}}{\partial f_{j_2}} = J_{p_{j_2}} \cdot J_{f_{j_2}} = H_{f_{j_2}}$$

$$\frac{\partial h}{\partial f_{j_3}} = \frac{\partial^i z_r}{\partial^w p_{j_3}} \cdot \frac{\partial^w p_{j_3}}{\partial f_{j_3}} = J_{p_{j_3}} \cdot J_{f_{j_3}} = H_{f_{j_3}}$$

$$\frac{\partial h}{\partial^i n_r} = \frac{\partial^i z_r}{\partial^i n_r} = V$$

The demonstration of the expressions of these jacobians can be found in [51], while the results are:

$$H_{p_i} = \frac{\partial^i z_r}{\partial p_w^{c_i}} = -\frac{1}{\hat{b}}\hat{n}^T$$

$$H_{\theta_i} = \frac{\partial^i z_r}{\partial (\delta\theta_w^{c_i})} = \frac{\hat{a}}{\hat{b}^2}\left(\lfloor {}^{c_i}u_r \times \rfloor C(\hat{q}_w^{c_i}) {}^w\hat{n}\right)^T$$

$$J_{p_{j_1}} = \frac{\partial^i z_r}{\partial^w p_{j_1}} = \frac{1}{\hat{b}}\left(\lfloor ({}^w\hat{p}_{j_3} - {}^w\hat{p}_{j_2}) \times \rfloor \left({}^w\hat{p}_{j_2} - {}^w\hat{p}_{I_i}\right)\right)^T$$

$$J_{p_{j_2}} = \frac{\partial^i z_r}{\partial^w p_{j_2}} = \frac{1}{\hat{b}}\left({}^w\hat{n} + \lfloor ({}^w\hat{p}_{j_1} - {}^w\hat{p}_{j_3}) \times \rfloor \left({}^w\hat{p}_{j_2} - {}^w\hat{p}_{I_i}\right)\right)^T$$

$$J_{p_{j_3}} = \frac{\partial^i z_r}{\partial {}^w p_{j_3}} = \frac{1}{\hat{b}} \left(\lfloor ({}^w\hat{p}_{j_2} - {}^w\hat{p}_{j_1}) \times \rfloor \, ({}^w\hat{p}_{j_2} - {}^w\hat{p}_{I_i}) \right)^T$$

$$J_{\theta_{i_1}} = \frac{\partial^w p_{j_1}}{\partial(\delta\theta_w^{c_{i_1}})} = -\frac{1}{\hat{\rho}_{j_1}} C(\hat{q}_w^{c_{i_1}})^T \left\lfloor \begin{bmatrix} \hat{\alpha}_{j_1} \\ \hat{\beta}_{j_1} \\ 1 \end{bmatrix} \times \right\rfloor$$

$$J_{\theta_{i_2}} = \frac{\partial^w p_{j_2}}{\partial(\delta\theta_w^{c_{i_2}})} = -\frac{1}{\hat{\rho}_{j_2}} C(\hat{q}_w^{c_{i_2}})^T \left\lfloor \begin{bmatrix} \hat{\alpha}_{j_2} \\ \hat{\beta}_{j_2} \\ 1 \end{bmatrix} \times \right\rfloor$$

$$J_{\theta_{i_3}} = \frac{\partial^w p_{j_3}}{\partial(\delta\theta_w^{c_{i_3}})} = -\frac{1}{\hat{\rho}_{j_3}} C(\hat{q}_w^{c_{i_3}})^T \left\lfloor \begin{bmatrix} \hat{\alpha}_{j_3} \\ \hat{\beta}_{j_3} \\ 1 \end{bmatrix} \times \right\rfloor$$

$$J_{f_{j_1}} = \frac{\partial^w p_{j_1}}{\partial f_{j_1}} = \frac{1}{\hat{\rho}_{j_1}} C(\hat{q}_w^{c_{i_1}})^T \begin{bmatrix} 1 & 0 & -\hat{\alpha}_{j_1}/\hat{\rho}_{j_1} \\ 0 & 1 & -\hat{\beta}_{j_1}/\hat{\rho}_{j_1} \\ 0 & 0 & -1/\hat{\rho}_{j_1} \end{bmatrix}$$

$$J_{f_{j_2}} = \frac{\partial^w p_{j_2}}{\partial f_{j_2}} = \frac{1}{\hat{\rho}_{j_2}} C(\hat{q}_w^{c_{i_2}})^T \begin{bmatrix} 1 & 0 & -\hat{\alpha}_{j_2}/\hat{\rho}_{j_2} \\ 0 & 1 & -\hat{\beta}_{j_2}/\hat{\rho}_{j_2} \\ 0 & 0 & -1/\hat{\rho}_{j_2} \end{bmatrix}$$

$$J_{f_{j_3}} = \frac{\partial^w p_{j_3}}{\partial f_{j_3}} = \frac{1}{\hat{\rho}_{j_3}} C(\hat{q}_w^{c_{i_3}})^T \begin{bmatrix} 1 & 0 & -\hat{\alpha}_{j_3}/\hat{\rho}_{j_3} \\ 0 & 1 & -\hat{\beta}_{j_3}/\hat{\rho}_{j_3} \\ 0 & 0 & -1/\hat{\rho}_{j_3} \end{bmatrix}$$

$$V = \frac{\partial^i z_r}{\partial^i n_r} = 1$$

$$a = ({}^w p_{j_2} - p_w^{c_i})^T \cdot {}^w n$$

$$b = {}^w u_r \cdot {}^w n$$

$${}^i z_r = \frac{a}{b}$$

$${}^w\hat{p}_{I_i} = \hat{q}_w^{c_i} + {}^w u_r \cdot {}^i z_r$$

With the same logic that has been used for SLAM and MSCKF, the jacobians relative to the different error states can be assembled in the corresponding positions into a single jacobian H_j which multiplies the whole error state vector. This way it is possible to rewrite the model (6.26) in the form of (4.10):

$${}^i\delta z_r \approx \frac{\partial h}{\partial(\delta x)}\delta x + \frac{\partial h}{\partial^i n_r} {}^i n_r$$

$${}^i\delta z_r \approx H\delta x + V {}^i n_r$$

or, given that V is equal to 1:

$${}^i\delta z_r \approx H\delta x + V {}^i n_r \tag{6.27}$$

In this case, the H matrix is just a row vector and the noise covariance matrix is just a scalar:

$$^{i}R_r = E\left[{}^{i}n_r\,{}^{i}n_r{}^{T}\right] = \sigma_R^2$$

6.4.2 Method

1. Perform a Delaunay triangulation in image space over the features in the state (SLAM features). Delaunay maximizes the smallest angle of all possible triangulations [65], resulting in less thin triangles, which do not provide strong planar constraints.

2. Find the triangle that intersect with the rangefinder ray. Since the ray is aligned with the camera optical axis and is assumed to originate from the optical center, the triangle is the one that contains the principal point in image space. Find the features at the vertices j_1, j_2 and j_3 anchored to camera poses i_1, i_2 and i_3

3. Predict the measurement with (6.24) and (6.25)

4. Compute the residual $^{i}\delta z_r$ between the real measurement of the rangefinder and the prediction in the previous step

5. Calculate the jacobians in (6.26)

6. Assemble the jacobians into the matrix H, according to the model (6.27) and to definition of the error states in section 5.2.2.

7. Create the matrix R, in fact just a scalar equal to σ_R^2

8. Check if the observation is an outlier with a validation gate based on the Mahalanobis distance, as described in section 6.5, with degrees of freedom equal to the size of the residual (1 DOF).

9. If the outlier check is passed, perform the update of the EKF according to the correction step equations described in 4.2, applying the Joseph form in 6.6.

6.5 Outlier rejection

Measurements that are too much in disagreement with the prediction get rejected and not employed for an update. To determine whether a sensor measurement is an outlier, it is passed through a validation gate based on the Mahalanobis distance. Generally, the Mahalanobis distance is a metric of the separation between a point x and a distribution with mean μ covariance S, taking into account the correlations that exist in the data:

$$d = \sqrt{(x-\mu)^T S^{-1} (x-\mu)}$$

Applied to our case, it is used to compute the distance between the sensor measurement and the predicted value:

$$d^2 = (z - \hat{z})^T S^{-1} (z - \hat{z}) = \delta z^T S^{-1} \delta z$$

Where S is the innovation (residual) covariance from 4.2:

$$S = HP^- H^T + VRV^T$$

Since S is the covariance of the whole term δz, then d is the distance between the two distributions of z and $\hat{z}$ taking into account both covariance matrices.
The distance is compared to a threshold and the test is passed if:

$$d^2 < \gamma (k, \alpha)$$

The threshold $\gamma (k, \alpha)$ is the inverse cumulative distribution function, evaluated at probability α, of the χ^2 distribution with k degrees of freedom. χ^2 is defined as the distribution of the sum of the squares of k independent standard normal random variables:

$$\sum_{i=1}^{k} x_i^2 \sim \chi_k^2 \tag{6.28}$$

By a change of variables, it is shown that The Mahalanobis distance is itself a sum of squares of randomly distributed values:

$$S = CC^T$$

$$y = C^{-1} (x - \mu) \sim \mathcal{N}(0, I)$$

$$\downarrow$$

$$d^2 = y^T I^{-1} y = \sum_{i=1}^{k} y_i$$

This means that the validation gate for the test (6.28) is a region of acceptance such that $100(1 - \alpha)\%$ of true measurements are rejected.

6.6 Joseph form and symmetry

The covariance matrix P in the EKF must remain symmetric and positive definite. The time update equation adds two positive definite quantities together, ensuring the result is positive definite:

$$P_t^- = FP_{t-1}F^T + WQW^T$$

Theoretically, also the measurement update equation preserves a positive definite result, but due to the subtraction operation it is possible for round-off errors to result in a non-positive-definite solution.

$$P_t = (I - K_t H) P_t^-$$

The Joseph form of the measurement update equation makes the subtraction occurs in the squared terms, guaranteeing a positive definite result

$$P_t = (I - K_t H)\, P_t^- \,(I - K_t H)^T + K_t V R V^T K_t^T$$

To ensure symmetry it is possible to simply substitute the covariance at any moment with:

$$P_{\text{new}} \leftarrow \frac{P + P^T}{2}$$

If the matrix ever becomes negative definite, it can be substituted with the nearest positive definite by first obtaining the Singular Value Decomposition:

$$P = USV^T$$

then:

$$J = VSV^T$$

$$P_{\text{new}} \leftarrow \frac{P + P^T + J + J^T}{4}$$

which also ensures symmetry

Chapter 7

State and filter management

7.1 Inserting a pose

The state includes a sliding window of M camera poses. Each time a new camera frame gets acquired, the oldest pose is removed from the state, the remaining ones get shifted one slot and the new camera pose is added to the free first slot. For example, if a state vector with a sliding window of length M is written like:

$$x_{t_i} = \begin{bmatrix} x_{I,t_i} \\ x_{S,t_i} \\ x_{F,t_i} \end{bmatrix} \qquad \text{where:} \qquad x_S = \begin{bmatrix} p_w^{c_M} \\ \vdots \\ p_w^1 \\ q_w^{c_M} \\ \vdots \\ q_w^{c_1} \end{bmatrix}$$

when camera frame $M+1$ is inserted the sliding window states get updated as:

$$x_{S,\text{new}} \leftarrow \begin{bmatrix} p_w^{c_{M+1}} \\ \vdots \\ p_w^{c_2} \\ q_w^{c_{M+1}} \\ \vdots \\ q_w^{c_2} \end{bmatrix}$$

The new pose of the camera is calculated knowing the relative pose of the camera with respect to the IMU (p_i^c, q_i^c), which is a fixed value that depends on the locations of the sensors.

$$p_w^{c_{M+1}} = \hat{p}_w^i + C(\hat{q}_w^i)^{Ti} p_i^c$$

$$q_w^{c_{M+1}} = \hat{q}_w^i \otimes q_i^c$$

The covariance matrix must be updated to reflect the changes. From [51], with M as the length of the sliding window and N as the number of features in the state,

the new error states for the new camera pose are:

$$\delta p_w^{c_{M+1}} = \delta p_w^i - C(\hat{q}_w^i)^T \lfloor p_i^c \times \rfloor \delta\theta_w^i$$

$$\delta\theta_w^{c_{M+1}} = C(q_i^c)\delta\theta_w^i$$

$$\downarrow$$

$$\delta p_w^{c_{M+1}} = \left[I_3,\ 0_{3\times 3},\ -C(\hat{q}_w^i)^T \lfloor p_i^c \times \rfloor,\ 0_{3\times 6(M+1)+3N}\right] \delta x$$

$$\delta\theta_w^{c_{M+1}} = \left[0_{3\times 6},\ C(q_i^c),\ 0_{3\times 6(M+1)+3N}\right] \delta x$$

Considering this change and also the translation of the error states of camera poses from 2 to M by three rows down to the next slots, the updated error states after inserting a pose are:

$$\delta x_{\text{new}} = J\delta x$$

where:

$$J = \begin{bmatrix} I_3 & & & & & & & & \\ & I_3 & & & & & & & \\ & & I_3 & & & & & & \\ & & & I_6 & & & & & \\ I_3 & & -C(\hat{q}_w^i)^T \lfloor p_i^c \times \rfloor & & & 0_{3\times 3} & & & \\ & & & & I_{3(M-1)} & & & & \\ & & C(q_i^c) & & & & & 0_{3\times 3} & \\ & & & & & & I_{3(M-1)} & & \\ & & & & & & & & I_{3N} \end{bmatrix}$$

By definition (4.2), the covariance matrix is:

$$P = E\left[\delta x \delta x^T\right]$$

so its update becomes:

$$P_{\text{new}} = E\left[\delta x_{\text{new}} \delta x_{\text{new}}^T\right] = E\left[J\delta x \delta x^T J^T\right] = JE\left[\delta x \delta x^T\right] J^T = JPJ^T$$

$$P_{\text{new}} \leftarrow JPJ^T$$

Initial frames

For the initial frames M, when there are less poses in the state vector than the maximum size of the sliding window M, the new pose $m+1$ is inserted by augmenting the sliding window states:

$$x_{S,\text{aug}} \leftarrow \begin{bmatrix} p_w^{c_{m+1}} \\ p_w^{c_m} \\ \vdots \\ p_w^{c_1} \\ q_w^{c_{m+1}} \\ q_w^{c_m} \\ \vdots \\ q_w^{c_1} \end{bmatrix}$$

and the covariance:

$$P_{\text{aug}} \leftarrow JPJ^T$$

where:

$$J = \begin{bmatrix} I_3 & & & & & & & \\ & I_3 & & & & & & \\ & & I_3 & & & & & \\ & & & I_6 & & & & \\ I_3 & & -C(\hat{q}_w^i)^T \lfloor p_i^c \times \rfloor & & & & & \\ & & & & I_{3m} & & & \\ & & C(q_i^c) & & & & & \\ & & & & & I_{3m} & & \\ & & & & & & & I_{3N} \end{bmatrix}$$

7.1.1 Feature reparametrization

As the oldest camera pose in the sliding window gets removed from the state to insert a new pose, all features that were anchored to it need to be reparametrized for a different anchor pose and the covariance matrix needs to be updated accordingly. To maximize the time between two reparametrizations for a given feature, the new anchor is chosen as the newest pose.

For a single feature j, anchored to camera pose i_j, that needs to be reparametrized to the latest pose i, the feature position in the latest frame ${}^{c_i}p_j$ is obtained from (6.4):

$${}^{c_i}p_j = C(q_w^{c_i}) \left(p_w^{c_{i_j}} + \frac{1}{\rho_j} C(q_w^{c_{i_j}})^T \begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} - p_w^{c_i} \right)$$

Then, as can be deduced from the geomtry in figure 5.1, the new inverse depth parameters are just:

$$f_j \leftarrow f'_j = \begin{bmatrix} {}^{c_i}x_j / {}_{c_i}z_j \\ {}^{c_i}y_j / {}_{c_i}z_j \\ 1 / {}_{c_i}z_j \end{bmatrix} \tag{7.1}$$

The equations above can be linearized to obtain the updated error states for j:

$$\delta f'_j \approx \frac{\partial f'_j}{\partial {}^{c_i}p_j} \cdot \frac{\partial {}^{c_i}p_j}{\partial(\delta x)} \cdot \delta x = J_{0,j} \cdot J_{1,j} \cdot \delta x$$

where:

$$J_{0,j} = \frac{\partial f'_j}{\partial {}^{c_i}p_j} = {}^i\hat{\rho}_j \begin{bmatrix} 1 & 0 & {}^i\hat{\alpha}_j \\ 0 & 1 & {}^i\hat{\beta}_j \\ 0 & 0 & {}^i\hat{\rho}_j \end{bmatrix} \tag{7.2}$$

$$J_{1,j} = \frac{\partial {}^{c_i}p_j}{\partial(\delta x)} = [\ 0_{3\times 15},\ J_{p_i},\ 0_{3\times 3(M-2)},\ J_{p_{i_j}},\ J_{\theta_i}, \tag{7.3}$$

$$0_{2\times 3(M-2)},\ J_{\theta_{i_j}},\ 0_{2\times 3(j-1)},\ J_{f_j},\ 0_{2\times 3(N-j)}\ ]$$

J_{p_i}, $J_{p_{i_j}}$, J_{θ_i}, $J_{\theta_{i_j}}$ and J_{f_j} are the same as (6.9), (6.10), (6.11), (6.12) and (6.13). The error states update then is:

$$\delta x' \approx J_j \cdot \delta x$$

where J_j is a identity matrix of size $15+6M+3N$, whose rows from $16+6M+3(j-1)$ to $16+6M+3j$ have been substituted by the product $J_{0,j}J_{1,j}$

$$J_j = \begin{bmatrix} \begin{bmatrix} I_{15+6M+3(j-1)} & 0_{[15+6M+3(j-1)]\times 3(N-j+1)} \end{bmatrix} \\ J_{0,j} \cdot J_{1,j} \\ \begin{bmatrix} 0_{3(N-j)\times(15+6M+3j)} & I_{3(N-j)} \end{bmatrix} \end{bmatrix} \tag{7.4}$$

By definition (4.2) of the covariance:

$$P = E\left[\delta x \delta x^T\right]$$

so the updated matrix is:

$$P' = E\left[\delta x' \delta x'^T\right] = E\left[J_j \delta x \delta x^T J_j^T\right] = J_j E\left[\delta x \delta x^T\right] J_j^T = J_j P J_j^T$$

$$P \leftarrow P' = J_j P J_j^T$$

The demonstrations of the expressions of the jacobians can be found in [51].

Batch reparametrization

If there are multiple features that need to be reparametrized, their states are updated individually with (7.1). However, a single matrix J is built by initializing an identity matrix of size $15+6M+3N$ and substituting, for each feature j, the rows from $16+6M+3(j-1)$ to $16+6M+3j$ with the product $J_{0,j}J_{1,j}$ of the jacobians (7.2) and (7.3), as in shown in (7.4) for a single feature. Then, the covariance is updated with a single operation:

$$P \leftarrow P' = JPJ^T$$

7.2 Feature initialization

To employ SLAM, the state needs to be augmented with feature states. Usually this happens after MSCKF measurements, when the conditions explained in section 6.1 are met, meaning that a previous estimate of the feature depth with respect to the camera is available. If such MSCKF measurements are not obtainable, for lack of a minimum translation between frames, or because it's the first frame, or if there's a need to immediately provide constraints to the system after a loss of tracking, then the initialization of features occurs with unknown-depth.
Initializing a feature does not only entail augmenting the state with feature parameters, but also augmenting the covariance matrix with appropriate values of uncertainty and cross correlations.

7.2.1 Unknown-depth initialization

During initialization, the 95% acceptance region for the depth from the camera is assumed to be between infinite and a parameter d_{min}. Even though it extends to infinity, thanks to inverse-depth parameterization [43], the region is defined by finite bounds:

$$0 \leq \rho \leq \frac{1}{d_{min}}$$

The expected value of the inverse depth is the average:

$$\hat{\rho} = \frac{1}{2d_{min}}$$

with standard deviation:

$$\sigma_\rho = \frac{1}{4d_{min}}$$

The other two parameters α and β, as it is apparent in figure 5.1, are simply the observation in the image plane of the anchor frame i_j:

$$\begin{bmatrix} \hat{\alpha}_j \\ \hat{\beta}_j \end{bmatrix} = {}^{i_j}z_j$$

with the standard deviation σ_V of the vision front.
The state is then augmented as:

$$x \leftarrow x' = \begin{bmatrix} \hat{x} \\ \hat{\alpha}_j \\ \hat{\beta}_j \\ \hat{\rho}_j \end{bmatrix} = \begin{bmatrix} \hat{x} \\ {}^{i_j}z_j \\ 1/2d_{min} \end{bmatrix}$$

And the covariance:

$$P \leftarrow P' = \begin{bmatrix} P & & & \\ & \sigma_V^2 & & \\ & & \sigma_V^2 & \\ & & & (1/4d_{min})^2 \end{bmatrix}$$

7.2.2 MSCKF initialization

When a feature track exceeds the length of the sliding window M, it is initialized into the state to be used for SLAM. [53] showed this is computationally advantageous if M is chosen appropriately and proposed a method to augment the state and covariance matrix with the new feature. The idea is to include it into the state vector, anchored to the last camera pose, and adding corresponding infinite variances in the covariance matrix, then using all m observations simultaneously to perform an EKF update. After the new pose has already been inserted:

$$x_{\text{aug}} \leftarrow \begin{bmatrix} x \\ f_j \end{bmatrix}$$

$$x_{\text{aug}} \leftarrow \begin{bmatrix} P & \\ & \mu I_3 \end{bmatrix}$$

The model is:

$$\delta z_j \approx H_j \delta x + H_{f_j} \delta f_j + V_j n_j$$

where V_j is the identity matrix I_2, while H_j is the matrix of size $2m_j \times 3$ made by stacking the jacobians ${}^iH_j$ for every observation i:

$${}^iH_j = \tag{7.5}$$

$$J_j \cdot \begin{bmatrix} 0_{2\times 15} & J_{p_i} & 0_{2\times 3(i_j-2)} & J_{p_{i_j}} & 0_{2\times 3(M-i_j)} & J_{\theta_i} & 0_{2\times 3(i_j-2)} & J_{\theta_{i_j}} & 0_{2\times 3(M-i_j)} & 0_{2\times 3N} \end{bmatrix}$$

where J_j, J_{p_i}, $J_{p_{i_j}}$, J_{θ_i} and $J_{\theta_{i_j}}$ are (6.8), (6.9), (6.10), (6.11) and (6.12), into:

$$H_j = \begin{bmatrix} {}^1H_j \\ {}^2H_j \\ \vdots \\ {}^{m_j}H_j \end{bmatrix}$$

H_{f_j} is the matrix of size $2m_j \times 3$ made by stacking the jacobians ${}^iH_{f_j}$ from (6.7) for every observation i into:

$$H_{f_j} = \begin{bmatrix} {}^1H_{p_j} \\ {}^2H_{p_j} \\ \vdots \\ {}^{m_j}H_{p_j} \end{bmatrix}$$

The new feature is included in the augmented state so the model can be used for an EKF update by rewriting it as:

$$\delta z_j \approx \begin{bmatrix} H_j & H_{f_j} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta f_j \end{bmatrix} + n_j$$

$$\delta z_j \approx \begin{bmatrix} H_j & H_{f_j} \end{bmatrix} \delta x_{\text{aug}} + n_j \tag{7.6}$$

Because of the fact that the variances μ of the feature parameters are infinite, f_j could in theory be chosen randomly. However, since it is used to calculate the jacobians, a good estimate is needed and is obtained by triangulation from the observations in different frames, employing the methods in 6.3.1. It remains the problem of choosing a value for μ. Simply picking a large value would lead to numerical problems, so instead the limit of the EKF update is computed for $\mu \to \infty$. The complete demonstration is explained in [63], what follows is only the results which are needed for a practical implementation.

U and A are two matrices which are basis, respectively, for column space and left nullspace of H_{f_j}. They can be calculated from the QR decomposition:

$$QR = H_{f_j}$$

$$Q = \begin{bmatrix} U & A \end{bmatrix}$$

$$U \in \mathbb{R}^{2m_j \times \text{rank}(H_{f_j})}$$

$$A \in \mathbb{R}^{2m_j \times (2m_j - \text{rank}(H_{f_j}))}$$

Model (7.6) is left multiplied on both sides by $W = \begin{bmatrix} A & U \end{bmatrix}$:

$$W^T \delta z_j \approx W^T \begin{bmatrix} H_j & H_{f_j} \end{bmatrix} \delta x_{\text{aug}} + W^T n_j$$

$$\begin{bmatrix} A^T \\ U^T \end{bmatrix} \delta z_j \approx \begin{bmatrix} A^T \\ U^T \end{bmatrix} \begin{bmatrix} H_j & H_{f_j} \end{bmatrix} \delta x_{\text{aug}} + \begin{bmatrix} A^T \\ U^T \end{bmatrix} n_j$$

$$\begin{bmatrix} A^T \delta z_j \\ U^T \delta z_j \end{bmatrix} = \begin{bmatrix} A^T H_j & A^T H_{f_j} \\ U^T H_j & U^T H_{f_j} \end{bmatrix} \delta x_{\text{aug}} + n_{c,j}$$

$$\begin{bmatrix} \delta z_{0,j} \\ \delta z_{1,j} \end{bmatrix} = \begin{bmatrix} H_{0,j} & \\ H_{1,j} & H_{2,j} \end{bmatrix} \delta x_{\text{aug}} + n_{c,j} \tag{7.7}$$

The covariance matrix of noise vector n_j is:

$$R_j = E\left[n_j n_j^T\right] = \sigma_V^2 \cdot I_{m_j}$$

and of noise vector $n_{c,j}$ is:

$$R_{c,j} = E\left[W^T n_j n_j^T W\right] = W^T E\left[n_j n_j^T\right] W = W^T R_j W$$

but since W is unitary:

$$R_{c,j} = \sigma_V^2 \cdot I_{2m_j} = \begin{bmatrix} R_{0,j} & \\ & R_{1,j} \end{bmatrix} = \begin{bmatrix} \sigma_V^2 \cdot I_{2m_j - \text{rank}(H_{f_j})} & \\ & \sigma_V^2 \cdot I_{\text{rank}(H_{f_j})} \end{bmatrix} = \sigma_V^2 \cdot I_{2m_j} \tag{7.8}$$

The state and covariance are updated as:

$$x_{\text{aug}} \leftarrow \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \tag{7.9}$$

$$P_{\text{aug}} \leftarrow \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix} \tag{7.10}$$

Where x_0 and P_{11} are the state and covariance that result from the standard MSCKF update:

$$x_0 = x + P H_{0,j}^T \left(H_{0,j} P H_{0,j}^T + R_{0,j}\right)^{-1} \delta z_{0,j}$$

$$P_{11} = P - P H_{0,j}^T \left(H_{0,j} P H_{0,j}^T + R_{0,j}\right)^{-1} H_{0,j} P$$

and the rest is:

$$x_1 = -H_{2,j}^{-1} H_{1,j} P H_{0,j}^T \left(H_{0,j} P H_{0,j}^T + R_{0,j}\right)^{-1} \delta z_{0,j} + H_{2,j}^{-1} \delta z_{1,j}$$

$$P_{22} = \left(H_{2,j}^{-1} H_{1,j}\right) P_{11} \left(H_{2,j}^{-1} H_{1,j}\right)^T + R_{1,j}$$

$$P_{21} = -H_{2,j}^{-1} H_{1,j} P_{11}$$

$$P_{12} = P_{21}^T$$

Batch initialization

N_t features can be initialized together as a batch by stacking residuals, jacobians and noise terms of each track j into:

$$\delta z_0 = \begin{bmatrix} \delta z_{0,1} \\ \delta z_{0,2} \\ \vdots \\ \delta z_{0,N_t} \end{bmatrix} \qquad \delta z_1 = \begin{bmatrix} \delta z_{1,1} \\ \delta z_{1,2} \\ \vdots \\ \delta z_{1,N_t} \end{bmatrix} \qquad n_c = \begin{bmatrix} n_{c,1} \\ n_{c,2} \\ \vdots \\ n_{c,N_t} \end{bmatrix}$$

$$H_0 = \begin{bmatrix} H_{0,1} \\ H_{0,2} \\ \vdots \\ H_{0,N_t} \end{bmatrix} \qquad H_1 = \begin{bmatrix} H_{1,1} \\ H_{1,2} \\ \vdots \\ H_{1,N_t} \end{bmatrix} \qquad H_2 = \begin{bmatrix} H_{2,1} \\ H_{2,2} \\ \vdots \\ H_{2,N_t} \end{bmatrix}$$

and using the model:

$$\begin{bmatrix} \delta z_0 \\ \delta z_1 \end{bmatrix} = \begin{bmatrix} H_0 & \\ H_1 & H_2 \end{bmatrix} \delta x_{\text{aug}} + n_c \tag{7.11}$$

The covariance of the noise vector n_c is:

$$R_c = E\left[n_c n_c^T\right] = \begin{bmatrix} R_0 & \\ & R_1 \end{bmatrix} = \sigma_V^2 \cdot I_{2m_1+\cdots+2m_{N_t}}$$

where:

$$R_0 = \begin{bmatrix} R_{0,1} & & \\ & \ddots & \\ & & R_{0,N_t} \end{bmatrix} \qquad R_1 = \begin{bmatrix} R_{1,1} & & \\ & \ddots & \\ & & R_{1,N_t} \end{bmatrix} \tag{7.12}$$

7.2.3 Method

1. For each feature track j:
   (a) Obtain a linear least square estimate of the feature position by the DLT method using two observations in two poses, for example the first and the last in the track.
   (b) Obtain a non-linear least square estimate of the feature by the Gauss-Newton method using all the observations
   (c) For each camera frame and observation i in the track:
      i. Estimate the feature position ${}^{c_i}\hat{p}_j$ with (6.5)
      ii. Estimate the observation with (6.2)
      iii. Compute the residual ${}^i\delta z_j$ between the real observation of the feature and the estimation in the previous step
      iv. Calculate the jacobians ${}^iH_j$ and ${}^iH_{f_j}$ with (7.5) and (6.7)
   (d) Stack residuals and jacobians ${}^i\delta z_j$, ${}^iH_j$, ${}^iH_{f_j}$ of all observations i into δz_j, H_j, H_{f_j} of model (6.19)

(e) Find the left nullspace A_j and column space U_j of H_{f_j}

(f) Calculate the residuals $\delta z_{0,j}$, $\delta z_{1,j}$ and jacobians $H_{0,j}$, $H_{1,j}$, $H_{2,j}$ of model (7.7)

(g) Create the noise covariance matrices $R_{0,j}$ and $R_{1,j}$ with (7.12)

(h) Check if the feature estimate with residuals is an outlier with a validation gate based on the Mahalanobis distance with residuals $\delta z_{0,j}$, as described in section 6.5, with degrees of freedom equal to the size of the residual ($(2 * m_j - \text{rank}(H_{f_j}))$ DOF)

2. Stack the residuals and jacobians of all features that passed the outlier check into δz_0, δz_1, H_0, H_1 and H_2 of model (7.11)

3. Create the noise covariance matrices $R_{0,j}$ and $R_{1,j}$ with (7.12)

4. Perform the updates and augment state and covariance as in (7.9) (7.10)

7.3 Filter loop

The main loop simply waits until either new IMU measurement or camera and rangefinder measurements are available.

- If IMU measurements are available, the time update step in chapter 5.3 is executed
- If new camera and rangefinder measurement are available (always together), then:

  1. The vision management and tracking routine is executed, which updates the feature tracks (SLAM and MSCKF), perform redetection of features if the conditions are met and removes SLAM features from the state whose tracks have been lost
  2. MSCKF update is performed with the MSCKF tracks that have been lost. This is done before the new camera pose is inserted in the state since the oldest pose has to be used in MSCKF before being removed.
  3. The new camera pose is inserted in the state, removing the oldest one and possibly triggering reparametrization of some feature states
  4. SLAM update is executed with the new observations of SLAM features

7.3.1 Vision management and tracking

This routine has the task to process the image, match features between frames, redetect new features, remove features from states, and update feature *tracks* with the latest observations. A track is the memory of all observations of a feature in the image plane and is strictly needed for MSCKF, but is maintained also for

SLAM features for practical reasons and statistical purposes. The routine takes the new image, converts it to grayscale, undistorts it thanks to the camera instrinsics parameters obtained from calibration, then:

1. Removes SLAM features from the state that failed the outlier gate test more times than a chosen parameter
2. Executes the tracking and detection step, which matches feature from the previous frame to points in the current frame and, if conditions are met, redetects new features
3. Removes SLAM features that have been lost from state and covariance
4. Initializes features from MSCKF. It is possible to know that a track is longer than M only when it reaches length $M + 1$. If that happens, the previous M observations are used to initialize the feature in the state, while the last is used later for SLAM update.
5. Update tracks with the new observations
6. Return tracks (SLAM and MSCKF)

The tracker used in our implementation is a Kanade-Lucas-Tomasi (KLT) algorithm [66][67] to track point features, which is quite standard in computer vision problems and is fast. The feature type used was mostly corners, detected with Shi-Tomasi algorithm, a modification of Harris and Stephens corner detector where only the minimum eigenvalue is considered, hence the name *minimum eigenvalue features*.
The image is divided in a grid and the number of features in each *tile* are counted. A redetection is triggered when either:

- The total number of features in the whole image is lower than a parameter
- The number of empty tiles in the image is greater than a parameter

When one of the two conditions is met, a feature detector is run on the **previous** image, on each tile that contains a number of features n less than a set amount $n_{\max}$. New features detected in a tile are ordered from strongest to weakest and the distance of each one from every feature that was already existing in that image is calculated. If the distance from any feature is lower than a *neighborhood parameter*, the feature is discarded. This avoids redetecting features too close to others or even the same features that have been lost in the following (current) image. In each tile, the strongest $n_{\max} - n$ features are selected, providing a set of points that are reasonably distributed in the whole image. These features are then tracked from the previous image to the current image.
The reason for performing detection on the previous image is a smoother operation of the filter, since in the current frame there would be observations already available for EKF updates. During redetection, existing features are preserved. Features are detected only in tiles where $n < n_{max}$ to bound the computational complexity by

limiting the number of features. However, since features can move from a tile to another with camera movement, there might be tiles where $n > n_{max}$. This means the total number of features in the image after redetection could be greater than the number of tiles times n_{max}, and potentially limitless. If this happens, old features are removed one by one from the richest tile at the moment, until the total number of old and new features is the same as the maximum number of total features allowed (the number of tiles times n_{max}).

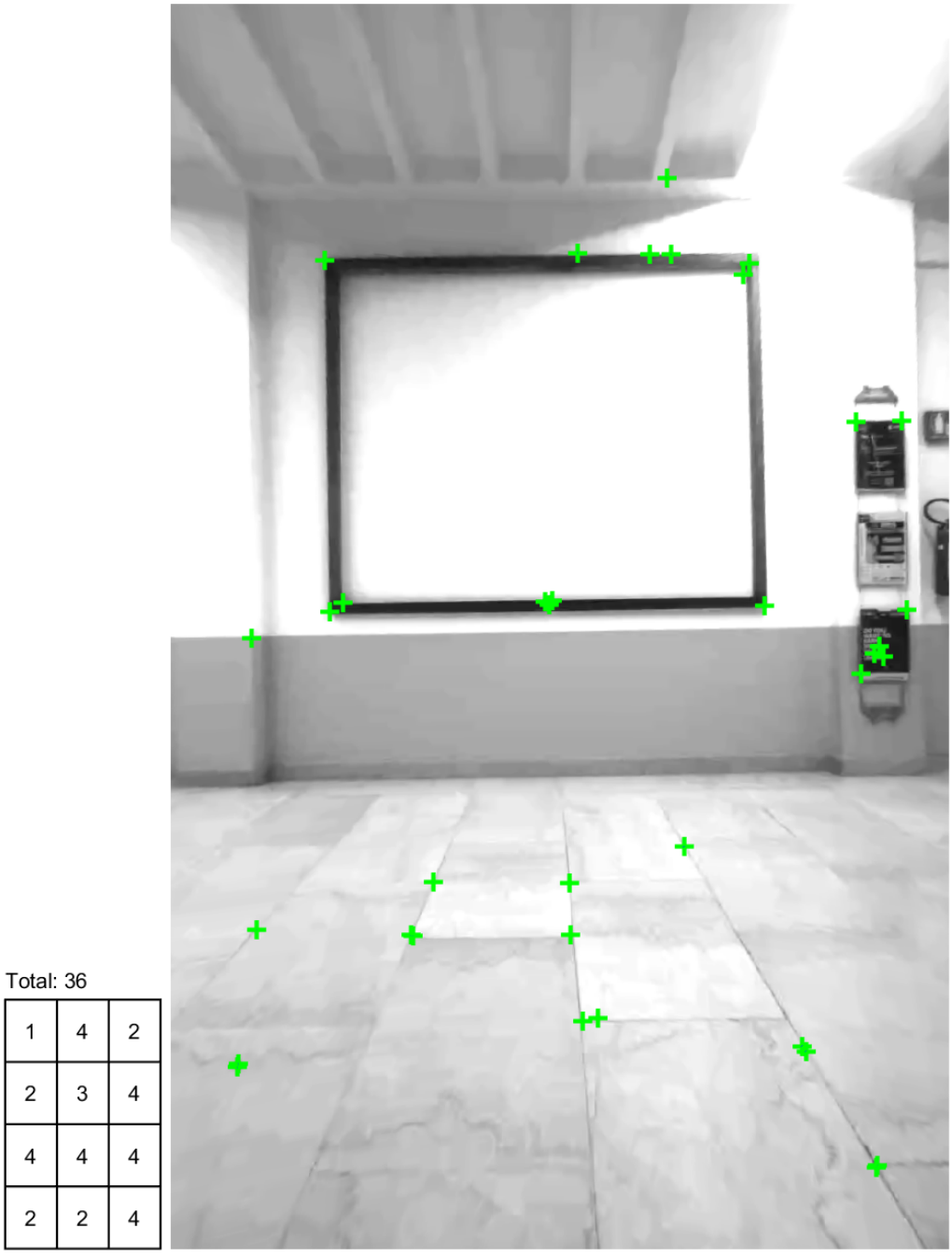


Figure 7.1: Example of corner features detected in an image. The number of features in each tile is indicated in the lower left

Chapter 8

Experimental tests and conclusions

The filter was implemented in MATLAB, with more than 2300 lines of code. The implementation is modular and uses objects, making it easy to change or extend its capabilities later. The kalman filter itself is very suitable for a modular approach since multiple updates with data from different sensors can be provided independently from each other.

The program takes as input a video file, text files for IMU and rangefinder data and text files for timestamps of camera and IMU. The rangefinder timestamps are the same as those of the camera. The parameters of the filters are loaded from a table and are:

- Starting position, velocity and orientation of the IMU
- Gravity vector
- Pose of the camera with respect to the IMU
- Intrinisc parameters of the camera
- Noise density and random walk of accelerometer and gyroscope
- Standard deviation of the visual front end noise in image plane
- Standard deviation of rangefinder noise
- Confidence of the χ^2 gate test for outlier rejection in SLAM, MSCKF and range update
- Number of consecutive times a SLAM feature is an outlier that make the tracker drop it
- Minimum depth for unknown depth initialization
- Minimum distance between rectified poses to allow triangulation for MSCKF

- Number of poses in the sliding window
- Number of horizontal and vertical tiles in an image
- Maximum number of features detected in a tile $n_{\max}$
- Number of total features under which redetection is triggered
- Number of empty tiles which triggers redetection
- Neighborhood parameter for the redetection
- Parameters of the KLT tracker, which are the number of pyramidal levels, maximum bidirectional error, block size and maximum iterations
- Parameters for feature detector, namely the minimum quality and filter size

The output is the estimation of position, velocity, and orientation of the IMU frame of reference. While running, a window shows the current and past position and the current orientation. The current camera frame is displayed overlaid with all features, the mesh of triangles between them, and the rangefinder measurements. The colors indicate the type of feature and whether it's valid or has been rejected as an outlier in the last update. Information on the side shows the number of features in each tile.

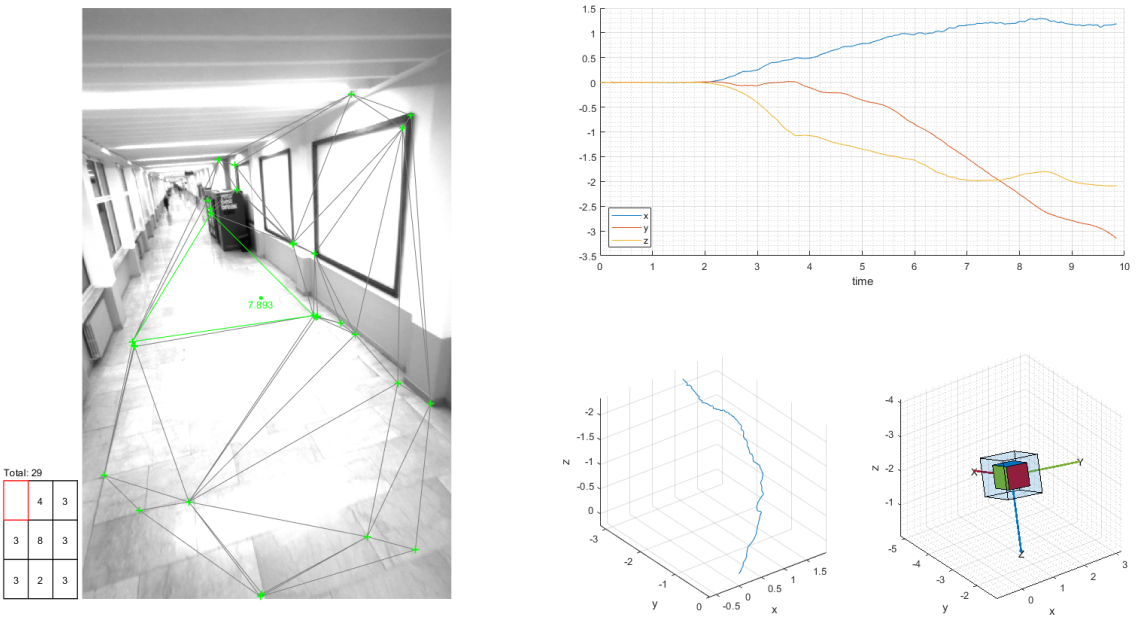


Figure 8.1: Still image from a run of the MATLAB program on a recorded dataset

8.1 Dataset acquisition

The program doesn't run in real time with live sensor data, but on a PC while being fed a datasets that has been recorded previously. To gather the data required as input for the MATLAB program, a platform was built which included the three sensors needed (camera, IMU and rangefinder) and a Raspberry Pi4b computer to record and store this data. The sensors were:

- Monochrome global shutter 1280×800 resolution OV9281 camera sensor on an Arducam B0224 board with 110° of horizontal FOV (after the image gets undistorted the FOV is reduced). In cameras that do not have a mechanical shutter, the function of the shutter is done by starting and ending *integration* of pixels electronically (when the charge is read from the pixels). Electronic shutter can be of two types: rolling shutter, which means pixel integration happens progressively in lines, or global shutter which means it happens for all pixels simultaneously. Rolling shutter, which is the most common in consumer electronics, distorts scenes or objects when they are moving. Obviously, for visual navigation this is undesirable because a correct measurement of the position of features in the image is critical, so a global shutter sensor is of utmost importance.
- SF22/C LiDAR rangefinder from Lightware. This is a single point LiDAR which measures the distance from a point in front of it with a range of over 100 m
- IMU: LSM9DS1 MEMS IMU in a Raspberry Sense HAT board. It includes accelerometer, gyroscope and magnetometer on a single chip (the magnetometer was not used).

The camera was calibrated to obtain its intriniscs parameters, which account for lens characteristics and distorsion, to allow the transformation a real camera image into an equivalent image taken by a perfect pinhole camera. The sensors and computer were mounted onto 3D printed PLA frame. Attention was given to place the rangefinder parallel and as close as possible to the camera, since in the filter model it is assumed to be coincident with the camera optical center. The position and orientation of the camera in the reference frame of the IMU chip was measured.

The program for acquisition running on the Raspberry Pi4b was written in Python and controlled with the joystick and leds of the Sense HAT board. The script communicated via I^2C protocol to IMU and rangefinder, while the camera was controlled through a library that uses MIPI. The script initializes the sensors with the necessary settings, then starts the video capture of the camera at 30 Hz. Every time a frame is captured, the rangefinder is polled to get the most recent measurement (internally, the rangefinder updates its measurement at 1000 Hz). Meanwhile, the IMU is polled as fast as possible, which is about 200 Hz.
The program writes IMU and rangefinder measurements and their timestamps in four differnt text files (the camera timestamps are the same of the rangefinder),

while camera frames are written on a video file. Because of bandwidth limitations in writing to a SD card, the video is saved in H264 compressed form.

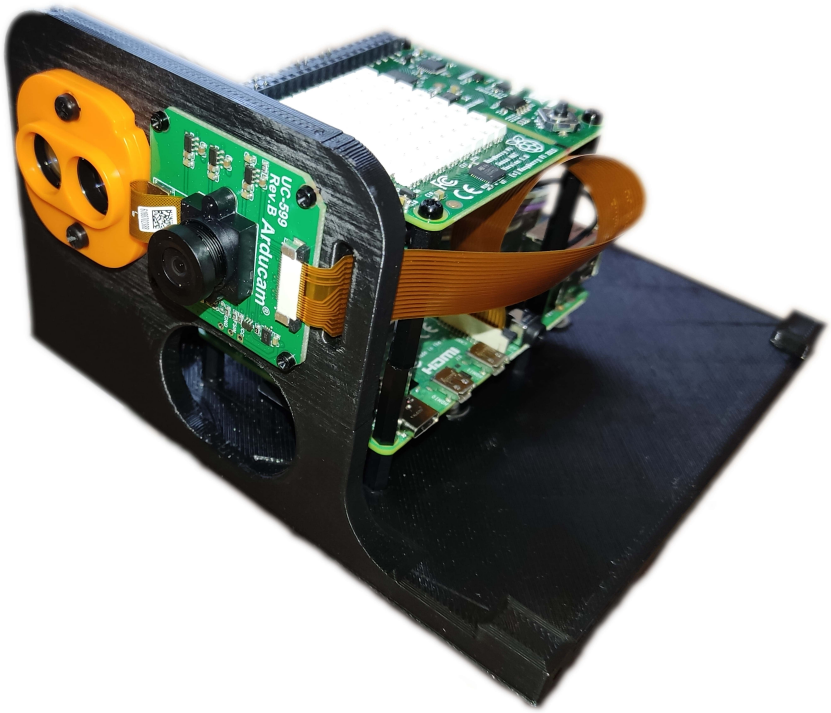


Figure 8.2: Sensor platform with computer for dataset recording

8.2 Results

The datasets were gathered by simply holding the platform by hand and walking in mostly indoor areas of Politecnico di Torino. No ground truth was recorded. The intent was just a qualitative test to assess the correct operation of the implementation and as a starting point for future developments of the navigation system. A proper tuning of the many parameters of the filter was not possible due to lack of the ground truth, instead parameters were chosen manually by running parameter sweeps in batches over datasets, until a configuration that was stable and had satisfactory result was found. For example, we'd expect the output of the filter from a dataset gathered on a long hallway to be a straight line, or the output from a dataset of the sensors spinning in one place to show no translation, etc. For the noise variances of the sensors, the values were taken from datasheets, then tuned manually to account for aditional errors and uncertainties in the platform or in the acquisition.

Some difficulties were encountered with the compressed video, to find the right balance of data bandwidth and quality. Compressed video has noise and artifacts that raw frames obviously don't have, but raw video couldn't be saved because of SD bandwidth. Another problem was the fact that the reflection of the rangefinder in-

frared laser on some surfaces was visible by the camera sensor. The vision processing algorithm sometimes detected features on the reflection point, creating the problem that these features would move with respect to the rest of the scene. However, the outlier rejection steps in the filter could exclude these features from the update if there were enough other features in the scene. Controlling the exposure of the camera during recording and without changing framerate was also found difficult though the available libraries from the manufacturer. Finally, time synchronization of sensors suffered from a lack of a common reference clock between IMU and camera due to limitations of the IMU chip.

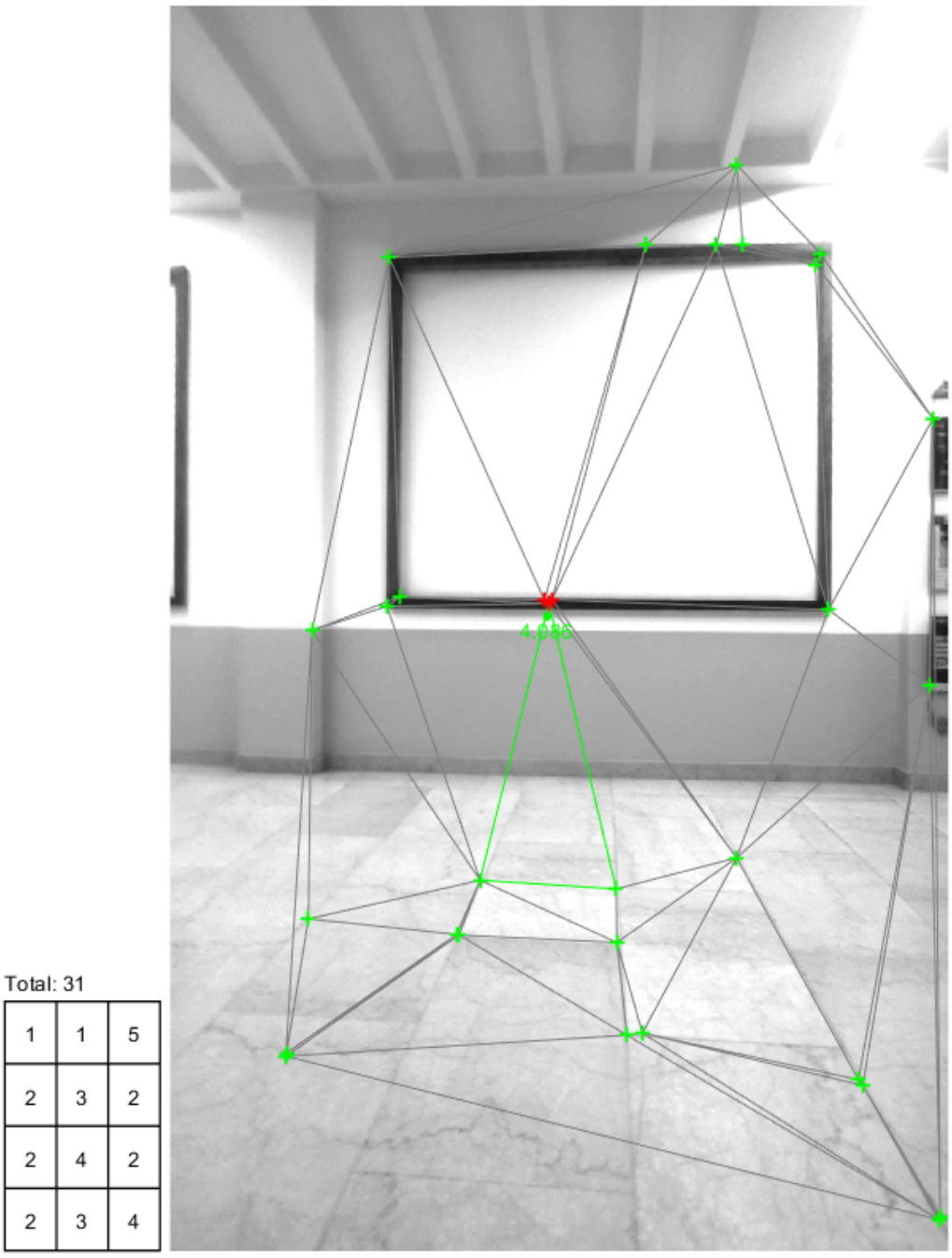


Figure 8.3: Features detected on the reflection of the laser beam moves with respect to the rest of the scene, but they get correctly identified as outliers (in red)

Notwithstanding all of that, the filter performed satisfactory most of the time and, if anything, these qualitative tests showed the capabilities of the navigation architecture, which worked even in unfavorable conditions. They also showed the clear influence of parameters on filter performance and, most evidently, on stability. This navigation architecture is promising for the LuNaDrone mission, for his ability to function without receiving external of any kind.

Currently, a new version of the sensor platform is being developed. It has an upgraded MEMS IMU, not only in terms of precision, but also in capabilities that allow faster measurement rate and better synchronization of timestamps with the camera. An SSD, with much better bandwidth, allows recording of raw image data, while a new lens for the camera doesn't let infrared light through to avoid seeing reflections of the rangefinder beam. The script for recording is being rewritten for better time synchronization between sensors and for parallelization using the multiple cores of the Raspberry Pi processor, dedicating, for example, an entire core for writing data on disk and another for listening to interrupts. A new library for controlling the camera sensor is available which makes possible to control and tune several parameters, allowing full control of the autoexposure algorithm. This new platform is encased in a 3D printed structure made to be mounted on a remote controlled drone that will fly on realistic trajectories closer to those of the LuNaDrone mission.

Figure 8.4: New sensor platform in development

Bibliography

[1] F. Horz. "Lava tubes - Potential shelters for habitats". In: *Lunar Bases and Space Activities of the 21st Century*. Ed. by W. W. Mendell. Jan. 1985, pp. 405–412.

[2] V. R. Oberbeck, W. L. Quaide, and R. Greeley. "On the Origin of Lunar Sinuous Rilles". In: *Modern Geology* 1 (Jan. 1969), pp. 75–80.

[3] Ronald Greeley. "Lava Tubes and Channels in the Lunar Marius Hills". In: *Moon* 3.3 (Dec. 1971), pp. 289–314. DOI: 10.1007/BF00561842.

[4] Cassandra R. Coombs and B. Ray Hawke. "A Search for Intact Lava Tubes on the Moon: Possible Lunar Base Habitats". In: *Lunar Bases and Space Activities of the 21st Century*. Ed. by Wendell W. Mendell et al. Sept. 1992, p. 219.

[5] A.s Arya et al. "Detection of potential site for future human habitability on the Moon using Chandrayaan-1 data". In: *Current science* 100 (Feb. 2011), pp. 524–529.

[6] Loic Chappaz et al. "Evidence of Large Empty Lava Tubes on the Moon using GRAIL Gravity: Evidence of Lunar Lava Tubes from GRAIL". In: *Geophysical Research Letters* 44 (Jan. 2017). DOI: 10.1002/2016GL071588.

[7] Tetsuya Kaku et al. "Detection of intact lava tubes at Marius Hills on the Moon by SELENE (Kaguya) Lunar Radar Sounder". In: *Geophysical Research Letters* 44 (Oct. 2017). DOI: 10.1002/2017gl074998.

[8] Audai Ed Theinat et al. "Lunar lava tubes: Morphology to structural stability". In: *Icarus* 338 (Oct. 2019), p. 113442. DOI: 10.1016/j.icarus.2019.113442.

[9] Junichi Haruyama et al. "Possible lunar lava tube skylight observed by SELENE cameras". In: *Geophysical Research Letters - GEOPHYS RES LETT* 36 (Nov. 2009). DOI: 10.1029/2009GL040635.

[10] J. Haruyama et al. "New Discoveries of Lunar Holes in Mare Tranquillitatis and Mare Ingenii". In: Mar. 2010.

[11] Robert Wagner and Mark Robinson. "Distribution, formation mechanisms, and significance of lunar pits". In: *Icarus* 237 (July 2014), pp. 52–60. DOI: 10.1016/j.icarus.2014.04.002.

[12] R. V. Wagner, A. Deran, and M. S. Robinson. "Habitability and Radiation Environment Within Lunar Pits". In: *Lunar and Planetary Science Conference*. Lunar and Planetary Science Conference. Mar. 2017, 1201, p. 1201.

[13] Y. Yokota et al. “Formation Scenario of Continuous Slopes Associated with Lunar Mare Pit/Hole Structures”. In: *Lunar and Planetary Science Conference*. Lunar and Planetary Science Conference. Mar. 2018, 1907, p. 1907.

[14] R. V. Wagner and M. S. Robinson. “What to Expect in Lunar Pits”. In: *Lunar and Planetary Science Conference*. Lunar and Planetary Science Conference. Mar. 2020, 1163, p. 1163.

[15] Chris Okubo and Stephen Martel. “Pit crater formation on Kilauea volcano, Hawaii”. In: *Journal of Volcanology and Geothermal Research* 86 (Nov. 1998), pp. 1–18. DOI: 10.1016/S0377-0273(98)00070-5.

[16] G. Cushing et al. “THEMIS observes possible cave skylights on Mars”. In: *Geophys. Res. Lett* 34 (Sept. 2007). DOI: 10.1029/2007GL030709.

[17] M Robinson et al. “Confirmation of sublunarean voids and thin layering in mare deposits”. In: *Planetary and Space Science* 69 (Apr. 2012). DOI: 10.1016/j.pss.2012.05.008.

[18] J. Ashley et al. “Geology of the King crater region: New insights into impact melt dynamics on the Moon”. In: *Journal of Geophysical Research* 117 (Nov. 2012), E00H29. DOI: 10.1029/2011JE003990.

[19] Paul Spudis, Patrick McGovern, and Walter Kiefer. “Large Shield Volcanoes on the Moon”. In: *Journal of Geophysical Research: Planets* 118 (May 2013). DOI: 10.1002/jgre.20059.

[20] Don Wilhelms, John McCauley, and Newell Trask. “The Geologic History of the Moon”. In: *U.S. Geol. Surv. Prof. Pap.* 1348 (Jan. 1987).

[21] Donald Gault. “Saturation and Equilibrium Conditions for Impact Cratering on the Lunar Surface: Criteria and Implications”. In: *Radio Sci.* 5 (Mar. 1970). DOI: 10.1029/RS005i002p00273.

[22] H. Hiesinger et al. “How old are young lunar craters?” In: *Journal of Geophysical Research* 117 (Feb. 2012). DOI: 10.1029/2011JE003935.

[23] R. Greeley, V. Oberbeck, and W. Quaide. “On the origin of lunar sinuous rilles”. In: 1 (Feb. 1969).

[24] Elena Martellato, B. Foing, and Johannes Benkhoff. “Numerical modelling of impact crater formation associated with isolated lunar skylight candidates on lava tubes”. In: *Planetary and Space Science* 86 (Sept. 2013), pp. 33–44. DOI: 10.1016/j.pss.2013.06.010.

[25] J. Ashley et al. “Voids in Lunar Mare and Impact Melt Deposits — A Commonsense Expedient to the Expansion of Humans into Space”. In: Oct. 2013. DOI: 10.13140/2.1.1231.5528.

[26] L. Kerber et al. “The Geologic Context of Major Lunar Mare Pits”. In: *Lunar and Planetary Science Conference*. Lunar and Planetary Science Conference. Mar. 2019, 3134, p. 3134.

[27] Angelo Pio Rossi et al. *DAEDALUS - Descent And Exploration in Deep Autonomy of Lava Underground Structures*. Mar. 2021. ISBN: 978-3-945459-33-1. DOI: 10.25972/OPUS-22791.

[28] Stefano Pescaglia et al. “LuNaDrone: nano drone for Lunar Exploration”. In: *IAC-22,A3,IP,61,x71889*. unpublished. IAF, 2022.

[29] Stefano Pescaglia. "Preliminary design of a Lunar Nano Drone for a mission of exploration of lava tubes on the Moon: study of the Mission Flight Profile and identification of the most suitable Energy Storage System". MA thesis. Politecnico di Torino, 2020. URL: https://webthesis.biblio.polito.it/17036/.

[30] Gabriele Podesta. "Lunar Nano Drone for a mission of exploration of lava tubes on the Moon: Propulsion System". MA thesis. Politecnico di Torino, 2020. URL: https://webthesis.biblio.polito.it/17038/.

[31] Shannah Withrow et al. "Mars Science Helicopter Conceptual Design". In: (Nov. 2020). DOI: 10.2514/6.2020-4029.

[32] Håvard Fjær Grip et al. "Flight control system for nasa's mars helicopter". In: *AIAA Scitech 2019 Forum*. American Institute of Aeronautics and Astronautics Inc, AIAA, 2019. ISBN: 9781624105784. DOI: 10.2514/6.2019-1289.

[33] Red Whittaker and Steven Huber. *Exploration of Planetary Skylights and Tunnels NASA Innovative Advanced Concepts (NIAC) Phase II*. 2012.

[34] William Whittaker, Uland Wong, and Steven Huber. *Exploration of Planetary Skylights and Tunnels*. Tech. rep. NASA, 2014.

[35] Issa Nesnas et al. "Moon Diver: A Discovery Mission Concept for Understanding the History of Secondary Crusts through the Exploration of a Lunar Mare Pit". In: Mar. 2019, pp. 1–23. DOI: 10.1109/AERO.2019.8741788.

[36] Pablo F. Miaja et al. "RoboCrane: A system for providing a power and a communication link between lunar surface and lunar caves for exploring robots". In: *Acta Astronautica* 192 (Mar. 2022), pp. 30–46. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2021.11.023.

[37] *ESA plans mission to explore Lunar caves*. URL: https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/ESA_plans_mission_to_explore_lunar_caves.

[38] *Peregrine Mission 1*. accessed: 2022-01-08. URL: https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=PEREGRN-1.

[39] Junichi Haruyama et al. "Exploration of Lunar Holes, Possible Skylights of Underlying Lava Tubes, by Smart Lander for Investigating Moon (SLIM)". In: *Transactions of the Japan Society for Aeronautical and Space Sciences, Aerospace Technology Japan* 10.ists28 (2012), pp. 7–10. ISSN: 1884-0485. DOI: 10.2322/tastj.10.pk_7.

[40] Baichuan Huang, Jun Zhao, and Jingbin Liu. *A Survey of Simultaneous Localization and Mapping with an Envision in 6G Wireless Networks*. 2019. DOI: 10.48550/ARXIV.1909.05214. URL: https://arxiv.org/abs/1909.05214.

[41] David Bayard et al. "Vision-Based Navigation for the NASA Mars Helicopter". In: Jan. 2019. DOI: 10.2514/6.2019-1411.

[42] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[43] J. Montiel, Javier Civera, and Andrew Davison. "Unified Inverse Depth Parametrization for Monocular SLAM". In: Aug. 2006. DOI: 10.15607/RSS.2006.II.011.

[44] Dimitrios Kottas, Kejian Wu, and Stergios Roumeliotis. "Detecting and dealing with hovering maneuvers in vision-aided inertial navigation systems". In: Nov. 2013, pp. 3172–3179. DOI: 10.1109/IROS.2013.6696807.

[45] Anastasios Mourikis and Stergios Roumeliotis. "A Multi-State Constraint Kalman Filter for Vision-Aided Inertial Navigation". In: vol. 22. May 2007, pp. 3565–3572. DOI: 10.1109/ROBOT.2007.364024.

[46] Edward Rosten and Tom Drummond. "Machine Learning for High-Speed Corner Detection". In: vol. 3951. July 2006. ISBN: 978-3-540-33832-1. DOI: 10.1007/11744023_34.

[47] Edward Rosten, Reid Porter, and Tom Drummond. "FASTER and better: A Machine Learning Approach to Corner Detection". In: *IEEE transactions on pattern analysis and machine intelligence* 32 (Jan. 2010), pp. 105–19. DOI: 10.1109/TPAMI.2008.275.

[48] Tarek Mouats et al. "Performance Evaluation of Feature Detectors and Descriptors Beyond the Visible". In: *Journal of Intelligent & Robotic Systems* 92 (Sept. 2018), pp. 1–31. DOI: 10.1007/s10846-017-0762-8.

[49] Miguel A San Martin et al. "A minimal state augmentation algorithm for vision-based navigation without using mapped landmarks". In: (2017).

[50] Jeff Delaune et al. "Extended Navigation Capabilities for a Future Mars Science Helicopter Concept". In: Mar. 2020, pp. 1–10. DOI: 10.1109/AERO47225.2020.9172289.

[51] Jeff Delaune, David Bayard, and Roland Brockers. *xVIO: A Range-Visual-Inertial Odometry Framework*. Tech. rep. Oct. 2020.

[52] Jeff Delaune, David Bayard, and Roland Brockers. "Range-Visual-Inertial Odometry: Scale Observability Without Excitation". In: *IEEE Robotics and Automation Letters* PP (Feb. 2021), pp. 1–1. DOI: 10.1109/LRA.2021.3058918.

[53] Mingyang Li and Anastasios Mourikis. "Optimization-Based Estimator Design for Vision-Aided Inertial Navigation". In: July 2012. DOI: 10.15607/RSS.2012.VIII.031.

[54] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[55] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[56] Ian Reid and Hilary Term. *Estimation II*. 2001. URL: https://www.robots.ox.ac.uk/~ian/Teaching/Estimation/LectureNotes2.pdf.

[57] J. Montiel, Javier Civera, and Andrew Davison. "Unified Inverse Depth Parametrization for Monocular SLAM". In: Aug. 2006. DOI: 10.15607/RSS.2006.II.011.

[58] Javier Civera, Andrew Davison, and J. Montiel. "Inverse Depth Parametrization for Monocular SLAM". In: *Robotics, IEEE Transactions on* 24 (Nov. 2008), pp. 932–945. DOI: 10.1109/TRO.2008.2003276.

[59] Nikolas Trawny and Stergios I Roumeliotis. "Indirect Kalman filter for 3D attitude estimation". In: *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep* 2 (2005), p. 2005.

[60] Stephan Weiss and Roland Siegwart. "Real-time metric state estimation for modular vision-inertial systems". In: June 2011, pp. 4531–4537. DOI: 10.1109/ICRA.2011.5979982.

[61] S Lynen et al. "A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation". In: *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*. 2013.

[62] ETH Zurich - Autonomous Sytems Lab. *MSF - Modular framework for multi sensor fusion based on an Extended Kalman Filter (EKF)*. https://github.com/ethz-asl/ethzasl_msf.git.

[63] Mingyang Li and Anastasios Mourikis. "Optimization-Based Estimator Design for Vision-Aided Inertial Navigation: Supplemental Materials". In: July 2012.

[64] Kris Kitani. *Triangulation*. 2017. URL: https://www.cs.cmu.edu/~16385/s17/Slides/11.4_Triangulation.pdf.

[65] Bernd Gärtner and Michael Hoffmann. "Computational geometry lecture notes HS 2013". In: *Dept. of Computer Science, ETH, Zürich, Switzerland* (2013).

[66] Bruce Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI)". In: vol. 81. Apr. 1981.

[67] Carlo Tomasi and Takeo Kanade. "Detection and tracking of point". In: *Int J Comput Vis* 9 (1991), pp. 137–154.