Politecnico di Milano

School of Industrial Engineering and Information
Department of Aerospace Science and Technology
Master of Science in Space Engineering

Visual Navigation for Autonomous Planetary Landing

Advisor: prof. Michèle Lavagna
Author: Luca Losi

Academic Year 2015/2016

"Never say never. Because limits, like fears, are often just an illusion"
M.J.

Abstract

Exploration of celestial bodies as the Moon, Mars, or even of smaller ones as comets and asteroids is becoming now days the next frontier of the space exploration. One of the most interesting and attractive purposes from the scientific point of view in this field, is the capability for a spacecraft to land on such bodies. In this context, it is often not possible to directly control the spacecraft from Earth during its descent, which is a delicate phase in which even a delay of a few seconds for a control command can result in a catastrophic event. For these reasons, the necessity for a completely autonomous landing system able to operate during the landing phases arises. This thesis presents the current development status of an autonomous navigation system designed from scratch, based on a monocular camera working in the visible spectrum. Vision based navigation works extracting salient features from the images and tracking them during motion. Position and orientation of the camera for each consecutive frame are retrieved with respect to the observed scenario thanks to this informations, and trajectory is reconstructed up to a scale factor. A description of all the building blocks constituting the system architecture is given, along with the different methods and approaches which have been tested in order to find the configuration which best fit for the application on a spacecraft landing scenario. Performance assessment of the navigation system using both synthetic video sequences of different landing trajectories on the Moon surface and a computer vision dataset benchmark is presented. To conclude, future work to be done to continue the development is listed in detail.

Keywords: Autonomous landing, Optical Navigation, SLAM, Visual Odometry

Sommario

L'esplorazione di corpi celesti come la Luna, Marte, ma anche di altri più piccoli come comete e asteroidi sta diventando oggigiorno la nuova frontiera dell'esplorazione spaziale. Una delle più interessanti e attraenti proposte dal punto di vista scientifico in questo campo, è la possibilità per un satellite di atterrare su questi corpi. In questo contesto, spesso non è possibile avere un controllo diretto del satellite dalla Terra durante l'atterraggio, che rappresenta una fase delicata in cui anche solo un piccolo ritardo di pochi secondi di un comando di controllo può risultare in un evento catastrofico per la missione. Questo è il motivo per cui nasce la necessità di avere a bordo un sistema di navigazione completamente autonomo operativo durante tutta la fase di atterraggio. Questa tesi presenta il corrente stato di sviluppo di un sistema di navigazione autonoma progettato da zero, basato su una singola camera che lavora nello spettro del visibile. La navigazione ottica lavora estraendo punti d'interesse dalle immagini e tracciandoli durante il movimento della camera. Posizione e orientamento di quest'ultima per ogni immagine consecutiva sono poi ricavate rispetto allo scenario osservato usando queste informazioni, e la traiettoria del satellite ricostruita a meno di un fattore di scala. Vengono descritti tutti i blocchi che costituiscono l'architettura del sistema, insieme anche hai differenti metodi e approcci che sono stati studiati e testati per trovare la configurazione che meglio si adatta ad essere applicata per l'atterragio di un satellite. Viene inoltre presentata una valutazione delle prestazioni del sistema di navigazione utilizzando sia sequenze video sintetiche di differenti traiettorie di atterraggio sulla superficie lunare, sia dataset normalmente utilizzati in computer vision. Per concludere, un progetto dettagliato del lavoro necessario per proseguire lo sviluppo del sistema è presentato.

Parole chiave: Atterraggio autonomo, Navigazione Ottica, SLAM, Visual Odometry

AKNOWLEDGMENTS

I would like to thank Prof. Michèle Lavagna for giving me the opportunity to work on this project, that even with its difficulties has proved to be interesting and challenging. I would also like to thank her for all the opportunities that without hesitation has offered to me.
The development of this work would not be possible without supervision of Paolo Lunghi, Ph.D. candidate. My gratitude goes to him for the support and the many advice given, despite the distances and busy schedules.
My sincere thanks to my friends Antonio and Lorenzo, for their help and for their unfailing company in "Sala Calcolo" during these months of work.
A last special thanks goes to my father and my mother, for their ever present love.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

In this chapter the task of landing a spacecraft on a celestial body is presented. Concepts of autonomous planetary landing and optical navigation are discussed, which put together represent the main topic with which this thesis deals with.
To have a proper understanding of the problem faced, state of the art systems for optical navigation are introduced. Space applications are considered along with ones coming from the computer vision field applied to robotics, which has been of inspiration for the development of this work. To conclude, thesis objectives and structure are described along with the chosen implementation tools used.

1.1 SPACECRAFT LANDING ON CELESTIAL BODIES AND AUTONOMOUS NAVIGATION

Spacecraft landing on celestial bodies, being these planets, asteroids or comets, represent the next frontier of space exploration and is becoming a topic of renewed great interest in recent years.
A long heritage of missions successfully completed or under development by space agencies all-over the world exist within this context. These missions have started from the exploration of our neighbor satellite, the Moon, with the first successful mission Luna2 by Russians in 1959, in which the spacecraft was intentionally made impact the Moon surface, followed by other tens of missions of the Luna program in which satellites with rovers have landed on the surface and also brought to Earth soil samples. Always talking about the Moon, it is of course worth to mention all the Surveyor missions by NASA, in which soft landing on the Moon was demonstrated and lots of imaging data were sent to the Earth, all experience gained to be exploited then by the Apollo program, in which six manned mission have landed on Lunar surface. In more recent years, Chandrayaan-1 Moon impact mission by ISRO has been successful, and will be followed by Chandrayaan-2 [3] planned for early 2018. Another interesting mission has been Chang'e-3 by CNSA, which softly landed a satellite with a rover on Lunar surface for the first time since 1976. Satellite carried a camera for autonomous navigation exploited during the last descent phase. Speaking about planned missions, Luna-25 and Luna-27 by ROSCOSMOS are designed to land on the Moon and will carry an autonomous navigation system developed in collaboration with ESA. Another target for landing missions, which in recent years is becoming of growing interest, is of course Mars. Landing missions on Mars

have started in 1971 with the Mars-3 lander by ROSCOSMOS, which lost contact with ground a bunch of seconds after touch-down. First successful landing on Mars surface was then obtained by NASA with the Viking program [4], followed by Mars Pathfinder mission which landed a rover on Mars surface for the first time with the "airbag" landing technique. Maybe the most famous mission on Mars then, is the "*Mars Science Laboratory*" (MSL), which landed the rover "Curiosity" on the surface of the planet [5]. Landing was autonomous with a pre-loaded software and parameters, no active guidance was used and the landing target was selected a priori. This may be considered as the first development step towards a completely autonomous landing system. Nowadays, ESA has launched ExoMars-2016, first of two missions of the *Exomars* program which will land a probe on the Mars surface, while ExoMars-2018, the second mission, will deliver a rover [6].

Speaking of other planets, an interesting landing mission is Cassini-Huygens, which successfully landed a probe on Saturn's moon Titan. ROSCOSMOS also, from the 60's to 80's, has successfully soft landed different spacecraft on Venus surface for the *Venera* program.

Planets are of course of great interest both from the scientific and "human" research point of view, but very interesting landing missions have been successfully performed also on asteroids and comets, especially in the last decades. It is worth to mention NEAR-Shoemaker mission, launched in 1996 by NASA, which has landed for the first time on an asteroid (433-Eros). JAXA, in 1998, has launched and successfully retuned to Earth in 2005 the MUSES-C spacecraft for the Hayabusa mission, which has made a touch-and-go maneuver on asteroid 25143-Itokawa to take samples of the ground. MUSES-C incorporated an autonomous optical guidance to drive the spacecraft during the last descent phase of touch-and-go maneuver [7]. Among missions towards comets, Deep Impact by NASA has been the first mission to impact a comet surface. It is impossible then not to mention the Rosetta mission by ESA, which has landed the first ever spacecraft (Philae) on comet 67P/Churyumov–Gerasimenko [8]. Rosetta is equipped with a set of cameras which are used to make relative navigation with respect to the comet through on-ground manually selected keypoint. For what concerns future missions, OSIRIS-REx is an interesting project to be launched in late 2016 by NASA, which aims at making a sample return from carbon-rich Bennu asteroid. Spacecraft is equipped with instruments for optical navigation as the NavCam, a multiple camera which will be used to track stars and landmarks on Bennu as a piece of the complete Guidance Navigation & Control system [9]. Another newsworthy mission is the *Impact and Deflection Assessment* (AIDA), in development with collaboration between NASA and ESA, which aims at demonstrate the kinetic-impact deflection and characterization of the binary near-Earth asteroid 65803-Didymos.

In a scenario like the one proposed for the missions described, it results evident how the delay in telecommunications between ground segment and spacecraft which is landing on a celestial body far from Earth inhibits the possibility of real-time controlled operations. Moreover, during landing, a change of the target spot may be needed because of some reason, for example due to a failure of one of the spacecraft subsystem or because of the detection of hazards near the selected landing site. For some interplanetary exploration missions, the target landing spot itself or even the surface of the target body may be unknown at mission design phase. In all these cases an autonomous navigation system able to properly manage the whole landing phase in real-time becomes an essential tool, if not the only way to make operations safe and guarantee the necessary flexibility for the mission.

An autonomous landing system is usually build up by different subsystems which accomplish different tasks. All the set of subsystem together constitute the so called Autonomous Guidance Navigation and Control (AGNC). AGNC is built to work in close-loop within a routine which, given measurements from sensors, is able to detect a possible landing site, understand relative position and orientation of the spacecraft with respect to this site, and then estimate the optimal trajectory and control sequence to reach this target. This routine works continuously all across the landing phase until touch-down and adapts autonomously its choices only based on the continuous flow of informations coming from the sensors.
The sensors adopted are the Light Detection and Ranging (LIDAR), Inertial Measurements Unit (IMU), RADAR and optical devices as star trackers or cameras. All of these gives a flow of data which is merged in order to give proper informations in input to the AGNC scheme.
Considering a camera as primary sensor, which is the configuration of interest for this work, Fig. 1.1 shows an AGNC possible architecture [10].

Three main components are actually of particular interest:

- Vision-based Navigation (VN).
- Hazard Detection subsystem (HD).
- Adaptive Guidance (AG).

Vision-based Navigation, or Optical Navigation (OP), has the important task of constantly track the relative position and orientation of the spacecraft with respect to the landing surface given images coming from the camera. In order to increase accuracy its data may be merged with ones coming from other sensors as an IMU or a LIDAR.

Hazard Detection subsystem detects possible hazards on the landing surface and gives proper informations to the landing site selection

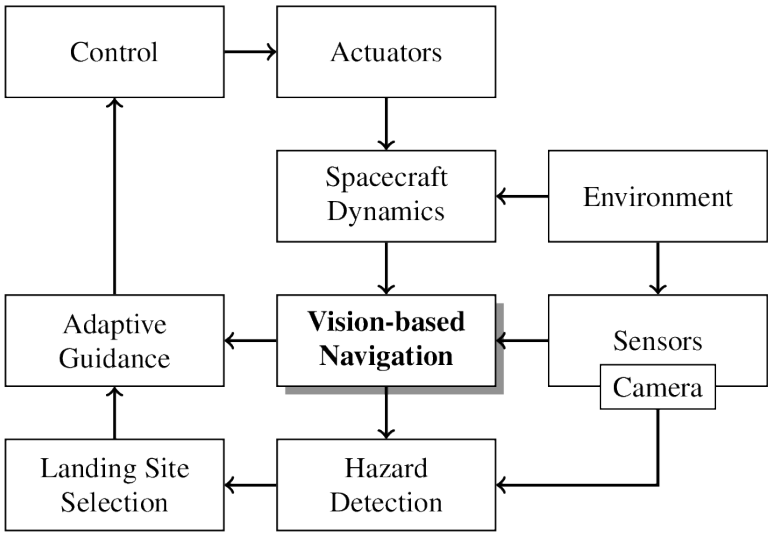


Figure 1.1: *Possible architecture of an AGNC system. Subsystem developed in this thesis is put in evidence.*

routine. Input images from the camera are converted into *Hazard Maps*, in which each pixel represent a degree of hazardousness of corresponding real space on observed surface, which are fed to the landing site selection routine which properly chose a safe landing spot.

Adaptive Guidance, given informations from HD and ON has to retrieve the optimal trajectory to reach the target landing site minimizing fuel consumption and taking into account control authority of the spacecraft.

This thesis, developed at Department for Aerospace and Technology of Politecnico di Milano, focuses on the development of the Visual based Navigation VN subsystem. Final objective is to build an algorithm which is capable of reconstructing the trajectory of a spacecraft landing on a generic celestial body given a flow of images coming from a single camera. Position and orientation of the spacecraft have to be reconstructed with the highest possible accuracy within the whole landing trajectory, the algorithm being robust to all possible conditions which may be encountered in such a scenario.

1.2 OPTICAL NAVIGATION

Optical Navigation has the task of reconstructing position and orientation of the spacecraft in time exploiting measurements coming from one or multiple cameras. It is a complex and multidisciplinary topic which involves a wide range of theory, many different computer

vision techniques and algebraic optimization problems.
First applications of cameras on spacecraft date back to the first exploration missions towards the Moon made by NASA and Russians. Both Luna and Surveyor cited programs for example, have equipped satellites with cameras. Within this context, cameras where just exploited as a tool to take pictures of the unknown Moon surface in order understand its features and map it, a concept which is not related to real-time navigation, even if in some way bounded to it.
Optical Navigation techniques were first developed in the computer vision field in robotics, and have been then for the first time applied on space missions for the navigation of Mars rovers in the *Mars Exploration Mission* (MER). Rovers "Opportunity" and "Spirit" carry a stereo camera system that is used to improve accuracy in navigation, which computes an update of the 6-DOF rover pose by tracking the motion of "interesting" terrain features between two pairs of images [11].
Up to now, direct applications of cameras for spacecraft landing have been used only by a few missions and not as independent subsystem to reconstruct trajectory, but rather as complementing tool which can provide additional informations for pose estimation.
Rosetta spacecraft for example uses two dedicated cameras (NAVCAM) for navigation. These were first exploited to find comet $67P$ when still far from it with the purpose of providing an input for the orbit determination process with the direction in azimuth and elevation of the centroid of the comet [12]. During the approach phase then, when full images of the comet where available, Rosetta navigated relying on landmark observations which were obtained on ground by processing NAVCAM images [13]. This process is not real-time nor autonomous, but represent anyway an efficient application of cameras for navigation purposes. A similar approach was used by MUSES-C spacecraft during its approach phase to Itokawa. When near the asteroid anyway, a real-time optical navigation algorithm was thought to be used for rendezvous phase, but failed to correctly track the asteroid. It was the first attempt to use an Optical Navigation algorithm for autonomous landing. Speaking of future missions, in OSIRIS-REx [9], three cameras are mounted on the spacecraft with the purpose of supporting the mission through all its phases from approach to sample collection (i.e. detecting the asteroid, mapping) [14]. The fact that such an ambitious and new mission that has not been already launched does not make use of a real-time Optical Navigation system during landing phase despite being equipped with cameras clearly underline how this technology is not mature yet in the space field.
To be used for an AGNC, an Optical Navigation algorithm has the stringent requirement to be able to operate in real-time. This excludes almost all the missions so far introduced, and in fact, as far as the author knowledge, currently no Optical Navigation algorithm for real-time trajectory estimation of a spacecraft during landing has never

flown. Despite this, real-time Optical Navigation is a topic of great interest nowadays and lots of projects are under development by different space agencies and research groups all over the world for future mission exploitation.

Most of these projects owe their origins or are inspired by another branch of engineering, which is robotics. The Optical Navigation subject is in fact born within this context, trying to augment possibilities of autonomy for robots which have the task to explore an unknown or even known ambient. It is this the field from which the most interesting innovations come from and which is rapidly and constantly growing being a trending topic in recent years. From this point of view, the most important concepts are Visual Odometry (VO) and Visual Simultaneous Localization and Mapping (V-SLAM):

- **VO** is the process of estimating the egomotion of a vehicle using only the input of a single or multiple cameras. Operational concept is to incrementally estimate the pose of the vehicle through examination of the changes that motion induces on the images [15].
- **V-SLAM** is the process by which a mobile robot can build a map of an explored environment using only images from one or multiple cameras and at the same time use this map to compute its own location [16].

Newest developed space algorithms exploit these concepts taken from robotics and adapt them for the space domain. The transition is not anyway trivial and many problems arise because of different reasons, one among the other being the poor computational power available on-board of satellites with respect to the powerful CPU nowadays used on ground. This is a partially solved problem by the way, since space industries are moving in this direction producing dedicated hardware designed exactly for this scope, as shown in [17], [18]. A simplified scheme which resumes the architecture of a generic real-time ON algorithm is shown in Fig. 1.2.

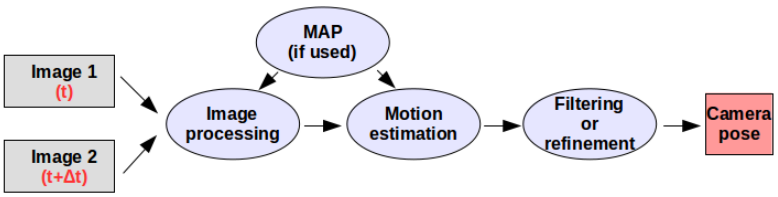


Figure 1.2: *Simplified architecture for a real-time Optical Navigation algorithm. After camera pose is retrieved it results trivial to obtain also position of the device at which it is attached.*

Images from the camera are processed and interesting features or regions from the observed scene are identified and related with ones

coming from the other frames or with the map, if used. This gives a relation between two consecutive frames or between the observed scene from a frame and the stored map which is then exploited to retrieve the motion of the camera.
A separate discussion shall be anyway made for the map, which can be a powerful instrument even if not strictly essential. A basic distinction shall be made between ON algorithms which use a pre-existent map already stored in memory and ones that does not and incrementally build it in real time (as in V-SLAM). Most of the state of the art algorithms currently developed for space applications exploit the first concept, usually applying it to Lunar landings, making the so called Terrain Relative Navigation (TRN). These systems are anyway bounded to work on already known planetary regions since they rely on a map built on ground.

1.2.1 *VO and SLAM Problems*

Due to their importance in computer vision and because being the Optical Navigation algorithm developed in this work inspired by these concepts, a brief introduction to Visual Odometry and visual SLAM is hereafter given. Description follows the one from [15].
As already stated, VO is the process of estimating incrementally in real-time the position and orientation of a vehicle examining changes that motion induces on images taken from one or multiple cameras. V-SLAM instead, is the problem of consistently build a coherent map of an environment being explored while simultaneously reconstructing trajectory, exploiting one or multiple cameras.
The two problems are in some way very similar, especially speaking about the tools used for implementations, but the objectives are different. In general SLAM wants obtain a global consistent estimate of the robot path and this is performed keeping a map of the environment, understanding when a loop closure occurs (i.e. when the same place is revisited a second time and so trajectory closes) and efficiently integrating this new constraint into the current map to increase accuracy. Conversely VO acts incrementally pose after pose potentially optimizing only last poses. The philosophy is different: VO cares about consistency of the trajectory while local map, if implemented, is only exploited as a tool, SLAM instead is concerned with map consistency. A V-SLAM method is potentially much more precise, but it is more complex and computationally expensive. With both methods, using a single camera, motion can only be recovered up to a scale factor.
Two main categories of VO and V-SLAM can be then distinguished: feature-based methods and appearance-based one. Feature based methods are based on salient and repeatable features that are tracked over the frames; appearance-based methods instead use the intensity

information of all the pixels in the image or subregions of it. Global methods are less accurate then feature based ones and are computationally more expensive. Feature-based methods on the other hand require the ability to robust match or track features across frames but are faster and more accurate.
Path of the camera is computed incrementally by VO, hence the errors introduced by each new frame accumulate over time. This generates a drift, and to keep it small as possible local optimization techniques are applied over the last camera poses and the map. This method is usually called *Bundle Adjustment* and is widely applied in computer vision. Also V-SLAM computes the camera path incrementally, but can exploit loop-closing to reduce the accumulated drift over time. Optimization techniques are then anyway used, in particular to increment the built map accuracy over time.
Since loop closing is not an exploitable option for a planetary landing, between the two methods, VO has been considered the most interesting one for a navigation system implementation, and has been used as a guideline for the development of the work here presented. A classical Visual Odometry functional architecture is given in Fig. 1.3.

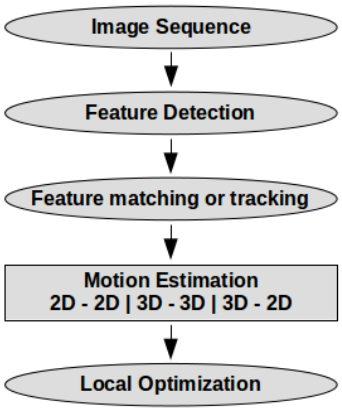


Figure 1.3: *Functional architecture of a Visual Odometry system, a similar flow may be considered valid also for V-SLAM adding "loop-closure".*

Feature detection step extracts $2D$ features from the images given by the camera. Features are salient points which can be repeatedly observed in successive images. Matching or tracking step respectively consists in matching those extracted features between couple of consecutive frames or alternatively track them using local search techniques. The motion estimation step computes then relative motion between consecutive frames. This can be done with different approaches depending whether correspondences from previous step are specified in three or two dimensions. Camera pose is finally obtained as a concatenation of all the relative motion between frames. In the end, the iterative refinement technique called "Bundle Adjustment" is applied over a fixed number of last frames to increase trajectory accuracy. For

an exhaustive description of VO and V-SLAM reader is referred to [15] and [19].

1.3 STATE OF THE ART

1.3.1 *Autonomous Guidance, Navigation and Control Systems*

In the following, state of the art for what concerns AGNC systems is presented. Despite nor of these system have been space proven yet, they represent as far as the knowledge of the author the most advanced and ready to be implemented tools for autonomous navigation. When available, a glance on the Optical Navigation subsystem will be given.

1.3.1.1 ***ALHAT***

Autonomous precision Landing and Hazard Avoidance Technology (ALHAT), is a project in development and testing phase by NASA. System is designed for completely automated descent and landing of a spacecraft and has three main elements: Sensors, Terrain Sensing and Recognition (TSAR), and Autonomy, Guidance, and Navigation (AGN). Sensor set includes a 3-D Flash LIDAR used to image the surface for hazard detection and avoidance (HDA) as well as hazard relative navigation (HRN). Navigation sensors include an inertial measurement unit (IMU), star tracker (ST), altimeter (ALT), and doppler velocimeter (VEL).
The AGN system provides state estimates based on sensor measurements including Hazard relative navigation (HRN), guidance to the landing target, and Terrain relative navigation (TRN) [20].

TRN is managed by APLNav, a passive optical terrain relative navigation system for precise landing. APLNav performs state estimation relative to surface terrain by comparing on-board images to estimated images from an on-board map. The algorithm can also estimate velocity by performing a similar calculation on sequential images when an on-board map is not available [21].

1.3.1.2 ***SINPLEX***

SINPLEX, *Small Integrated Navigation System for PLanetary Landing*, is a system in development by a consortium guided by DLR, which aims to develop a solution in order to reduce significantly the mass of the navigation subsystem for exploration missions which include a landing and/or a rendezvous and capture or docking phase. SINPLEX exploits a star tracker (STR) to provide inertial relative attitude. A navigation camera provides target relative position and attitude information. A laser altimeter/range finder (LA) provides range information relative

to the target and scaling information for the navigation camera images. An inertial measurement unit (IMU) provides high-rate specific force and angular rate information. A navigation filter is then used to merge all the measurements and obtain a state estimation[22].
System has been tested on the DLR's TRON facility, yielding accurate trajectory reconstructions for simulated Lunar landings [23].

1.3.1.3 *MOSS*

Multi Ocular Smart Sensor (MOSS) is a project co-financed by the Italian Space Agency to develop a compact high performance equipment specifically designed for visual navigation which uses multiple cameras and dedicated hardware.
MOSS exploits three different cameras and a dedicated processing unit for visual navigation. System is currently under development and a Visual Based Navigation algorithm has been prototyped to asses the performance of the system. MOSS components are thought to be combined so to implement different configurations, in accordance with the requirements of different operational scenarios. [17].

1.3.1.4 *LVS*

Lander Vision System (LVS), is a project being developed by JPL for Mars landing applications within the context of *Mars 2020* mission for the parachute descent phase of the lander. LVS is composed of a camera, an Inertial Measurement Unit (IMU) and a dedicated compute element loaded with real-time software and FPGA firmware to perform state estimation at a 1Hz image update rate [24]. The system runs Map Relative Localization (MRL) algorithms which fuse landmark matches between pixels in a descent image and 3D locations in a map with inertial angular rates and accelerations provided by the IMU. Landmark matching is divided into a coarse phase to cut out initial large error measurements and in a fine phase which has the objective to improve position estimation accuracy to less then a required threshold.
LVS has flown in a real-time helicopter field test obtaining accuracy of the order of 40 meters [24].

1.3.2 *Navigation Algorithms*

To have a better overview of the Visual Based Navigation problem, a brief description of the main software developed for optical navigation is given. Due to the underlined multidisciplinary application of this subject, it has been decided to split in two different parts

the overview: software specifically developed for space applications (mostly of them in the context of Moon exploration) and software developed for robotics.

1.3.2.1 *Optical Navigation Algorithms for Space*

Within the space application context, different works have been developed for the Lunar landing scenario. [25] describes a system designed for Lunar landing which makes a data fusion between an optical navigation algorithm, an IMU, an altimeter and a star tracker. Navigation algorithm relies on a crater detection scheme which detects and match observed craters with an internal dataset to determine motion between couple of frames. Global system accuracy is below 10 m in position and 0.15° in attitude. A similar approach is used in [26], which proposes a feature based navigation algorithm that matches observed craters to a pre-existing database, tracks them across the image and then uses an Extended Kalman Filter (EKF) to estimate the pose of the vehicle.
An interesting work developed for an asteroid landing scenario has been done by JAXA in view of the launch of the Hayabusa-2 mission, as a development of the algorithms used for Hayabusa-1. A SLAM method based on a Rao-Blackwellized particle filter able to estimate pose and attitude of a spacecraft navigating near an asteroid is proposed [27]. Algorithm uses visual signatures to recognize landmarks on the observed surface and triangulate its position with respect to it. Measurements from the camera are then fused along with ones from an IMU and an altimeter and fed in input to a filter for pose estimation. Among the described algorithms, this is the only one designed to be completely autonomous, not relying on a database map.

1.3.2.2 *Algorithms for Robotics*

Algorithm developed for robotics represent at the moment the most advanced tools for optical trajectory estimation of a camera. Breaking works in this field were *monoSLAM*, the first successful application of the V-SLAM problem to the vision domain exploiting a single camera [28], and *Visual Odometry for Ground Vehicle* [29], that coined for the first time the term "Visual Odometry" and uses real-time motion estimation for navigation with a mono or stereo camera system. Within this context it is also worth to cite *PTAM* [30], which paved the way for the development of the currently most performing algorithms introducing the concept of multi-thread splitting of the navigation tasks, making it possible to use in real-time heavy optimization techniques never used before.
In the following, two state of the art robotic algorithms for optical

navigation with mono-camera which are considered as the most representative of their category by the author and which have also been of inspiration for the development of the work presented in this thesis are described.

SVO Fast Semi-Direct Visual Odometry: SVO is a semi-direct monocular Visual Odometry algorithm developed in the context of micro-aerial-vehicles with a down working monocular camera [31]. Algorithm operates directly on pixel intensities and uses a probabilistic mapping method that explicitly models outlier measures to estimate 3D map points. Two parallel threads are implemented, one for estimating camera motion, and a second one for mapping as the environment is being explored.

Motion estimation thread implements a semi-direct approach to relative-pose estimation. Pose is first initialized through image alignment: camera pose relative to the previous frame is found through minimizing the photometric error between pixels corresponding to the projected location of the same 3D points.
Given an image and its pose the mapping thread estimates the depth of 2D features. Depth estimate is modelled with a probability distribution updated at every subsequent observation. When the variance of the distribution becomes small enough, the depth-estimate is converted to a 3D point which is directly inserted into the map and used for motion estimation.

SVO reconstructs the scene and the trajectory up to scale, unless it works in parallel with an Inertial Measurement Unit, and reaches precision superior to most of the state of the art algorithms (even if not the best)[31].
It have to be anyway underlined that algorithm works well if camera is always looking downwards, but precision drops quite fast if some out-of-the-pointing-plane rotation of the camera occurs. This has become evident also from some test done at Politecnico di Milano with synthetic images of Lunar descent trajectories.
About real time performance, SVO is declared to work on an embedded platform on a MAV at 55 fps with 3.04 milliseconds required to estimate motion at each step.

ORB-SLAM: ORB-SLAM is a feature based monocular V-SLAM system that operates in real-time and is able to manage large outdoor environments. It works on three dedicated parallel threads: tracking, mapping and loop closing.
Tracking is in charge of localizing the camera with every frame and decide when sending new informations to the map. Matching between current images and the map are used to retrieve motion.
The mapping thread manages the map, processing new keyframes

and performing optimization to increase accuracy of the reconstructed environment. A culling policy is also applied to detect and delete "bad points" introduced in the map.
Last thread, loop closing, searches for recognition of already visited places in order to close the trajectory and optimize it reducing the accumulated drift.

ORB-SLAM has an accuracy of the order of some meters in large outdoor scenario. Authors declare this algorithm to be the most reliable and complete solution for monocular SLAM currently available [32].

1.4 THESIS OBJECTIVE

This thesis deals with the development of an Optical Navigation algorithm based on a monocamera. The following requirements shall be met:

- Algorithm shall operate in real-time.
- Algorithm shall work independently from the environment, being this a planet or a smaller body.
- Trajectory shall be reconstructed both in terms of attitude and position of the spacecraft with the highest possible accuracy.
- Algorithm shall be able to operate autonomously, without prior knowledge of the landing environment.

Requirement one seems trivial, but is the most stringent from the development point of view. Requirement two is set because the aim is to realize a robust system, able to operate under different conditions. While third requirement is granted, the last one implies that no prior map (e.g. from a database) of the landing site can be used.
The final objective, of which this work constitutes the first development step, is the buildup of an autonomous Optical Navigation system to be integrated in an Adaptive Guidance, Navigation and Control chain for autonomous spacecraft landings of which the Hazard Detection and Avoidance and the Guidance subsystem are already under implementation at PoliMi. Algorithm shall be validated through extensive testing exploiting synthetic images of simulated spacecraft landing trajectories and tanks to an experimental facility currently under construction at PoliMi.

1.5 DEVELOPMENT TOOLS USED

The whole work hereafter presented has been developed in C++ exploiting open source software. This choice has been made since the beginning in order to accommodate the need of building an algorithm whose final scope is to be implemented on hardware, being free from proprietary licensing, and being as cost effective as possible. From this point of view, C++ exactly met these requirements, being also a powerful instrument both in terms of implementation possibilities and computational speed. Moreover most of the state of the art free source algorithms studied are written with this language.
The core of the algorithm has been built exploiting OpenCV-3.1.0 library[33], which is specifically developed for computer vision and that includes implementation of many useful state of the art algorithms.
Compiler used for C++ is GNU's g++ 5.2.1. For what concerns data post-processing, trajectories comparison and results analysis have been done using Matlab® has been used.

1.6 THESIS OUTLINE

The work done in this thesis is structured as follows: Chapter 2 gives an overview of the theoretic which describes cameras and multiple view geometry, which is at the basis of motion estimation from multiple camera views. Chapter 3 introduces all the algorithms and the tools which put together constitute the building blocks used to create the proposed Optical Navigation Algorithm. Chapter 4 describes the current development status of the Navigation system along with its different implemented configurations and the tests performed on them in order to assess their performance with simulated landing trajectories on the Moon. To conclude, in Chapter5 a detailed development program for the future enhancement of the proposed system is given and, in the end, Chapter 6 presents the conclusions for this work.

2 FUNDAMENTALS

In this chapter some basic geometric and mathematical concepts, necessary for a complete understanding of the Visual Navigation with monocamera problem are expounded.
First, a detailed description of the model used for a camera which projects 3D world points to 2D points on an image will be given. Then, the so called epipolar geometry, which describes the motion of a camera in relation to these observed 2D and 3D points, will be analysed. To conclude, a brief overview of RANSAC, a statistical approach for the estimation of a model given a set of data with possible outliers, will be given.

2.1 CAMERA MODEL

One of the most important thing in computer vision is the application of a model which coherently describes the camera used. This is fundamental to address taken measurements from images.
A camera represent a mapping between the 3D world (object space) and a 2D image [1]. The model here described is the most simple, that is the *basic pinhole camera*. This model can be simply described thanks to projective geometry, while geometric entities of the camera can be computed through matrix representation.
Pinhole camera model can be representative of any classical camera used for Optical Navigation in space and for this reason has been chosen, nevertheless, using a different model would not have affected the validity of obtained results.

The model representation given in Fig. 2.1 is used for the pinhole camera. World points are central projected onto a plane with centre of projection which is also the origin of a Euclidean coordinate system. Plane $Z = f$ is called *image plane* or *focal plane*. With this model, a

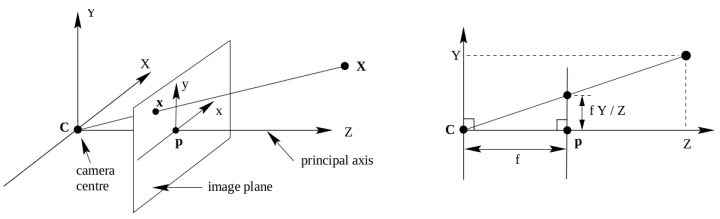


Figure 2.1: *Pinhole camera model. Image from [1]*

point in space with coordinates $\mathbf{X} = (X, Y, Z)^T$ is mapped to a point on the image plane as the intersection with the image plane itself with

the line joining the point **X** and the centre of projection.
By similar triangles it is possible to compute that a $3D$ point is mapped on the image plane as:

$$(X, Y, Z)^T \rightarrow \left(f\frac{X}{Z}, f\frac{Y}{Z} \right)^T \tag{2.1}$$

This is a mapping from Euclidean space $\mathbb{R}^3$ to Euclidean space $\mathbb{R}^2$.
The centre of projection is called the *camera centre* or *optical centre*. The line from the camera centre perpendicular to the image plane is called the *principal axis* or principal ray of the camera, and the point where the principal axis meets the image plane is called the *principal point*. The plane through the camera centre parallel to the image plane is called the *principal plane* of the camera [1].
Exploiting an homogeneous representation of the world and image points, the central projection can be expressed as a linear mapping:

$$\begin{pmatrix} fx \\ fy \\ z \end{pmatrix} = \begin{bmatrix} f_x & 0 & x_0 & 0 \\ 0 & f_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = K[I|\mathbf{0}] \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{2.2}$$

Here f_x and f_y are focal length of the camera. Focal length is the distance between the pinhole and the image plane. In a true pinhole camera, both f_x and f_y have the same value.
x_0 and y_0 represent instead the "principal point offset", which is the location of the principal point relative to the image plane origin. The exact definition depends on which convention is used for the location of the origin.
The matrix K is called *intrinsic camera matrix*, and transforms 3D camera coordinates to 2D homogeneous image coordinates with its parameter representing the geometric property of the camera. $[I|0]$ are called *extrinsic parameters*, and are really important since they describe location of the camera in the world and in which direction it is pointing; in this form, the camera is assumed to be located at the origin of a Euclidean coordinate system with the principal axis of the camera pointing as the Z-axis. This reference frame is called *camera coordinate frame*.
Calling world points represented by the homogeneous vector $(x, y, z, 1)^T$ as **X**, and **x** the homogeneous image points, Eq. 2.2 can be expressed as:

$$\mathbf{x} = P\mathbf{X} \tag{2.3}$$

where $P = K[I|0]$ is called camera *projection matrix*.

In general, points in space are expressed in terms of a different Euclidean coordinate frame then the camera coordinate one. This frame is called *world coordinate frame* and is related to the previous

through a rotation and a translation. The extrinsic matrix takes thus the form of a rigid transformation matrix: a 3×3 rotation matrix in the left block, and a 3×1 translation column vector in the right:

$$P = K[R|t] \tag{2.4}$$

The most intuitive way to understand how these reference system work is to specify the camera's pose directly rather than specifying how world points are related to camera coordinates. Building an extrinsic camera matrix this way is just a matter of building a rigid transformation matrix that describes the camera's pose in world coordinates and then take its inverse.
Define C as the column vector describing the location of the camera center in world coordinates, and R_c as the rotation matrix describing the camera's orientation with respect to the world coordinate axes. Camera pose is then described by the transformation matrix $[R_c|C]$. The matrix is make square by adding an extra row of $(0, 0, 0, 1)$ for convenience, and the extrinsic matrix is then obtained by inverting the camera's pose matrix:

$$\begin{bmatrix} R & t \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_c & C \\ \mathbf{0} & 1 \end{bmatrix}^{-1} = \left[\begin{bmatrix} I & C \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R_c & 0 \\ \mathbf{0} & 1 \end{bmatrix} \right]^{-1} = \begin{bmatrix} R_c & 0 \\ \mathbf{0} & 1 \end{bmatrix}^{-1} \begin{bmatrix} I & C \\ \mathbf{0} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R_c^T & 0 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} I & -C \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_c^T & -R_c^T C \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.5}$$

Thus, the relationship between the extrinsic matrix parameters and the camera's pose is straightforward:

$$R = R_c^T \tag{2.6}$$

$$t = -RC \tag{2.7}$$

This means that when referring to a camera matrix $P = K[R|t]$, R and t will represent the position of the world reference frame with respect to the camera coordinate frame.
This notation will be used in the rest of this thesis.

2.2 PROJECTIVE GEOMETRY: HOMOGRAPHY

Most of the theory presented in this chapter is based on projective geometry. $2D$ projective geometry is the study of properties of the projective plane $\mathbb{P}^2$ that are invariant under a group of transformations known as projectivities. A projectivity is basically a mapping between $2D$ points from $\mathbb{P}^2$ to itself.
Since the inverse of a projectivity is still a projectivity, projectivities form a group. An alternative name for a projectivity is *homography*, and can be algebraically defined as follows [1]:

A mapping $h : \mathbb{P}^2 \leftrightarrow \mathbb{P}^2$ is an homography if and only if there exists a non-singular 3×3 matrix H such that for any point in $\mathbb{P}^2$ represented by a vector x it is true that $h(x) = Hx$.

Considering two 2D points expressed in homogeneous coordinates, homography can be expressed as a 3×3 non singular matrix:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{13} & h_{23} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.8}$$

or, in short, $\mathbf{x}' = H\mathbf{x}$.
H is an homogeneous matrix, since any multiplication by a non-zero scale factor leads to the same projective transformation. A projective transformation projects every figure into a projectively equivalent figure, leaving all its projective properties invariant.
Homography holds just for projectivity between planes: if applied to a real image, which is planar but that represent a scene which may not be, than an homography may not exist. This is an important concept as will be shown in later chapters, since if an homography between two real images is found, then motion between the two cameras can be retrieved in terms of rotation and translation. If the scene observed is not planar or not at least approximable as planar then the whole process is not applicable since an homography cannot be found.

2.3 EPIPOLAR GEOMETRY: FUNDAMENTAL AND ESSENTIAL MATRIX

Most of the theory here described is taken from [1], which is the most complete and exhaustive book for computer vision applications. The reader is encouraged to consult it for a more complete and detailed description of these topics.

The epipolar geometry is the intrinsic projective geometry between two views. It is independent of scene structure, and only depends on the camera's internal parameters and relative pose [1].
Considering two views, this geometry is described by the intersection of the image planes with the pencil of planes having the line joining camera centres (baseline) as axis. The fundamental matrix is then the algebraic representation of epipolar geometry.
Before deriving the fundamental matrix F, two important geometric entities must be defined. Considering Fig. 2.2, the first one is the *epipolar line l*: given a pair of images the epipolar line is the projection in the second image of the ray from the point x through the camera centre C of the first camera. Considering a point x on an image, there exists a corresponding epipolar line l' in the other image, any point

x' in the second image matching point x must lie on the epipolar line l. The second entity is the *epipole* **e**, which is the point given by the intersection of the line connecting the camera centres with the image plane. All epipolar lines intersect at the epipole.

A simple way to derive the fundamental matrix F is then the geomet-

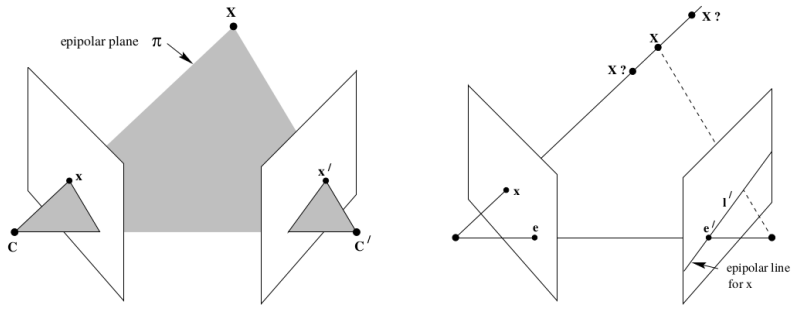


Figure 2.2: *Epipolar geometry, from [1]*

ric one. Referring to Fig. 2.2, imagine a plane β in space not passing through either of the two camera centre. A ray from point $\mathbf{x}$ on first camera identifies $\mathbf{X}$ lying on β, which is then projected to $\mathbf{x}'$ on second image. Since $\mathbf{X}$ lies on the ray corresponding to $\mathbf{x}$, projected point $\mathbf{x}'$ must lie on the epipolar line $\mathbf{l}'$ corresponding to the image of this ray. Considering the set of all possible points $\mathbf{x}_i$ lying on the first ray, the corresponding ones $\mathbf{x}'_i$ in the second image will be projectively equivalent. As a consequence, a 2D homography H_β mapping each $\mathbf{x}_i$ to $\mathbf{x}'_i$ exist.

Considering point $\mathbf{x}'$, the epipolar line $\mathbf{l}'$ passing through $\mathbf{x}'$ and the epipole $\mathbf{e}'$ can be written as $\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{x}'$. Since $\mathbf{x}'$ may be written as $\mathbf{x}' = H_\beta \mathbf{x}$, then:

$$\mathbf{l}' = [\mathbf{e}'] \times H_\beta \mathbf{x} = F\mathbf{x} \tag{2.9}$$

where the fundamental matrix is defined as:

$$F = [\mathbf{e}'] \times H_\beta \tag{2.10}$$

Fundamental matrix F is a matrix of rank 2, since it represents a mapping from a 2-dimensional onto a 1-dimensional projective space. The most important property of F is that for any pair of corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$ in two images the following constraint holds:

$$\mathbf{x}'^T F \mathbf{x} = 0 \tag{2.11}$$

this equation is really important since it gives a way to characterize the fundamental matrix only knowing image point correspondences. F has seven degrees of freedom, but in the case of pure translation between the camera, which may be of interest for this application, fundamental matrix becomes $F = [\mathbf{e}']\times$, which is skew-symmetric and has only 2 degrees of freedom, corresponding to the position of

the epipole. An other important property of the fundamental matrix F is that it is singular, being of rank 2. Moreover the left and right null-spaces of F are generated by the two vectors representing (in homogeneous coordinates) the epipoles in the two images. This is the so called *singularity constraint* and is a property widely exploited by applications that rely on fundamental matrix.

Let's introduce now a specialization of the fundamental matrix for the case of normalized image coordinates, that is the *Essential matrix* E.
Normalized coordinates are obtained applying the inverse of the camera intrinsic matrix K to a point $\mathbf{x}$: $\hat{x} = K^{-1}\mathbf{x}$. Considering a camera matrix $P = K[R|t]$, then $x = PX$ for a point in the image and consequently $\hat{x} = [R|t]\mathbf{X}$. The camera matrix $K^{-1}P = [R|t]$ is called *normalized camera matrix*, with the effect of the intrinsic matrix being removed.
Given two normalized camera matrices $P = [I|0]$ and $P' = [R|t]$, the essential matrix corresponds to the fundamental matrix defined for the two normalized cameras with the form:

$$E = [t] \times R = R[R^T t]_\times \tag{2.12}$$

With the following constraint equation:

$$\mathbf{x}'^T E\mathbf{x} = 0 \tag{2.13}$$

in the form of normalized image coordinates for corresponding points $\hat{x} \leftrightarrow \hat{x}'$.
If one substitutes $\hat{x}$ and $\hat{x}'$ inside Eq, 2.13 obtains:

$$\mathbf{x}'^T K'^T E K^{-1}\mathbf{x} = 0 \tag{2.14}$$

comparing this one with the relation Eq. 2.11 for the fundamental matrix, the relationship between the fundamental and essential matrices is obtained:

$$E = K'^T F K \tag{2.15}$$

Essential matrix has only five degrees of freedom: both the rotation matrix R and the translation t have three degrees of freedom, but there is an overall scale ambiguity which is characteristic of E. The most important property of the essential matrix is that once it is known, the camera matrices and thus the pose of the cameras may be retrieved. This problem will be explained in detail in later chapters, and represent one of the core step for motion reconstruction.

2.4 RANDOM SAMPLE CONSENSUS: BACKGROUND

Random Sample Consensus (RANSAC), is an iterative algorithm build for fitting a model to experimental data containing outliers. It is a

non-deterministic algorithm, producing results only with a certain probability, with this probability increasing as more iterations are done. A full description of the method can be found in [34].

The basic assumption behind RANSAC is that the data consists of *inliers*, which are data whose distribution can be explained by some model, and *outliers*, which are data that for some reason (i.e. noise) do not fit the model. RANSAC assumes also that given a small set of inliers a procedure which can estimate the parameters of a model that optimally fits this data exists.
The RANSAC procedure consists thus in using an initial dataset as small as possible and then enlarge it with consistent inliers. Given a model that requires a minimum of n data points to instantiate its free parameters, and a set of data points P such that the number of points in P is greater than n, $P \geq n$, randomly select a subset $S1$ of n data points from P and instantiate the model. Use the instantiated model $M1$ to determine the subset $S1^*$ of points in P that are within some error tolerance of $M1$. The set $S1^*$ is called the consensus set of $S1$.
If $S1^*$ is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use $S1^*$ to compute (possibly using least squares) a new model $M1^*$.
If $S1^*$ is less than t, randomly select a new subset $s2$ and repeat the above process. If, after some predetermined number of trials, no consensus set with t or more members has been found, either solve the model with the largest consensus set found, or terminate in failure.

An explanatory example may be done considering the estimation of the essential matrix E from a set of point correspondences $\mathbf{x}' \leftrightarrow \mathbf{x}$ containing some outliers (i.e. points that do not satisfy Eq. 2.13): RANSAC approach would be to select the minimum set of point needed to estimate E, test the obtained E with all the remaining points and repeat the procedure for a new set of minimum points. The trial with more points verifying the model is then selected as the good one and enforced with all the other points which verifies it, for example with least square smoothing.

3

OPTICAL NAVIGATION TOOLS

In this chapter a complete description of the possible tools available for each building blocks of the navigation algorithm is given along with a comparative analysis. At first feature detection algorithms are considered; then different methods for motion estimation are introduced. To conclude, an overview on refinement techniques exploited to increase accuracy of the trajectory is given.

3.1 OVERVIEW

The input for a vision based navigation algorithm are the raw images coming from a single or multiple camera system. In particular in this work, it has been considered as requirement a monocular approach. The choice made for the construction of the navigation algorithm is to follow a Visual Odometry type method; this is dictated by the fact that a landing trajectory, independently from the celestial body on which is done, is open, and consequently the same scene is not revisited more than one time. A V-SLAM approach is therefore not feasible, since it would not be possible to exploit loop-closing to reduce drift. Despite this, it has been considered important to build a map of the explored environment as all SLAM methods does. This is important for two main reason: First of all having a reconstructed $3D$ map gives the chance to implement refinement methods otherwise not exploitable, secondly, a reconstructed map of the landing environment, if enough accurate, is a key tool which can give lots of information to other subsystem of the AGNC increasing its capability(an example among the other being the Hazard Detection System, which can improve its landing site choice knowing also depth of the scene). The approach which has been decided to follow so is to try reconstruct trajectory in a dead reckoning mode contemporaneously building a map of the explored environment.

3.2 FEATURE DETECTION & DESCRIPTION

Feature-based methods work on salient and repeatable *features* that are found over the incoming frames from the camera. In this context, feature detection represent the first processing step of the images, and it is usually one of the most demanding ones in terms of computational burden.
A *feature* is thought to be a point of interest in an image, as could

be a line or a corner, which is distinctive and so can be repeatedly found across other images from different viewpoints. Features are usually found by methods which compare brightness of a point with its neighbours. Along with the detection itself, which only finds the position in the image, a "label" may be also assigned to each feature in order to make it distinctive. This label is the so called *descriptor*. Descriptors are built extracting properties from a small image patch around the found features.
To be useful for navigation purposes, features shall be robust, which means should have the following properties:

- Scale invariance
- Resistance to noise
- Resistance to light changes
- Rotation invariance

The objective is to find the same features continuously across incoming images in order to later exploit multiple view geometry to extract information about the camera position and orientation.

3.2.0.1 *Harris Detector*

The most classical and intuitive way to extract corners from an image is to search for regions surrounding a pixel with strong local gradient in orthogonal directions. This idea was proposed by Harris [35] in 1988 as an improvement of the work introduced by Moravec years before [36].
Considering image intensity I at pixel location x,y, a change of intensity in the image for a shift u,v can be expressed as:

$$E(u,v) = \sum w(x,y) \cdot [I(x+u, y+v) - I(x,y)]^2 \tag{3.1}$$

Where w is a window function which is usually a Gaussian. For a small shift it is possible to apply the following bilinear approximation:

$$E(u,v) \simeq [u\, v] \cdot M \cdot \begin{bmatrix} u \\ v \end{bmatrix} \tag{3.2}$$

where M is a 2x2 matrix of image derivative:

$$M = \sum w(x,y) \cdot \begin{bmatrix} I_x^2 & I_x \cdot I_y \\ I_x \cdot I_y & I_y^2 \end{bmatrix} \tag{3.3}$$

With $I_x = dI/dx$ and $I_y = dI/dy$ intensity derivatives. Intensity change in a shifting window can be calculated by eigen value analysis of M. Harris detector proceeds by searching for points where the second-moment matrix M around x has two large eigenvalues. Instead

of explicitly computing the eigenvalues of M, this method checks the value of the ratio:

$$r = \frac{\lambda_1}{\lambda_2} \tag{3.4}$$

considering:

$$det(M) = \lambda_1 \cdot \lambda_2 \tag{3.5}$$

$$trace(M) = \lambda_1 + \lambda_2 \tag{3.6}$$

the following equivalence is obtained:

$$\frac{trace^2(M)}{det(M)} = \frac{(r+1)^2}{r} \tag{3.7}$$

and the index r can be computed as:

$$r = det(M) - k \cdot trace^2(M) \tag{3.8}$$

with k constant with typical values in the range of $0.04 - 0.06$. r is large for corners, negative with large magnitude for an edge and with small modulus for a flat region. In this way there is no need to compute exact eigen values. Harris points are typically precisely located as a result of using first derivatives and of taking into account a larger image neighborhood, with invariance to rotation and affine intensity change. A detailed description of the Harris corner detector can be found in [35].

While shown to be remarkably robust to image plane rotations, illumination changes and noise, the locations returned by the Harris detector are only repeatable up to relatively small scale changes, the reason for this being that they rely on derivatives computed at a certain fixed base scale. This is not an optimal choice for a matching/tracking situation in which a consistent variation in altitude and consequently in scale occurs.

3.2.0.2 *Shi-Tomasi Detector*

The Shi-Tomasi detector was developed in 1994 as an improvement of the Harris detector in [37].
While the condition to find a corner for Harris was expressed by:

$$r = det(M) - k \cdot trace^2(M) \tag{3.9}$$

Shi-Tomasi proposed:

$$r = min(\lambda_1, \lambda_2) > \lambda \tag{3.10}$$

Where λ_1, λ_2 are the eigen values of second order matrix of first derivative M as before, and λ is a predefined threshold.

This different choice with respect to the Harris detector is motivated by the fact that under the assumption of affine image deformation the corner strength function of Eq. 3.10 provides better results in terms of feature tracking across more frames (i.e. more features robustly tracked). For an exhaustive explanation, reader is referred to consult [37].

3.2.0.3 *Features from Accelerated Segment Test Detector*

A more recent work which performs well both in terms of performance and accuracy is represented by the Features from Accelerated Segment Test (FAST) detector introduced by Edward Rosten and Tom Drummond in [2].
FAST works by selecting single pixels which have to be identified as interest points or not. Supposing the intensity of one of those is I_p, after selecting an appropriate threshold t, it builds a circle of pixels around p. Segment test criterion operates then classifying pixel p as a corner if there exists a set of n contiguous pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$.
A high-speed test first exclude a large number of non-corners. Sup-

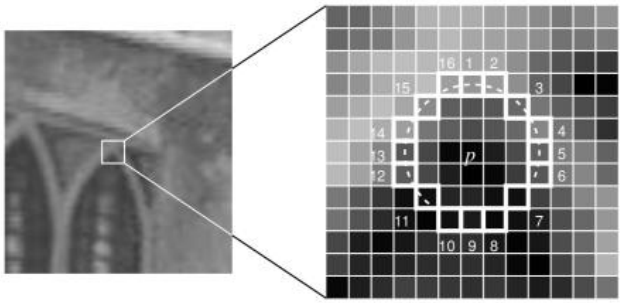


Figure 3.1: *Example of segment test for a candidate corner p, the dashed arc indicates 12 pixels all brighter than p by more than a threshold (Image from [2])*

posing a circle of 16 pixels as shown in the image, this test examines only the four pixels at 1, 9, 5 and 13. If p is a corner, then at least three of these must all be brighter than $I_p + t$ or darker than $I_p - t$. If neither of these is the case, then p cannot be a corner. The full segment test criterion can then be applied to the passed candidates by examining all pixels in the circle.
This method alone has anyway several weaknesses according to the authors, which are all addressed with a machine learning approach and using non-maximal suppression. First, machine learning creates a decision tree which can correctly classify all corners seen in a training set, then converted into a string of nested if-then-else statements which is used as corner detector. Non-maximal suppression is in the end applied to a specifically built score function in order to remove corners which have an adjacent corner with an higher score.

FAST in general offers considerably higher performances than the other feature detectors, but is not very robust in the presence of noise. It have to be considered also that by itself it does not give an orientation to the features extracted and also no scale information.

3.2.0.4 *Scale Invariant Feature Transform Detector and Descriptor*

Scale Invariant Feature Transform (SIFT) is a feature detector and descriptor developed by David G. Lowe and published in [38].
The SIFT detector first identifies locations that are invariant to scale change of the image. This is done by searching for stable features across all possible scales, using a continuous function of scale known as *scale space*; it is demonstrated that under a variety of assumptions the only possible scale-space kernel is the Gaussian function $G(x, y, \sigma)$. Lowe proposes to use scale-space extrema in the difference-of-Gaussian function convolved with the image, $D(x, y, \sigma)$, which is computed from the difference of two nearby scales separated by a constant multiplicative factor k:

$$D(x, y, \sigma) = (G(x, y, k \cdot \sigma) - G(x, y, \sigma)) \cdot I(x, y) = L(x, y, k \cdot \sigma) - L(x, y, \sigma) \tag{3.11}$$

This function is particularly efficient to compute and in addition provides a close approximation to the scale-normalized Laplacian of Gaussian (Normalization of the Laplacian is required for true scale invariance and its maxima and minima produce the most stable image features).
The initial image so is incrementally convolved with Gaussians to produce images separated by a constant factor k in scale space. Adjacent image scales are then subtracted to produce the difference-of-Gaussian. In order to detect features, which correspond to the local maxima and minima of $D(x, y, \sigma)$, each sample point is compared to its neighbors in the current image and in the scale above and below. It is then selected only if it is larger than all of these neighbors or smaller than all of them. Once a keypoint candidate has been found, a fit to the nearby data for location, scale, and ratio of principal curvatures is performed in order to reject points that have low contrast (consequently sensitive to noise) or that are poorly localized along an edge. This is done with Taylor series expansion of scale space to get a more accurate location of extrema, while an approach similar to the one used by Harris detector is used to check if ratio of curvatures is below a certain threshold.

Once features are found, an invariant *descriptor* have to be associated to characterize them. An orientation is first assigned to each keypoint in order to achieve image rotation invariance: A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation

histogram with 36 bins covering 360 degrees is created. The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. A 16×16 neighbourhood around the keypoint is then taken and devided into 16 sub-blocks of 4×4 size. For each sub-block, an 8 bin orientation histogram is created. These are concatenated for a total of 128 bin values. Keypoint descriptor is then obtained representing this with a vector. In addition, several measures are taken to achieve robustness against illumination and 3D viewpoint changes. For a complete and exhaustive description of the method see [38].

SIFT detector and descriptor gives high quality features, with accuracy superior to almost all the other methods hereby presented. By the way, this high performances are paid with a really high computational effort needed both in terms of time and memory consumption which usually does not fit well for real time application. It must be also underlined that SIFT is a licensed software which cannot be used without permission.

3.2.0.5 *Speed Up Robust Features Detector and Descriptor*

Speed Up Robust Features (SURF) is a feature detector and descriptor developed by Herbert Bay, et al., and presented in [39]. Partially inspired by SIFT, it is specifically developed to be faster without loss of accuracy.
The SURF detector exploits integral images [1] and is based on an approximation of the Hessian matrix.
Given a point $\mathbf{x} = (x, y)$ in an image I, the Hessian matrix in $\mathbf{x}$ with scale σ is:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix} \tag{3.12}$$

where $L_{xx}(x, \sigma)$ is the convolution of the Gaussian second order derivative with the image I at point $\mathbf{x}$, and similarly for $L_{xy}(x, \sigma)$ and $L_{yy}(x, \sigma)$. This convolution can be efficiently approximated with box filters, which can evaluate really fast second order derivatives with integral images. Denoting this approximation with D_{xx}, D_{yy} and D_{xy}, weighting the rectangular regions, the following equation is obtained:

$$det(H_{approx}) = D_{xx} \cdot D_{yy} - (0.9 \cdot D_{xy})^2 \tag{3.13}$$

Thanks to the use of box filters and integral images, such filters can be applied at any size directly on the original image, without need of iteratively scale it. In order to localize features, a non maximum suppression in a neighbourhood of the selected pixel is applied. The

1 The entry of an integral image $I_\Sigma(\mathbf{x})$ at a location $\mathbf{x} = (x, y)$ represents the sum of all pixels in the input image I of a rectangular region formed by the point $\mathbf{x}$ and the origin, $I_\Sigma(\mathbf{x}) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i, j)$

maxima of the determinant of the Hessian are then interpolated in scale and image space.

To obtain SURF descriptor instead, first an orientation is extracted for each feature (for rotation invariance). This is done calculating Haar-wavelet response in x and y directions in a circular neighbourhood around the interest point which is then weighted with a Gaussian. Response is then represented in a vector and dominant orientation is estimated summing up all components in a sliding orientation window. For the extraction of the descriptor a square region oriented as the feature is constructed around it. This region is splitted into $4X4$ square sub-regions and for each one a few simple features are computed. Wavelet responses are computed for each sub-region and summed up to form the first entries of the descriptor. To bring information about polarity and intensity changes, sum of absolute values of the responses are also retained to complete the descriptor.

SURF descriptor and detector represent an absolutely valid alternative to SIFT, being much more faster and reaching almost the same accuracy. To speed up much more the process it also exist a version called U-SURF, which does not calculate orientation of the features (and so is not rotation invariant) that can be used in al the cases where the camera is rotating about only one axis. As SIFT, SURF is also given under license.

3.2.0.6 *BRIEF Descriptor*

To speed up matching and reduce memory consumption, the easiest way is to work with short descriptors.
BRIEF, developed in [40], uses binary strings to describe image patches and fasten descriptor extraction process. While this is usually done also in other state of the art algorithms, these usually do that by first compute the full descriptor and then shortening it. This is not computationally and timing efficient.
BRIEF directly builds a short descriptor by comparing intensities of pairs of points. A bit vector out of test responses is built after smoothing the image patch.
Test τ on a patch p of size $S \times S$ is defined as:

$$\tau(p; \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & if I(\mathbf{x}) < I(\mathbf{y}) \\ 0 & otherwise \end{cases} \tag{3.14}$$

where $I(\mathbf{x})$ is the pixel intensity in the smoothed version of I at $\mathbf{x} = (x, y)$. Choosing a set of n_d $\mathbf{x}, \mathbf{y}$ location pairs uniquely defines a set of binary tests. The BRIEF descriptor is then built as the *nd*-

dimensional bit string:

$$f_{n_d}(I) = \sum_{1 \leq i \leq n_d} 2^{i-1} \cdot \tau(I; \mathbf{x}, \mathbf{y}) \tag{3.15}$$

and when creating it the only choice to be made is the kernel used to smooth the image patch. Kernel is applied to increase stability and repeatability of the descriptor, being the test of 3.14 noise sensitive. The n_d tests location are set from an isotropic Gaussian distribution. For a complete description and analysis of the descriptor along with test and result see [40].

BRIEF descriptor is very efficient to compute and store in memory, outperforming other fast descriptors as SURF and U-SURF in terms of speed. It must be taken in consideration although it is not rotation invariant.

3.2.0.7 *Oriented FAST and Rotated BRIEF*

ORB is a feature detector and fast binary descriptor based on BRIEF and FAST which is rotation invariant and resistant to noise. Developed in 2011, it is specifically built to be fast and computationally low-cost for real time applications.
ORB detector works with a modified version of FAST, called oFAST, which retains only the best features within a multi-scale pyramid and adds the orientation component to them. Harris corner measure is first used to order FAST keypoints. For a target set of N keypoints, first a low threshold is used to get more than N points, then these are ordered according to their Harris measure and only the better N are retained. This process is repeated for each scale of a pyramid which is generated from the basic image in order to obtain multi-scale features. Corner orientation is then added by a simple measure exploiting the intensity centroid. Intensity centroid assumes that a corner's intensity is offset from its center.
Defining the moment of a patch as:

$$m_{pq} = \sum_{x,y} x^p \cdot y^p \cdot I(x, y) \tag{3.16}$$

The centroid may be found as:

$$C = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right) \tag{3.17}$$

Constructing a vector from the corner center to the centroid, then the orientation can be simply compute as:

$$\theta = atan \left(\frac{m_{0,1}}{m_{1,0}} \right) \tag{3.18}$$

To improve rotation invariance of this measure, moments are computed with x and y remaining within a circle of radius r equal to the patch size.

ORB descriptor is obtained from a modification of BRIEF, called rBRIEF. First BRIEF descriptor is steered according to orientation of keypoints. For any set of n binary test at location x_i, y_i the following matrix is defined:

$$S = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{bmatrix} \tag{3.19}$$

Using patch orientation θ and corresponding rotation matrix R_θ a steered version of S is computed:

$$S_\theta = R_\theta \cdot S \tag{3.20}$$

and the steered BRIEF operator 3.15 becomes:

$$g_n(p, \theta) = f_n(p) | (x_i, y_i) \in S_\theta \tag{3.21}$$

angle is discretized to increment of $2\pi/30$ and a lock up table of precomputed BRIEF patterns is constructed. Once BRIEF is oriented along keypoint direction, it loses his property of having each bit feature with high variance and mean near 0.5, which becomes more distributed. High variance makes a feature more discriminative, since it responds differentially to inputs. Another desirable property is to have the tests uncorrelated, since then each test will contribute to the result. To resolve all these, ORB runs a search among all possible binary tests to find the ones that have both high variance and means close to 0.5, as well as being uncorrelated.

ORB offers performance quite superior both than SIFT and SURF, being orders of magnitude faster with lower computational effort needed. Moreover, it is free from any patent and so can be used without any permission. The interested reader is encouraged to see [41] for a complete description of ORB along with test and comparison with SURF and SIFT.

3.3 FEATURE MATCHING AND FEATURE TRACKING

Once features have been extracted from an image, the objective is to find them in all the other consecutive incoming images from the camera in such a way that a correlation between features seen from different view is obtained. This correlation is fundamental because according to epipolar geometry, as described in Chapter 2, it allows to find rotation and translation which relate consecutive camera poses and thus allows to reconstruct the complete trajectory.

There are actually two ways to obtain this correlation. The first one is the so called *"feature tracking"*: Features are extracted only once in the first frame and then according to optical flow are tracked in consecutive images searching for them in a constrained zone around the previous position. Proceeding this way features number will soon or later drop (some feature will fail to be found in subsequent images after being propagated) and a new detection procedure shall be done to enrich again it to a defined level.
The second way to proceed is the so called *"feature matching"*. Feature matching is based on the use of the already introduced *descriptors*: Features are matched between couples of consecutive frames (that is matching features of image 1 with those of image 2, and then ones of image 2 with those of 3 and so on...) measuring how much two descriptor are similar to each other. If this measure is good then descriptor are the same, which means they are describing the same features and so a match is obtained. This is done iteratively until all the features in the two images have been matched.
In both cases erroneously tracked features or false match can be obtained. This is inevitable but at the same time a very conditioning fact for the whole pose estimation process. For this reason, a method as effective as possible to remove all the *outliers* and retaining only the *inliers* have to be implemented.

3.3.1 *Features Tracking*

The most classical way to track features in optical navigation is to exploit optical flow. Lukas-Kanade algorithm is one of the most diffused tool which do this.
As originally proposed in 1981, was an attempt to produce dense results (i.e. working on the whole image intensity). Applyed to a subset of the points in the input image, it became a popular method for feature tracking. The LK algorithm can be applied this way because it relies only on local information that is derived from some small window surrounding each of the points of interest. The disadvantage of using small local windows in Lucas-Kanade is that large motions can move points outside of the local window and thus become impossible for the algorithm to find. This problem led to development of the "pyramidal LK" algorithm, which tracks features starting from the highest level of an image pyramid (lowest detail) and working down to lower levels (finer detail). Tracking over image pyramids allows large motions to be caught by local windows.
Lukas-Kanade algorithm rests on three assumptions:

- *Brightness constancy*, a pixel from a scene does not change its appearance in consecutive frames.

- *Small movements*, image motion of a surface patch changes slowly in time
- *Spatial coherence*, neighboring points in a scene belong to the same surface and have similar motion.

Thanks to this assumptions it is possible to write the optical flow equation for a window surrounding the feature p, local image flow vector (V_x, V_y) must satisfy:

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\vdots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned} \tag{3.22}$$

where $q_1, q_2 ... q_n$ are the pixels inside the window, I_x, I_y, I_t are the image intensity first derivatives in space and time.
These equations can be written in matrix from $Av = b$ with:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \tag{3.23}$$

$$v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \tag{3.24}$$

$$b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix} \tag{3.25}$$

This system is overdetermined and LK algorithm solves this by least-squares minimization in the form $min \parallel Av - b \parallel^2$:

$$(A^T A)v = A^T b \tag{3.26}$$

which can be solved when $(A^T A)$ is invertible, and $(A^T A)$ is invertible when it has full rank (2), which occurs when it has two large eigenvectors. This happens in image regions that include texture running in at least two directions, which correspond to corners. $(A^T A)$ will have then the best properties when the tracking window is centred over a corner region in an image, which is actually the case when tracking features.
Large and non-coherent motions of the camera can anyway happen. In this case, the standard LK algorithm loses its reliability: a large window would be required to catch large movement, but the coherent motion assumption would be probably broken. This is why pyramidal LK takes place: algorithm tracks first over larger spatial scales using an

image pyramid and then refine the initial motion velocity assumptions by working down the levels of the image pyramid until it arrives at the raw image pixels.
A complete and exhaustive presentation of the Lucas-Kanade algorithm in its various forms and evolutions can be found in [42], for a more compact description see [43]. Despite being quite effective in tracking features across the flow of images, LK is computationally expensive and, moreover, expected motion of the camera have to be carefully taken into account in order to not violate the assumptions made to build the algorithm, which can lead to bad results. It should also be underlined the fact that LK does not make use of descriptors, which means they have not to be extracted, and that features detection process has not to be done on each image, which can result in a major save of time.

3.3.2 *Features Matching*

Feature matching consists, given two images along with their respective keypoints and descriptors extracted, in correlating as precisely as possible (i.e. avoiding false match) each keypoint in the first image with its corresponding one in the second image. This process can be done by comparing all the descriptors of the two images in terms of a distance, which can be different depending on the kind of descriptor extracted. Each feature in the first image is then matched with the corresponding feature in the second image which has the lowest distance. The simplest approach is the so called "Brute-Force matching"; each descriptor from the first set of features is matched with all other feature descriptors in the second set. Matches are then retained in order of distance and only the lowest ones are taken.
Distance used are the classical *L2 Norm* or the Hamming distance. *L2 Norm* can be expressed as:

$$d(D_1, D_2) = \sqrt{\sum_{i=1}^{n} (D_{1i} - D_{2i})^2} \tag{3.27}$$

in which D_1, D_2 are the descriptors and d the distance between them. Euclidean distance perfectly fists for SIFT and SURF.
Hamming distance is used instead for binary string based descriptors, being a measure in two string of the same length of the number of position in which corresponding symbols are different:

$$e.g. \quad dH(D_1, D_2) = dH(10\mathbf{11}101, 10\mathbf{01}\mathbf{0}01) = 2 \tag{3.28}$$

This kind of measure works optimally for ORB and BRIEF descriptors, and is particularly fast to compute with respect to the *L2 Norm*.
In using feature matching it has to be considered the fact that not all

the features extracted in one image can always be found in the second one and also that, for similar textured objects, more features can be very similar to each other. This can result in lot of false matches which have to be rejected. Different methods to asses this problem exist and will be later described.
Despite being quite simple and maybe at first glance too rude, Brute-Force feature matching if well implemented is as quite effective as a tracking method like Lucas-Kanade one, but with the advantage of being several times faster. Complexity of course grows proportionally with the number of features used and detection process have to be repeated on each image. This have to be taken into account, and consequently a save of time is not straightforward.

3.3.3 *Feature Detection and Matching Comparison*

Before starting to analyse in detail all the aspects that lead to the choice of the feature detection and description algorithm and which method to use to recognize extracted features between frames, it is important to underline the properties that the whole algorithm block must have to efficiently work in its application field. The scenario is of course the one in which a satellite has a descending trajectory to an unknown surface of a celestial body.
During a possible real time application satellite will have a downward pointing camera with features changing scale as long as the altitude will decrease, with possible illumination changes depending on the trajectory. Moreover, due to maneuvers made to follow correctly the defined path, different sudden motions may occur resulting in sudden new images maybe with the same features seen from different angles. All of these represent a challenging situation for any of the detectors presented, and in particular for matching or tracking features. It is in this context that robustness of the descriptors or a LK pyramidal approach becomes important to recognize the same features across the incoming frames.
Considering this, the following requirements can be set and have to be met by the feature detector and descriptor:

- Rotation invariance
- Resistance to illumination changes
- Scale invariance

Moreover it must be kept in mind that the processing power available on board will be constrained and not so high, which gives rise to another requirement:

- Computationally low cost

Considering what defined above, starting from feature extraction and description, the choice has fall on the Oriented FAST and Rotated BRIEF (ORB). ORB outperforms all the other algorithms both in terms of computational time and efficiency, satisfying all the requirements outlined. While SURF represent a valid alternative, it is anyway too expensive computationally speaking, while SIFT offers good performance but at a too high computational cost for a real time application. Harris and Shi-Tomasi work well for feature detection, but are anyway too expensive and less performant than FAST and ORB (they are not scale invariant) and a descriptor have to be coupled to them.
FAST itself can be a good choice for detection, but without any improvement (which is instead given by oFAST in ORB) it is not nor scale nor rotation invariant.
Again, both SURF and BRIEF descriptor are efficient, but outperformed by rBRIEF implemented in ORB; the first only in terms of computational time, while the second because of its sensitivity to rotations. All of these gives a great push towards the choice of ORB, which is also used by many state of the art SLAM algorithms, as the already described ORB-SLAM [32].

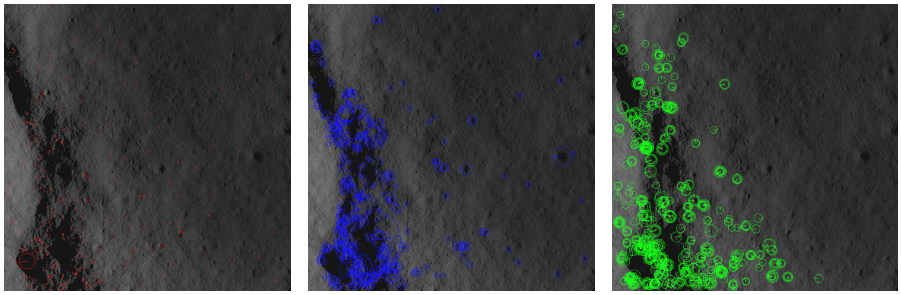
In order to better asses the choice, test have anyway been made on ORB, SIFT, SURF and FAST with images from a lunar landing trajectory. Very preliminary tests showed that Harris and Shi-Tomasi performances are poorer than FAST and they have been excluded from this assessment.
Test have been made this way: Algorithms have been setted to extract the same number of features (500), with SURF, SIFT and ORB, also working with the same number of pyramid layers and octave layers (four and three respectively). Two generic successive images from the trajectory have been selected, and features extracted have been matched with the Brute Force approach (as described in section 3.3.2) and filtered according to the method proposed by [38] in order to retain only inliers. Total number of features matched and time required by the process have then been compared. Table3.1 shows the results obtained. Being FAST only a detector which has to be associated with a descriptor in order to do matching, only its computational time has been considered. Tests have been run on a single CPU core, exploiting ORB, FAST, SIFT and SURF algorithms already implemented in OpenCV-3.1.0 *features2d* module.

As it is possible to see, SURF obtains the largest number of matches, immediately followed by ORB, while SIFT retains the lowest number. Considering the computational time instead, ORB clearly outperforms both SIFT and SURF by order of magnitudes as expected, fully justifying its choice for a real time application even at the cost of having less matches.

Method	Features matched	Detection & Description [ms]	Matching [ms]
FAST	*none*	1	*none*
SIFT	109	469	30
SURF	272	286	15
ORB	180	23	11

Table 3.1: *Feature detection and description performance comparison*

A speech exception is made for FAST, which is by far the fastest method for detection but does not extract a descriptor and does not build a pyramid for scale invariance of the features, both tasks which are time consuming if added and that do not justify its choice instead of ORB (which uses anyway a modified version of FAST).



(a) **(b)** **(c)**

Figure 3.2: *(a)SIFT, (b)SURF, (c)ORB comparison.* 500 *features extracted on a* 4 *level pyramid.*

Fig. 3.2 shows examples of SIFT, SURF and ORB features extracted with a 4 scale level pyramid. Each circle is centred on a feature with size corresponding to the scale level at which it has been detected, with orientation represented by the radius direction. It can be seen how features tend to concentrate on craters and rougher areas as could be expected. Concentric circles mean that the same feature have been found on different scales.

For what concerns the choice between feature matching and feature tracking, a similar approach has been considered. The Lukas-Kanade algorithm has been run on the 500 ORB features extracted, and its computational time along with the number of features tracked has been compared to previous results obtained with matching. OpenCV-3.1.0 *video* library comes along with a sparse iterative version of the Lucas-Kanade optical flow in pyramids [44], which has been exploited with a 21×21 pixels search window for a 3 scale levels pyramid. Table3.2 shows the results obtained. As can be observed, Lucas-Kanade algorithm tracks almost all the features from which it has been initialised in a time almost doubled with respect to the one

Pyramidal Lucas-Kanade	
Features tracked	436
Elapsed time [ms]	23

Table 3.2: *Pyramidal LK algorithm performance*

needed for matching the same ORB features. It has to be considered by the way that the process to match features between images requires features extraction at each step, while LK optical flow approach requires features extraction only at the first step and eventually when the number of features tracked goes below some fixed threshold. This basically shows that the whole process of feature detection and match costs 33 ms for each new frame, matching correctly less than a half of the features. Detection and tracking with LK algorithm instead cost 46 ms for the first iteration, 23 ms for the others, and tracks more than the double number of features with respect to the matching approach. The choice is consequently fell on the Lucas-Kanade features tracking approach, which basically gives more features correspondences per frame at a lower computational cost. To ensure the fulfilment of the hypothesis required by the algorithm, a frame rate sufficiently high with respect to the camera motion should be taken into account. An example of ORB features tracked with LK algorithm is given in Fig. 3.3, with lines representing tracked feature position.

In conclusion, the approach selected is to exploit ORB features detection and description along with Lucas-Kanade feature tracking algorithm. ORB Features are first extracted from an initial frame and then tracked in subsequent frames with pyramidal LK.

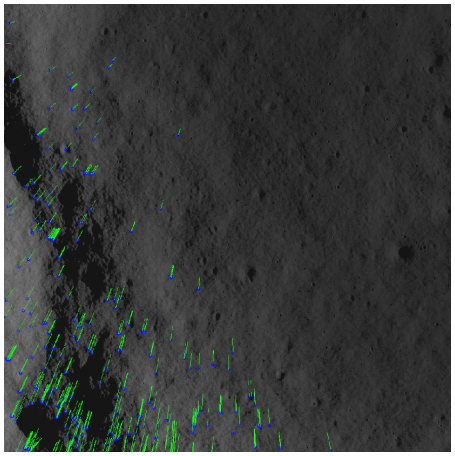


Figure 3.3: *Feature tracking with pyramidal Lucas-Kanade algorithm implementation. ORB features are represented by green dots while lines represent tracked feature future position*

3.4 MOTION ESTIMATION

Motion estimation is the computational core of the navigation algorithm and consists in finding at each step of the process the camera motion between the current and previous image.
There are basically three possibilities for estimating motion between two cameras: Exploiting 2*D*-2*D* frame to frame features correspondences, 3*D* to 2*D* (map to frame) correspondences, or 3*D* to 3*D* correspondences. Discarding the third case which is computationally demanding and reported to be less efficient than the other two [15], the second method requires anyway a 3*D* map of points which has to be first initialized. This can be done only knowing at last the pose between the two frames, which can only be retrieved exploiting the first method (it is impossible to obtain 3*D* position of seen features between two frames from scratch without knowing motion between them). For this reason, focus will be given first on 2*D*-2*D* motion estimation, to continue then with 3*D*-2*D* one.

3.4.1 *Frame-to-Frame Motion Estimation*

2*D* to 2*D* motion estimation works with extracted features from a pair of images. Once those are tracked, a set of correspondences is available between the two frames. Feature correspondences are related by epipolar geometry, as seen in Chapter 2, which completely describes the structure of the two consecutive camera poses and of the world points seen by them. This description is enclosed inside the *essential matrix E* for calibrated cameras, while *fundamental matrix F* holds for uncalibrated ones. Once one of this two matrices is available, motion can be retrieved up to scale by a simple algebraic decomposition.
For particular cases in which the viewed scene is planar, feature correspondences may be also related by an homography matrix H, from which motion can be again extrapolated up to scale.

3.4.1.1 *Normalized 8-point Algorithm*

The normalized 8-point algorithm developed by Richard Hartley [1] is one of the simplest and widely diffused method to obtain a fundamental matrix F from a set of 8 features correspondences without knowing the intrinsic parameters of the camera. It is actually an improvement of the 8-point algorithm first developed in 1981 by Christopher Longuet-Higgins for the estimation of the essential matrix.
Algorithm simply involves the construction and least square solution of a set of linear equations, and uses normalized input data for better conditioning of the problem and stability of the result. The suggested normalization is a translation and scaling of each image so that the

centroid of the reference points is at the origin of the coordinates and the RMS distance of the points from the origin is equal to $\sqrt{2}$.

From Section 2.3 it is known that for the fundamental matrix F the following constraint is always valid:

$$x_i^T F x_i = 0 \tag{3.29}$$

This gives rise to a set of linear equations of the form $Af = 0$. If A has rank 8, then it is possible to solve for f up to scale, and this is the classical way of implementing the 8-point algorithm. In the case where the matrix A has rank seven, it is still possible to solve for the fundamental matrix by making use of the singularity constraint.
The matrix F found by solving the set of linear equations will not in general have rank 2 (i.e. will not be exactly singular due to presence of noise in the extracted features x, x'), consequently, steps to enforce this constraint are taken. The most convenient way to do this is to correct the matrix F found by the SVD (see Appendix A.1) solution from A. Matrix F is replaced by the matrix F' that minimizes the Frobenius norm $|F - F'|$ subject to the condition $detF' = 0$. In the normalized 8-point algorithm singularity constraint is enforced solving directly for F' using SVD, which is simple and rapid. Exact matrix F is then finally retrieved denormalizing matrix F' obtained from the normalized data. This method is direct and efficient to compute, but has anyway a major drawback which is associated to F matrix itself. Independently from the method used in fact, fundamental matrix suffers from the so called "planar structure degeneracy". If the observed points from the camera lies on a plane (as it could be the case during a landing phase, where surface is seen from far away and appears to be flat) F is determined only up to three degree of freedom, which leads to a three-parameter family of possible fundamental matrices F (one of the parameters accounts for scaling the matrix so there is only a two-parameter family of homogeneous matrices). This ambiguity is not solvable and is a consequence of the fact that the camera intrinsics (K matrix) are not included in F.

3.4.1.2 *Five-Point Algorithm*

Given a set of $2D$ feature correspondences, the most efficient solution to estimate the essential matrix E is represented by the Five-Point algorithm developed by David Nister and presented in [45].
The problem is to find the possible solutions for relative camera pose between two calibrated views given five corresponding points. Algorithm consists in computing the coefficients of a tenth degree polynomial in closed form and subsequently finding its roots. Only relative positions of the points and cameras can be recovered, overall scale of the configuration represents an ambiguity and can never be

extrapolated solely from images.
Image points are represented by homogeneous 3-vectors x and x' in the first and second view respectively. World points are represented as homogeneous 4-vectors X. Perspective view is represented by the 3×4 camera matrix P, with $P = K \cdot [R|t]$ for a finite projection centre (with K intrinsic matrix of the camera and R and t rotation matrix and translation vector respectively), from which it is known $x = P \cdot X$ up to scale, as explained in Chapter 2.

Considering five point correspondences, the constraint $x'^T E x = 0$ can be written as:

$$\widetilde{x}^T \cdot \widetilde{E} = 0 \tag{3.30}$$

where:

$$\widetilde{x} = \begin{bmatrix} x_1x'_1 & x_2x'_1 & x_3x'_1 & x_1x'_2 & x_2x'_2 & x_3x'_2 & x_1x'_3 & x_2x'_3 & x_3x'_3 \end{bmatrix}^T \tag{3.31}$$

$$\widetilde{E} = \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^T \tag{3.32}$$

By stacking the vectors $\widetilde{x}^T$ for all five points, a 5×9 matrix is obtained. Four vectors $\widetilde{X}, \widetilde{Y}, \widetilde{Z}, \widetilde{W}$ that span the right null space of this matrix are then computed with QR-factorisation. The four vectors correspond directly to four 3×3 matrices X, Y, Z, W and the essential matrix must be of the form:

$$E = xX + yY + zZ + wW \tag{3.33}$$

for some scalars x, y, z, w up to a common scale factor, from which it is consequently assumed $w = 1$.
Eq. 3.33 is enforced through the two following constraints:

$$det(E) = 0 \tag{3.34}$$

$$EE^T E - \frac{1}{2} trace(EE^T)E = 0 \tag{3.35}$$

and Gauss-Jordan elimination with partial pivoting performed. This way an equation system is obtained, from which a tenth degree polynomial is solved for x, y and z roots. Essential matrix is then obtained from Eq. 3.33.
This algorithm is thought to be used in conjunction with preempitve RANSAC in order to be more robust to the presence of outliers (i.e. false matches, wrong tracking) between the features. A number of random samples are taken, each containing five points. Five-point algorithm is applied to each sample and thus a number of hypotheses are generated. The best hypothesis is then chosen according to a robust measure over all the tracks and is in the end iteratively refined.

One of the most important features of the five-point algorithm is that it works for any kind of configuration of the features considered, avoiding the planar structure degeneracy. According to [45], this algorithm results to be efficient both in terms of accuracy and speed. In

comparison to other state of the art methods, despite being weaker for determining rotations, five-point works optimally for sideways motion and similarly for forward motion, although slightly worse when baseline between the two cameras is very small. Using the Essential matrix also removes the projective ambiguity which arises using fundamental matrix F, and provides a metric (or Singular) reconstruction, which means the 3D points are true up to scaling alone, and not up to a projective transformation.

3.4.1.3 *R and t Recover from Essential Matrix*

Once essential matrix from two frames is retrieved, camera pose in terms of rotation R and translation t are obtainable. Process is described in detail both in [1] and [45]. In case fundamental matrix F is used, then one can simply retrieve essential matrix with $E = K^T F K$.
Camera matrix can be retrieved up to scale and a four-fold ambiguity. Assuming first camera as $P = [I|0]$, to compute the second camera matrix P', E is factorized into the product SR of a skew-symmetric and a rotation matrix. Being the SVD (see Appendix A.1) of E $Udiag(1,1,0)^T V^T$ there are two possible factorization $E = SR$:

$$S = UZU^T \qquad R = UWV^T \qquad or \qquad UW^T V^T \tag{3.36}$$

From this factorization translation may be determined up to scale: $t = U(0,0,1)^T = \mathbf{u}_3$, last column of U. The sign of E, and consequently of t, cannot be determined. This is why there are four possible choices of the camera matrix P', based on the two possible choices of R and two possible signs of t.
The difference between the first two solutions is simply that the direction of the translation vector from the first to the second camera is reversed. For what concerns the other two it can be demonstrated that is a rotation through 180° about the line joining the two camera centres. Two solutions related in this way are known as a twisted pair.
In order to chose between the possible solutions the so called *cheriality check* is done. A reconstructed $3D$ point $\mathbf{X}$ will be in front of both cameras in one only of these four cases. Consequently, testing with a single point to determine if it is in front of both cameras is sufficient to decide between the four different solutions for the camera matrix P'. Fig. 3.4 shows the different solutions.
In a real application, since a single point may be an outlier (e.g. arising from a false match) or a too far point which can be poorly triangulated, all the $2D$ features are exploited and the solution with highest percentage of points in front of the camera is taken as the good one.

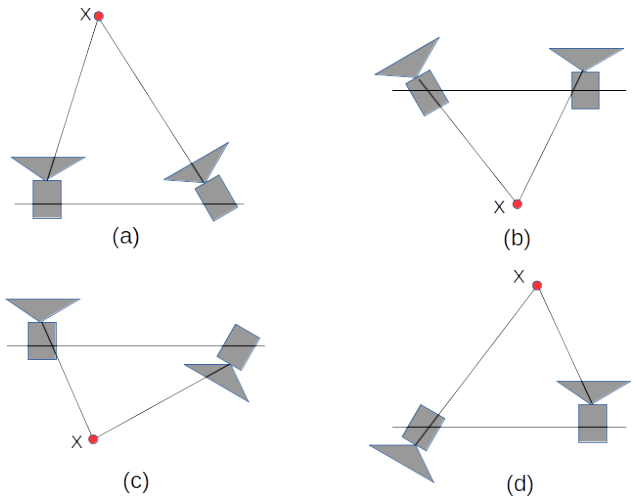


Figure 3.4: *Four possible solutions for pose reconstruction from E. Between left and right side there is a baseline reversal, while between top and bottom second camera is rotated by* 180°.

3.4.1.4 *Homography Estimation*

For particular cases where the scene viewed by the camera is planar, it exist the possibility to exploit homography to reconstruct the relation between the features seen in the first image and the second image and then retrieve motion. Planarity of the scene is of course a restrictive requirement which is difficult to verify for most of the real applications; by the way, two considerations shall be done.
First, the scenario for a spacecraft landing, as already described, is the one in which landing surface will be first seen from far away, proceeding then nearer and nearer as the altitude decrease. In this kind of situation, scene seen from the camera during the first phases (high altitude) will appear as almost flat, thus verifying the planarity condition.
Secondly, assuming the algorithm to be used within RANSAC, one can rely on the fact that if some plane inside the image exist, then if its corresponding features are used by RANSAC as best model, it is still possible to retrieve an homography even if observed scene is not completely planar. This condition can easily occur during a planetary landing (e.g. suppose to consider a landing on the Moon: despite craters, boulders and maybe some sort of canyons, there is anyway a high probability to see some portion of flat zones in the images, particularly when altitude is still quite high).
Homography between two set of corresponding features, if those are planar, can be obtained quite easily and solved with the classical DLT algorithm.
Given a set of four correspondences $x' \leftrightarrow x$ expressed as homogeneous, homography transformation gives:

$$x_i^{'} = Hx_i \tag{3.37}$$

The equation may be expressed in terms of the vector cross product as $x_i^{'} \times Hx_i = 0$. This form enables a simple linear solution for H to be derived. If the j-th row of the matrix H is denoted by h_j^T, it is possible to write:

$$Hx_i = \begin{bmatrix} h_1^T x_i \\ h_2^T x_i \\ h_3^T x_i \end{bmatrix} \tag{3.38}$$

Writing $x_i^{'} = x_i^{'}, y_i^{'}, w_i^{'T}$, cross product can be written as:

$$x_i^{'} \times Hx_i = \begin{bmatrix} y_i^{'} h_3^T - w_i^{'} h_2^T x_i \\ w_i^{'} h_1^T - x_i^{'} h_3^T x_i \\ x_i^{'} h_2^T - y_i^{'} h_1^T x_i \end{bmatrix} \tag{3.39}$$

This gives a set of three equations in the entries of H, which can be written in the form $A_i h = 0$, with h 9-vector made up of the entries of the matrix H. Although there are three equations, only two of them are linearly independent (since the third row is obtained, up to scale, from the sum of x_i times the first row and y_i times the second). Thus each point correspondence gives two independent equations in the entries of H:

$$\begin{bmatrix} 0^t & -w_i^{'} x_i^T & y_i^{'} x_i^T \\ w_i^{'} x_i^T & 0^T & -x_i^{'} \mathbf{x}_i^T \end{bmatrix} \tag{3.40}$$

Given a set of four such point correspondences, a set of equations $Ah = 0$ is obtained, where A is the matrix of equation coefficients built from the matrix rows A_i contributed from each correspondence, and h is the vector of unknown entries of H. If more than four point correspondences are given, then the set of equations $Ah = 0$ is over-determined. If the position of the points is exact then the matrix A will have rank 8, a one dimensional null-space, and an exact solution h exist. This is not the case if the measurement of image coordinates are inexact. For this reason, instead of demanding an exact solution, and an approximate solution is searched for, namely by minimization. To avoid the solution $h = 0$ an additional constraint is required. Generally, a condition on the norm is used, such as $||h|| = 1$.
OpenCV library comes with an already implemented algorithm to calculate homography between two planes using RANSAC in order to improve the solution in case of outliers. Many different random subsets of points are used to estimate H along with a quality measurement, the best subset is then used to produce the initial H estimate which is further refined with the Levenberg-Marquardt method (see Appendix A.2 for details) to reduce reprojection error.

This algorithm is quite efficient and robust to the presence of outliers thanks to the implementation of RANSAC, but relies on the fact that at least a subset of features correspondences must lie on a plane to

give a meaningful solution. Anyway, taking in consideration what stated in the beginning of the section, this method results attractive. For a deeper understanding of the homography projective transform and related estimation algorithm, consult [1] and [46].

3.4.1.5 *R and t Recover from Homography Matrix*

Decomposing an homography matrix in order to obtain pose of the camera between two consecutive frames in terms of rotation R and translation t is not as straight forward as doing the same from the essential matrix. Different numerical procedures and analytical solutions exists to solve this problem. Hereafter an analytical solution proposed in [46] and exploited by the OpenCV library is explained.
The method provides analytical expressions for the solutions of the problem, obtaining R, t and n (normal with respect to the plane) directly as a function of H. Four different solutions are obtained, among which two are basically the opposite of the other. There are actually two ways to proceed, one way is to calculate first the normal vector n and then R and t while the second is to do the opposite calculating R and t first. Only the first method is given here. Since the whole procedure is quite long and tedious, only a brief outline is proposed.

A matrix S is first defined from H to simplify computations:

$$S = H^T H - I = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{bmatrix} \tag{3.41}$$

S is singular and so condition $det(S) = 0$ can be imposed to obtain s_{33} as a function of the other S components.
Denoting the opposites of the two-dimension minors of this matrix as $M_{s_{ij}}$, following relations holds:

$$M_{s_{12}} = \epsilon_{12}(\sqrt{M_{s_{11}}}\sqrt{M_{s_{22}}}) \tag{3.42}$$

$$M_{s_{13}} = \epsilon_{13}(\sqrt{M_{s_{11}}}\sqrt{M_{s_{33}}}) \tag{3.43}$$

$$M_{s_{23}} = \epsilon_{23}(\sqrt{M_{s_{22}}}\sqrt{M_{s_{33}}}) \tag{3.44}$$

with $\epsilon_{ij} = sign(M_{s_{ij}})$ S may be written in terms of R, t and n the following way:

$$S = (R^T + nt^T)(R + tn^T) - I = R^T tn^T + nt^T R + nt^T tn^T \tag{3.45}$$

introducing x and y defined as follows:

$$x = \frac{R^T t}{||t||} \tag{3.46}$$

$$y = ||t||n \tag{3.47}$$

S can be written as:

$$S = xy^T + yx^T + yy^T \tag{3.48}$$

which is linear in x. Two sets of linear equations are obtained from S matrix written this way, which are solved for x and y. Using the given formulation, the four valid solutions of the homography decomposition problem are those corresponding to the choice of the minus sign for a coefficient which arise solving the linear set of equations. Denoting the four different solutions as:

$$\begin{matrix} R_a, t_a, n_a \\ R_b, t_b, n_b \\ R_{a-}, -t_{a-}, -n_{a-} \\ R_{b-}, -t_{b-}, -n_{b-} \end{matrix} \tag{3.49}$$

depending on which way coefficients of the linear system are rearranged, different way of writing the solutions arises, and this is exploited to avoid cases where zeros appear at denominator of a fraction. One of the possible way to obtain the normals n_a and n_b is:

$$n_a = \begin{bmatrix} h_1^T h_2 + \sqrt{(h_1^T h_2)^2 - (||h_1|| - 1)^2(||h_2|| - 1)^2} \\ (||h_2|| - 1)^2 \\ h_2^T h_3 + \epsilon_{13}\sqrt{(h_2^T h_3)^2 - (||h_2|| - 1)^2(||h_3|| - 1)^2} \end{bmatrix} \tag{3.50}$$

$$n_b = \begin{bmatrix} h_1^T h_2 + \sqrt{(h_1^T h_2)^2 - (||h_1|| - 1)^2(||h_2|| - 1)^2} \\ (||h_2|| - 1)^2 \\ h_2^T h_3 - \epsilon_{13}\sqrt{(h_2^T h_3)^2 - (||h_2|| - 1)^2(||h_3|| - 1)^2} \end{bmatrix} \tag{3.51}$$

Where $h_{1,2,3}$ are columns of the homography matrix H. Expressions for the translation vector are then obtained from the normals as:

$$t_a = \frac{||n_a||||n_b||}{2s_{22}} n_b - \frac{||t_e||^2}{2} n_a \tag{3.52}$$

$$t_b = \frac{||n_b||||n_a||}{2s_{22}} n_a - \frac{||t_e||^2}{2} n_b \tag{3.53}$$

where:

$$t_e = 2 + trace(S) - 2\sqrt{1 + trace(S) - M_{s_{11}} - M_{s_{22}} - M_{s_{33}}} \tag{3.54}$$

Finally, expression for R may be then simply retrieved as:

$$R = H\left(I - \frac{1}{1 + x^T y} xy^T\right) \tag{3.55}$$

Once the set of four solution is obtained, the right decomposition is chosen with cheiriality check as done for the essential matrix E, controlling that a triangulated feature X_i stands in front of both cameras.

3.4.2 *Map-to-Frame Motion Estimation*

Motion of a camera can be retrieved also with respect to a known map, comparing observed features on the images with ones in the map database. Thus, if a sparse map of $3D$ points of the observed environment is available, it is possible to match those with the $2D$ features currently seen in the image at each step and obtain pose of the camera relative to the map. This is the so called *"Perspective-n-point problem"* (PnP), which is namely the problem of estimating the pose of a calibrated camera given a set of n $3D$ points in the world and their corresponding $2D$ projections in the image.
Considering a Visual Odomentry like implementation of the navigation algorithm and considering the given requirements, the possibility of exploiting a pre-existent map built from a known image database is not considered as an option for this work. Therefore the only possibility available to use a PnP approach is that a map of the observed environment is built in real time during camera exploration. The simplest way to do this is to exploit features correspondences between two frames along with relative camera pose, and triangulate those features to obtain a $3D$ sparse map of the observed environment. Once the map is available, then PnP problem can be solved and pose of the camera for successive frames retrieved.
A map built in this way will not be anyway enough precise for motion tracking due to the presence of different error sources, which affects quality of the PnP results. Consequently, a dedicated process to take care of map accuracy and to eliminate bad points while continuously introducing new ones during exploration have to be considered.

3.4.2.1 *3D Sparse Map: Triangulation*

Supposing known features correspondences between two frames ($\mathbf{x} \leftrightarrow \mathbf{x}'$) and their relative calibrated camera pose, the problem is to find position of a point in space from the intersection of the two rays starting from camera centre and passing to the known position of the features on the image as shown in Fig. 3.5.

When noise is present, this two rays will not intersect and an optimal point of intersection has to be found.
To expound the problem, in the following a single $3D$ point $\mathbf{X}$ visible on two image is considered. The two camera matrices P and P' corresponding to these images are supposed known along with the fundamental matrix F. $\mathbf{x}$ and $\mathbf{x}'$ are the projections of $\mathbf{X}$ on the images. Knowing a complete set of $2D$ feature correspondences, it is then possible to apply the triangulation procedure on the whole set and obtain a sparse map of $3D$ points representing the observed environment by the camera.

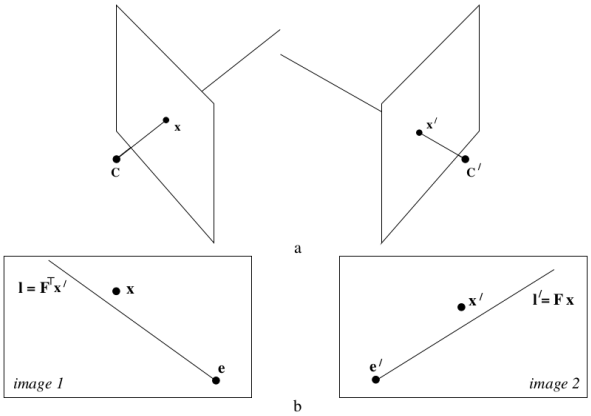


Figure 3.5: *Representation of the triangulation problem from [1]. When noise is present rays for corresponding image points do not intersect.*

LINEAR TRIANGULATION METHOD

Let's first consider the easiest method to triangulate points, the linear one presented in [1], which for an ideal case in which geometric relations are perfectly verified gives an exact solution for the estimated $3D$ point. This is however not the case for real applications in presence of noise, and hence solution found as here presented will not be optimal. Linear triangulation problem can be set similarly to the homography estimation problem of Section 3.4.1.4, and can therefore be solved with the DLT method.
Concept is that a measurement $\mathbf{x} = P\mathbf{X}$, $\mathbf{x}' = P'\mathbf{X}$ is available for each image, and these can be combined to obtain a system of the form $A\mathbf{x} = 0$. A cross product is first applied to obtain three equations for each image point, two of which are linearly independent; this is achieved, considering for example the first image, making $\mathbf{x} \times (P\mathbf{X}) = 0$, which leads to:

$$\begin{aligned} x(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{1T}\mathbf{X}) &= 0 \\ y(\mathbf{p}^{3T}\mathbf{X}) - (\mathbf{p}^{2T}\mathbf{X}) &= 0 \\ x(\mathbf{p}^{2T}\mathbf{X}) - y(\mathbf{p}^{1T}\mathbf{X}) &= 0 \end{aligned} \tag{3.56}$$

with x, y image coordinates and $\mathbf{p}^{iT}$ rows of P. Equation of the form $A\mathbf{X} = 0$ is then composed:

$$\begin{bmatrix} x\mathbf{p}^{3T} - \mathbf{p}^{1T} \\ y\mathbf{p}^{3T} - \mathbf{p}^{2T} \\ x'\mathbf{p}'^{3T} - \mathbf{p}'^{1T} \\ y'\mathbf{p}'^{3T} - \mathbf{p}'^{2T} \end{bmatrix} \mathbf{X} = 0 \tag{3.57}$$

with two equations included from each image. When more than one correspondence is used (triangulating more points), system becomes overdetermined and solution of the homogeneous set of equations can be obtained as the unit singular vector corresponding to the smallest singular value of A.

This method is quite simple to implement and gives reliable results, but is quite sensible to noise and to be safely implemented one has to be sure to use the most refined data as possible has input. To gain robustness against noisy data, an alternative approach is to use an iterative solution of this linear system with adaptively weighted equations as shown in [47].

OPTIMAL TRIANGULATION METHOD

A more efficient solution to robustly triangulate points which takes into account the fact that under the presence of noise the two rays projected from the image points do not intersect is the one proposed in [47, 1]. Optimal triangulation finds the global minimum of a cost function using a non-iterative algorithm which reduces the problem to the solution of a 6^{th} order polynomial.
Two rays for a match of corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$ will met in space if and only if condition $\mathbf{x}'F\mathbf{x} = 0$ is satisfied. This usually not happens for real images, where errors in location of the features in the images occurs due to different reasons (camera distortion itself, inaccurate detection...). It is then common to assume that images are subject to Gaussian noise which displaces the features from their correct location. In reality correct values of the corresponding image points should be $\hat{\mathbf{x}} \leftrightarrow \hat{\mathbf{x}}'$ lying close to the measured points $\mathbf{x}, \mathbf{x}'$ and satisfying equation $\hat{\mathbf{x}}'F\hat{\mathbf{x}} = 0$ exactly. With the optimal method points that minimize the following cost function are therefore searched:

$$d(\mathbf{x}, \hat{\mathbf{x}})^2 + d(\mathbf{x}', \hat{\mathbf{x}}')^2 \tag{3.58}$$

Once points $\mathbf{x}, \mathbf{x}'$ which minimize Eq. 3.58 are found, the previously explained linear triangulation method can be used to retrieve $\mathbf{X}$.
According to epipolar geometry, any pair of points satisfying the epipolar constraint must lie on a pair of corresponding epipolar lines in the two images. Optimum point $\hat{\mathbf{x}}$ lies on an epipolar line l and $\hat{\mathbf{x}}'$ lies on corresponding l', and any other point lying on l or l' will also satisfy the constraint. This is also true for the point $\bar{\mathbf{x}}$ on l which is closest to the measured point $\mathbf{x}$, and for the correspondingly defined point $\bar{\mathbf{x}}'$ on l'. Of all pairs of points on the lines l and l', the points $\bar{\mathbf{x}}$,$\bar{\mathbf{x}}'$ minimize the sum of Eq. 3.58. It follows $\hat{\mathbf{x}}' = \bar{\mathbf{x}}'$ and $\hat{\mathbf{x}} = \bar{\mathbf{x}}$, and it is consequently possible to write $d(\mathbf{x}, \hat{\mathbf{x}}) = d(\mathbf{x}, l)$, where d represents the perpendicular distance from the point $\mathbf{x}$ to the line l. A similar expression holds for $d(\mathbf{x}', \hat{\mathbf{x}}')$, and minimization can be then reformulated as follows:

$$d(\mathbf{x}, l)^2 + d(\mathbf{x}', l')^2 \tag{3.59}$$

Supposing neither of the two considered points correspond to the epipole, which means point in space does not lie on the line joining camera centres, then analysis is simplified by applying a rigid

transformation to each image in order to place both points $\mathbf{x}$ and $\mathbf{x}'$ at the origin $(0,0,1)^T$. Epipoles are placed on the x-axis at points $(1,0,f)^T = (1,0,f')^T$ respectively. Applying this has no effect on Eq. 3.59. Considering an epipolar line in the first image passing through $(0,t,1)^T$ and the epipole $(1,0,f)^T$ expressed as $l(t)$, vector representing this line is given by $(0,t,1) \times (1,0,f) = (tf,1,-t)$, so the squared distance from the origin is:

$$d(\mathbf{x}, l(t))^2 = \frac{t^2}{1+(tf)^2} \tag{3.60}$$

corresponding epipolar line in the other image is:

$$l'(t) = F(0,t,1)^T = (-f'(ct+d), at+b, ct+d)^T \tag{3.61}$$

with the square distance from the origin as:

$$d(\mathbf{x}', l'(t))^2 = \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2} \tag{3.62}$$

total square distance is then given by:

$$s(t) = \frac{t^2}{1+(tf)^2} + \frac{(ct+d)^2}{(at+b)^2 + f'^2(ct+d)^2} \tag{3.63}$$

and the problem becomes to find the minimum of this function, which can be done using techniques of elementary calculus, computing derivative of $s(t)$ and putting it equal to zero. Polynomial has degree 6, so 6 roots are possible. Solving for this polynomial leads to find the optimal point correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$ between the two images, which can be then safely given in input to the linear triangulation method to find the $3D$ point X.

3.4.2.2 *Perspective-n-Pose Problem*

Once a sparse map of $3D$ features has been built and correspondences with $2D$ features observed in the current image are set, Perspective-n-Pose problem can be solved. One of the most efficient solution of the problem is given by EPnP, an algorithm proposed by Lepetit, et. al., and presented in [48].
EPnP gives a non-iterative solution to the PnP problem, estimating the pose of a calibrated camera from n $3D$ to $2D$ feature correspondences. Method is applicable for all $n \geq 4$ and can handle efficiently both planar and non-planar configurations.

The idea behind EPnP is to write coordinates of the n $3D$ points as a weighted sum of four virtual control points. This reduces the problem to estimating the coordinates of the control points in the

camera referential, which is done efficiently expressing these coordinates as weighted sum of the eigenvectors of a 12×12 matrix, and solving a small constant number of quadratic equations to pick the right weights. With a set of $3D$ coordinates in world reference frame and their respective projection on $2D$ image, algorithm aims then at retrieve the unknown orientation and translation of the camera as the Euclidean motion which aligns both sets of coordinates.
At least 4 control points are needed for non-planar configurations, while only 3 are sufficient for planar ones. Given this formulation, coordinates of the control points in the camera coordinate system become the unknown of the problem. Solution is expressed as a weighted sum of the null eigenvectors of a matrix of size $2n \times 12$ or $2n \times 9$ named M.
Let the reference $3D$ points in the world coordinate system be X_i and the 4 control points used to express their world coordinates be c_j. Each reference point is expressed as a weighted sum of the control points:

$$X_i^w = \sum_{j=1}^{4} \alpha_{ij}^w c_j^w \qquad with \qquad \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{3.64}$$

with α_{ij} homogeneous barycentric coordinates.
The same relation holds in the camera coordinate system:

$$X_i^c = \sum_{j=1}^{4} \alpha_{ij}^w c_j^c \tag{3.65}$$

Control point are chosen with one as centroid of the reference point and the rest in such a way that they form a basis aligned with the principal direction of the data in order to increase stability. Introducing the camera internal calibration matrix K and $\mathbf{x}_i$ the $2D$ projections of the X_i reference points, have:

$$w_i \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix} = K X_i^c = K \sum_{j=1}^{4} \alpha_{ij}^w c_j^c \tag{3.66}$$

where w_i are scalar projective parameters. Expanding this expression by considering the specific $3D$ coordinates $[x_j^c, y_j^c, z_j^c]$ of each c_j^c control point, the $2D$ coordinates $[x_i, y_i]$ of the $\mathbf{x}_i$ projections, and the f_u , f_v focal length coefficients and u_c, v_c principal point that appear in the K matrix, equation becomes:

$$w_i \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \tag{3.67}$$

The unknown parameters of this linear system are the 12 control point coordinates and the n projective parameters w_i.
Substituting the last row of equation 3.67 inside expression of the

first two rows yields two linear equations for each reference point in which w_i disappear. By concatenating these equations for all the n reference point a linear system of the form $M\mathbf{x} = 0$ is obtained, with $\mathbf{x} = c_1^c, c_2^c, c_3^c, c_4^c$. Solution of the system belongs then to the null space of M.
Once the coordinates of the four control point are obtained, the R and t matrices that minimize the reprojection error of the world reference points, X_i^w , and their corresponding actual image points X_i^c , are then calculated. Computational cost of the method grows linearly with the number of correspondences $O(n)$.

According to [48], EPnP is consistently faster, more accurate and stable than the other non-iterative methods, especially for large amounts of noise, both for planar and non-planar configurations. Its accuracy is almost as good as the one of iterative algorithms, being much faster and without requiring an initial estimate. EPnP is also applied in many state of the art algorithms as [32].

3.4.3 *Algortihm Tests on Synthtetic Point Clouds*

A series of tests has been carried out on the different methods previously described for motion estimation in order to better asses the proper choice to be applied for the Navigation System. Tests have been made on synthetic set of clouds of $3D$ randomly generated points, with uniform distribution both on the plane viewed by the camera and in depth. Depth distribution of the points has been changed to different values in order to simulate both planar and non-planar scenes.
Considering the first camera as fixed ($P_0 = K[R|I]$), a default motion (rotation and translation) has been considered to define the second camera according to the model described in Chapter 2:

$$P_1 = K[R_{def}|t_{def}] \tag{3.68}$$

With R_{def}, t_{def} defined rotation and translation describing the motion. Simulated $2D$ points on the images plane have been simply obtained by applying the perspective camera model, multiplying $3D$ homogeneous coordinates by P_0 and P_1 to obtain the $2D$ homogeneous coordinates in the two different cameras. R_{def} and t_{def} have been changed in order to simulate different conditions (e.g. pure translations, small translations exc...), and algorithms have been tested with the obtained $2D$ projected points to reconstruct the camera motion and check if it was coherent up to scale with the imposed one and at which degree of accuracy. It has been chosen to try to simulate a condition similar to the one of the synthetic images of Lunar landing trajectories that will be later used as benchmark for the Navigation

Algorithm. For this reason intrinsic camera parameters have been set to the following:

$$K = \begin{bmatrix} f_u & 0 & u \\ 0 & f_v & v \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 886.51 & 0 & 512 \\ 0 & 886.51 & 512 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.69}$$

which correspond to a 60° aperture projective camera with 1024×1024 pixel screen and thus center at coordinates $(512, 512)$. The $3D$ point cloud has been set according to Table 3.3.

Number of Points	1000
Direction	**Interval**
x **distribution**	$-1000 \div 1000$
y **distribution**	$-1000 \div 1000$
Depth distribution	$1500 : 2000 \div 2000$

Table 3.3: *Configuration of the 3D planar point cloud with uniform random distribution.*

The depth of the points is generated with uniform distribution in an interval increasingly spanning between 1500 and 2000 units, that is points start from a planar configuration and end up with a non planar one. Errors have been measured at each step as:

$$E_{R\%} = \frac{||R_e - R_{ref}||}{||R_{ref}||} \cdot 100 \tag{3.70}$$

$$E_{t\%} = \frac{||t_e - t_{ref}||}{||t_{ref}||} \cdot 100 \tag{3.71}$$

with R_e and t_e being extracted rotation and translation. Three different cases have been considered: baseline equal to $3, 30$ and 100 units, which corresponds to a motion of about $1, 10$ and 50 pixels of the features in the image respectively. This is to simulate the possibility of having a very small baseline (high frame rate), a "standard" one and a quite high one(low frame rate, large motion between each image). All the algorithms used for the tests are the ones already implemented in the OpenCV-3.1.0 library inside the *calib3d* module. All these algorithms versions are already implemented within a RANSAC scheme for outliers rejection which has been set to work with a 99% level of confidence and a 1 pixel threshold for the maximum distance of a point from the model to be considered an outlier.
In the following, results have been expressed as percentage error in function of the increasing depth interval of the $3D$ points cloud, starting from the planar configuration to conclude with the most non-planar one. Red markers on the plots of the essential and fundamental matrices indicate solutions obtained with a low quality. The quality

measurement used is the determinant of the matrix: both E and F shall have null determinant, therefore motions obtained from a non-zero determinant of E or F matrices have been considered erroneously retrieved.

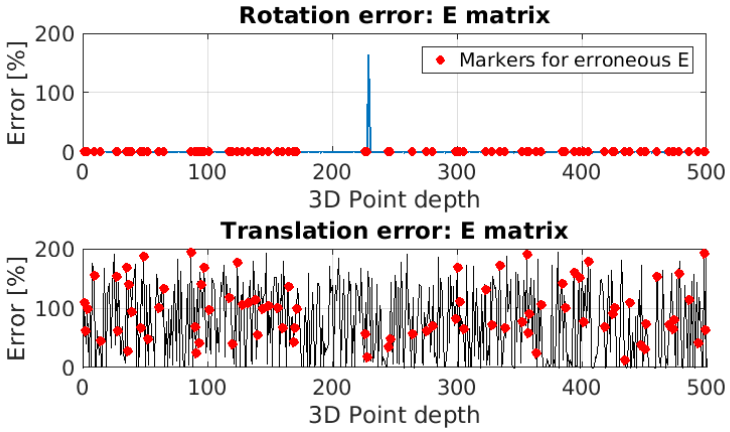
3.4.3.1 *Pure Translational Motion*

The results for the pure translation configuration are reported. This can be seen as a single step of a scenario similar to the one of a spacecraft landing, but with no motion along z axis (i.e. altitude reduction). Fig. 3.6 reports errors on reconstructed rotation and translation for each of the three different baselines using the 5-point method for essential matrix estimation. As can be observed markers are concentrated in all the results for the small baseline case (Fig. 3.6a), algorithm in fact does not work well for this kind of motion. This is in agreement with what described in [45], which states that the algorithm performs well for sideways motion but that the performance drops for small baselines between frames. Accuracy appears anyway to be lower then expected, in particular all the peaks present in both three configurations seems to underline a high sensibility to the change in distribution of the points. It has to be underlined however that the performance with long baseline (Fig. 3.6c) and sufficiently spread in depth $3D$ points is quite optimal. 5-point algorithm works optimally when there is a large baseline between the frames and when observed $3D$ point are non-planar.
Fig. 3.7 shows the results obtained with the 8-point algorithm for the estimation of the fundamental matrix. Method works quite well for the wide baseline (Fig. 3.7c), while with the small one (Fig. 3.7a) reports lots of erroneous retrieved motions, being almost all the points marked, indicating that solution comes from a wrong or imprecise fundamental matrix F. Results for the medium baseline set at 10 units (Fig. 3.7b) are still not accurate, being too much segmented. It shall be remembered by the way that F does not work for planar configurations, partially justifying the strange behaviour at the start of the plot in Fig. 3.7c, where points are still quite planar. One of the possible reasons of the so high oscillating behaviour for both E and F solutions in terms of motion for small baselines might reside in triangulation. As already explained in Section 3.4.1.3, triangulation of at least one point is needed to choose between the four possible motion solutions arising from the decomposition of E; if triangulation is carried out with such a small baseline and for distant features, uncertainty becomes higher and the method becomes error-prone.
For what concerns the homography H, whose plots are reported in Fig. 3.8, behaviours act as expected, with the errors increasing as the points become less and less coplanar. What can be observed by the way, is that H offers quite acceptable solutions even if the points are not perfectly coplanar, and that this solution degrades as the baseline

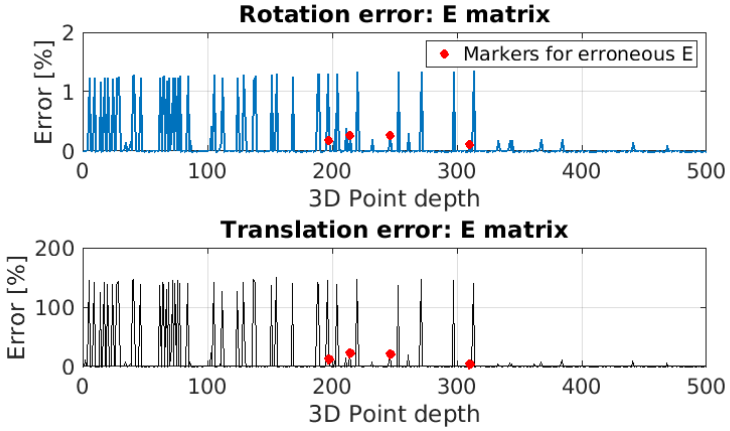
between the cameras is augmented. It is also interesting to see how rotations (null ones in this case) are always acceptably recovered even when translations are wrong.
Last considered method, ePnP is the only algorithm here presented which works with $3D$ to $2D$ correspondences. To make simulation more realistic, $3D$ points were obtained via triangulation of the $2D$ correspondences exploiting the linear method described in Section 3.4.2.1. Since the correspondences used are defined with an ideal projective geometry without any source of errors, using the optimal method for triangulation would be meaningless. Obtained results are represented in Fig. 3.9. EPnP algorithm works quite efficiently all-over the considered cases, for each depth distribution and baseline variations. Rotation in particular is always precisely recovered, while translation accuracy slightly diminish as the baseline becomes small.

3.4.3.2 *Rototranslational Motion*

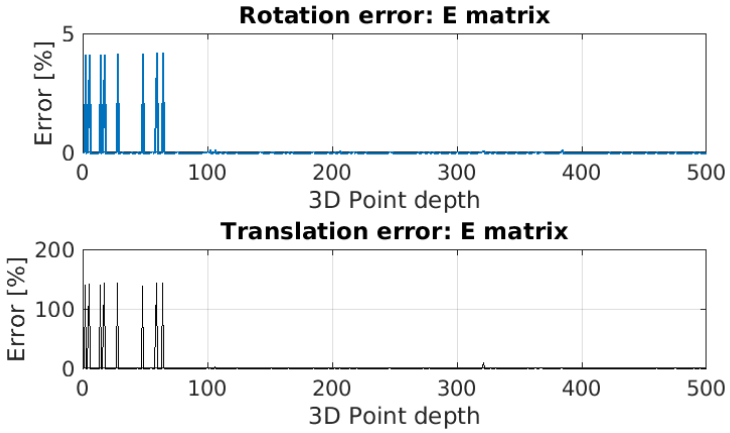
The same tests made for the pure translation case have been carried out for a motion of the camera with added rotation about two axes. The same cloud of points with depth distribution varying from planar to highly non-planar has been used. Rotations imposed from the first camera to the second one are two consecutive of 30° around two different axes.
Fig. 3.10 shows the obtained results for the essential matrix for motion with the added rotation. While for the small baseline case (Fig. 3.10a) solutions are still inaccurate, it can be noticed how for both medium and wide baselines (Fig. 3.10b and Fig. 3.10c), solutions start to become smoother and less affected by imprecisely calculated E matrices (i.e. less red markers). For quasi-coplanar configurations of the $3D$ point cloud, errors are still highly oscillating making estimation in that range not robust and very sensible to feature position.
For what concerns the fundamental matrix obtained via 8-point algorithm results are shown in Fig. 3.11. Solutions are no more heavily affected by wrong calculated F matrices. Results for the small baseline (Fig. 3.11a) are still erroneous, but for the 10 and 100 units translation case (Fig. 3.11b and Fig. 3.11c) solutions become almost exact, with high errors occurring only for almost planar $3D$ points configurations as expected. Results retrieved with the homography matrix are reported in Fig. 3.12. Solutions obtained have an error behaviour similar to the one obtained for pure translational motion, showing independently from the baseline a good accuracy as long as the cloud point is near a planar configuration. Considering in the end EPnP, whose results are given in Fig. 3.13, motion estimated is another time highly accurate, with only a slight decrease in precision on estimated rotations.



(a) $t = [-3, 0, 0]^T$

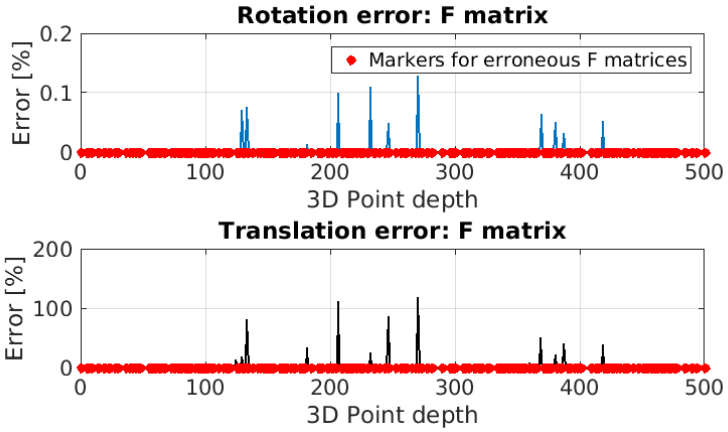


(b) $t = [-30, 0, 0]^T$

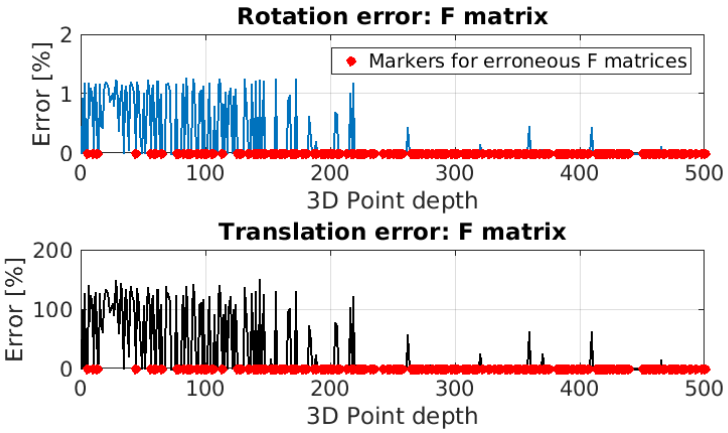


(c) $t = [-100, 0, 0]^T$

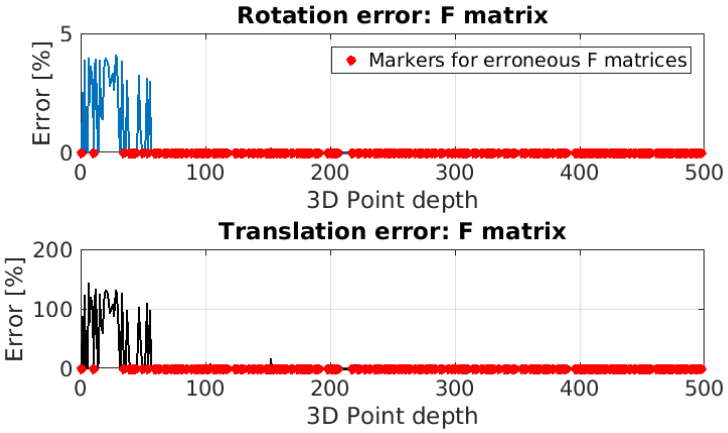
Figure 3.6: *Percentage error for the essential matrix E calculated via 5-point algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units.*



(a) $t = [-3, 0, 0]^T$

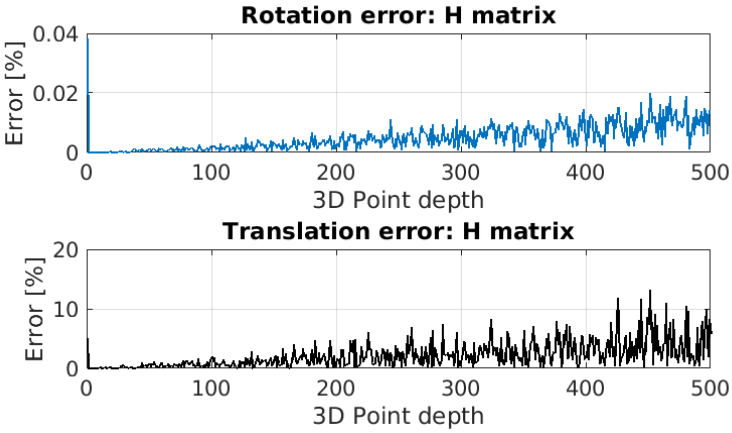


(b) $t = [-30, 0, 0]^T$

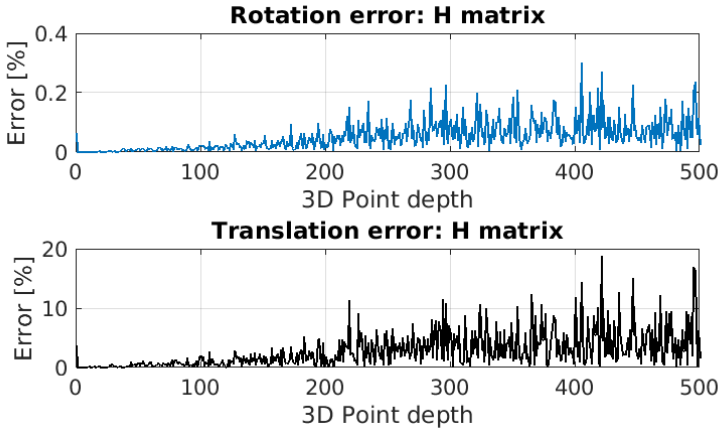


(c) $t = [-100, 0, 0]^T$

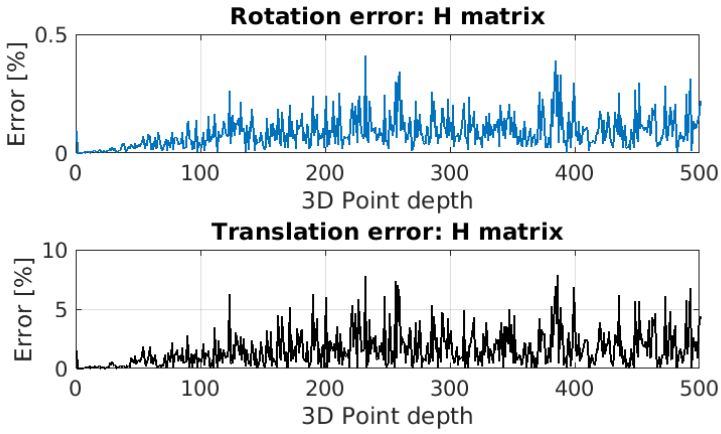
Figure 3.7: *Percentage error for the essential matrix E calculated via 5-point algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units.*



(a) $t = [-3, 0, 0]^T$

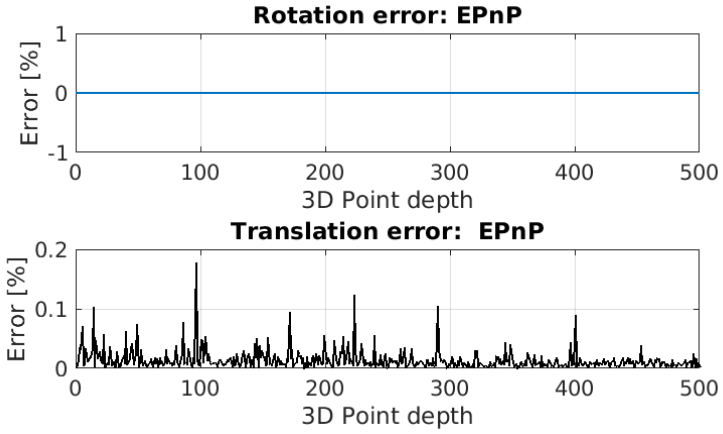


(b) $t = [-30, 0, 0]^T$

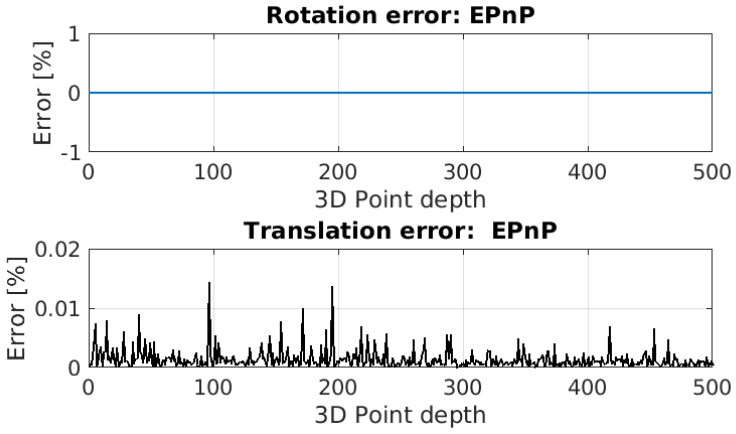


(c) $t = [-100, 0, 0]^T$

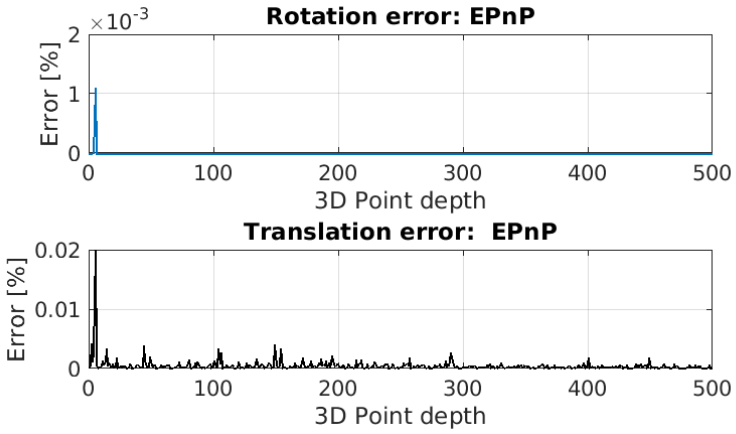
Figure 3.8: *Percentage error for the homography matrix H calculated with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units.*



(a) $t = [-3, 0, 0]^T$

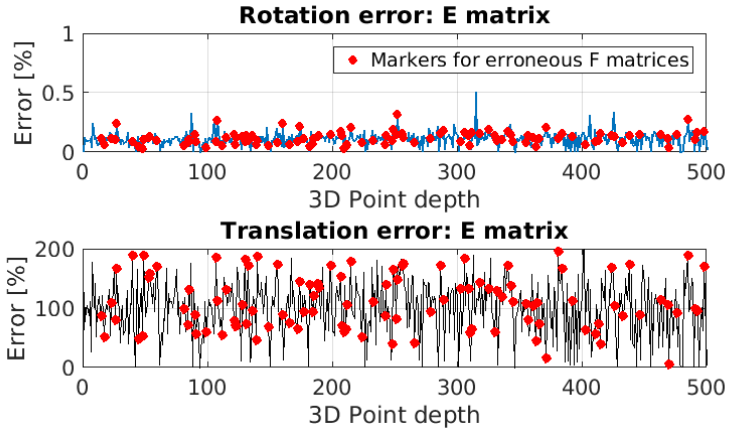


(b) $t = [-30, 0, 0]^T$

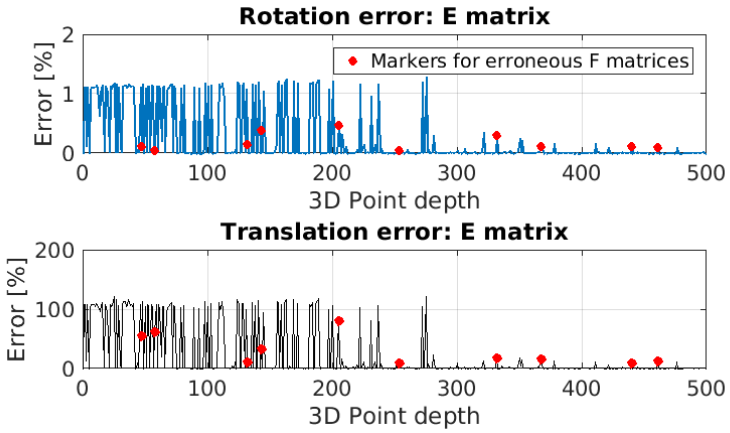


(c) $t = [-100, 0, 0]^T$

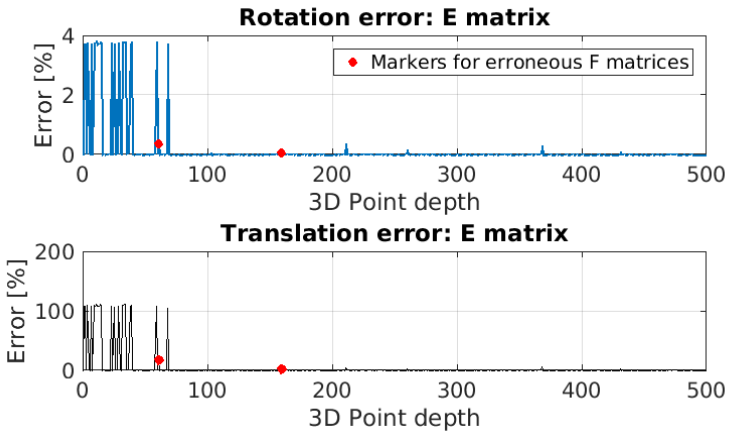
Figure 3.9: *Percentage error for the motion estimation with Epnp algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units.*



(a) $t = [-3, 0, 0]^T$

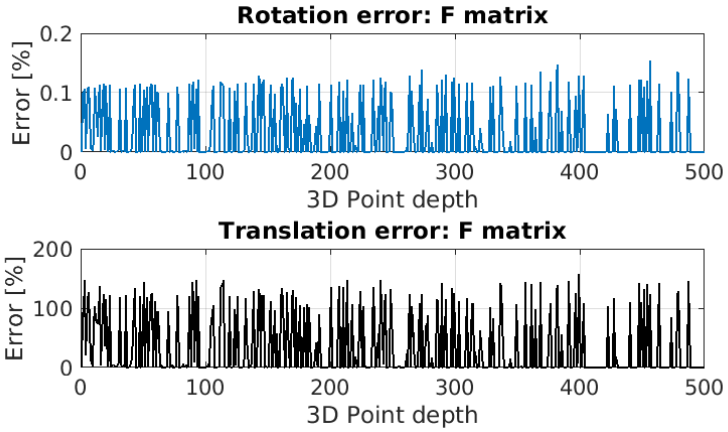


(b) $t = [-30, 0, 0]^T$

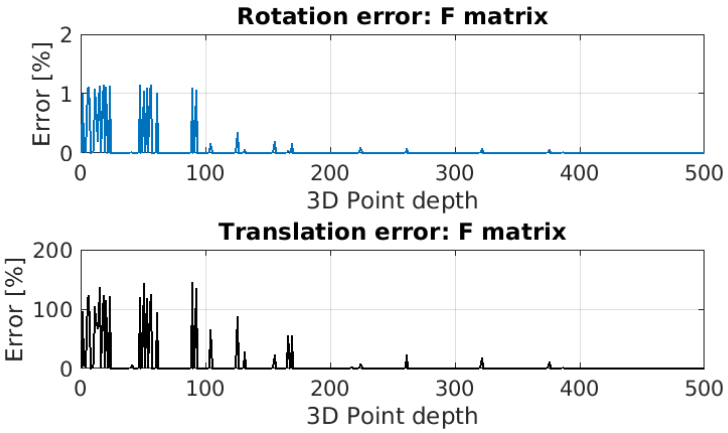


(c) $t = [-100, 0, 0]^T$

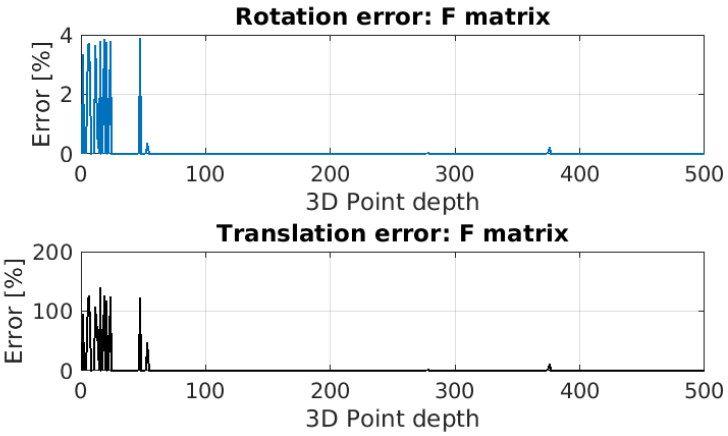
Figure 3.10: *Percentage error for the essential matrix E calculated via 5-point algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units and rotation of 30° about the x and y axes.*



(a) $t = [-3, 0, 0]^T$

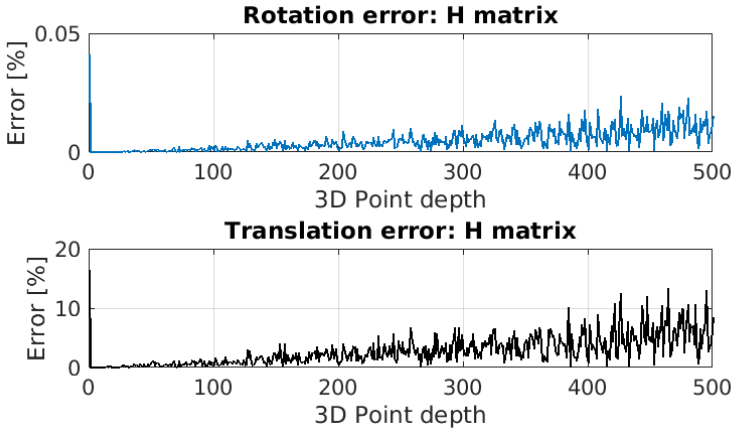


(b) $t = [-30, 0, 0]^T$

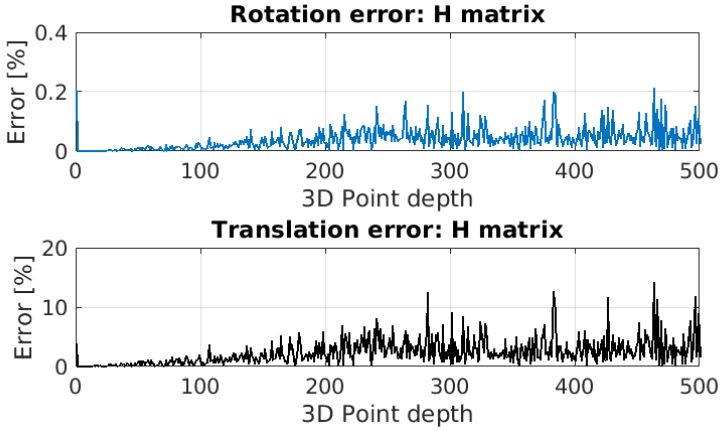


(c) $t = [-100, 0, 0]^T$

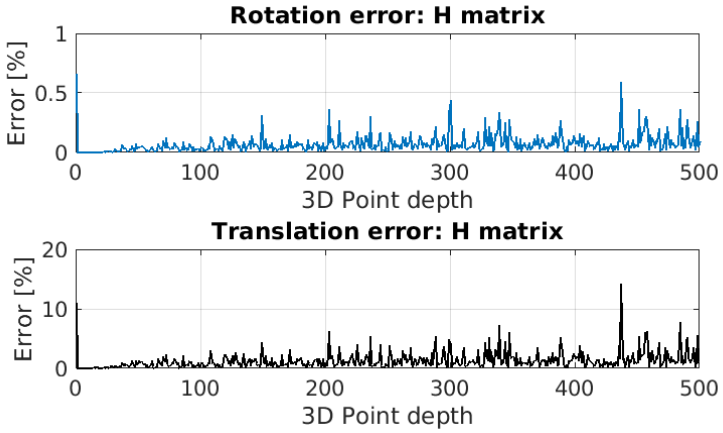
Figure 3.11: *Percentage error for the fundamental matrix F calculated via normalized 8-point algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units and rotation of* 30° *about the x and y axes.*



(a) $t = [-3, 0, 0]^T$

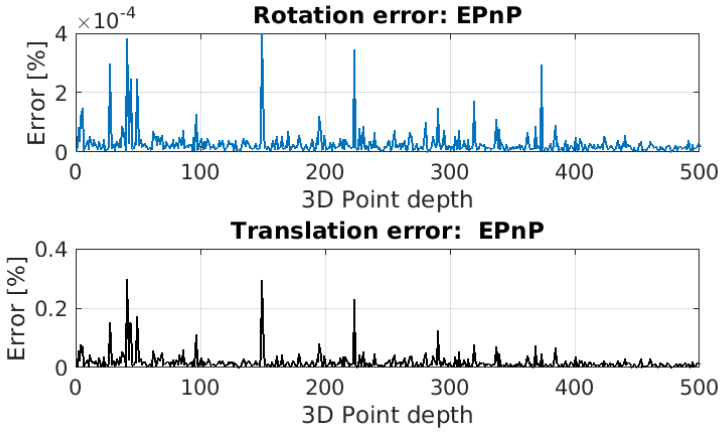


(b) $t = [-30, 0, 0]^T$

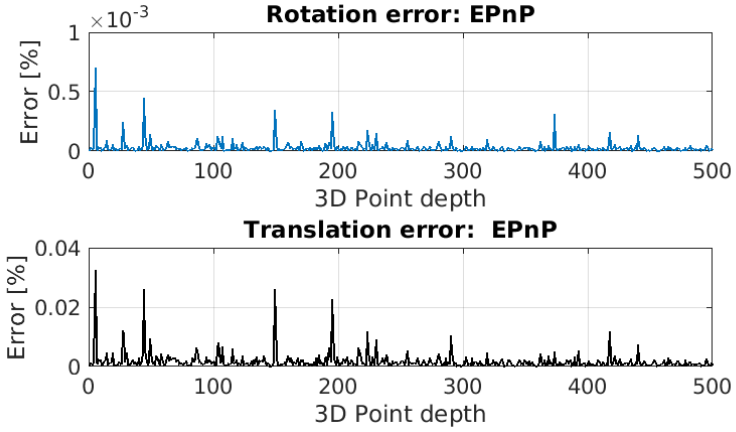


(c) $t = [-100, 0, 0]^T$

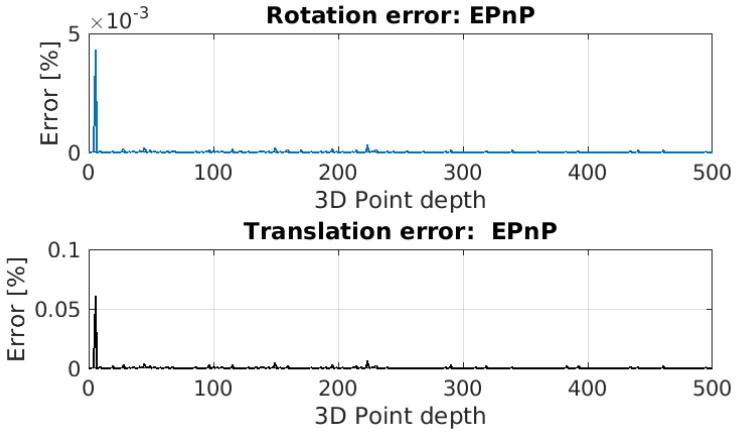
Figure 3.12: *Percentage error for the homography matrix H calculated with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units and rotation of 30° about x and y axes.*



(a) $t = [-3, 0, 0]^T$



(b) $t = [-30, 0, 0]^T$



(c) $t = [-100, 0, 0]^T$

Figure 3.13: *Percentage error for motion estimation with EPnP algorithm with baseline between the two cameras of (a)*3*, (b)*30 *and (c)*100 *units and rotation of 30° about x and y axes.*

3.4.4 *Algorithm Comparison*

According to the results presented and to the documentation available the five-point and the EPnP algorithms of [45] and [48] seems to be the most efficient. Homography for motion estimation of [46], while being quite efficient and precise even for case slightly non planar, implies a restrictive requirement, which is precisely that the features must lie on a plane.
Normalized 8-point algorithm described in [1] works also efficiently as proven with the tests done and by its high diffusion in state of the art algorithms, by the way it lacks for a solution when the scene is planar, and even when the scene is quasi-planar its accuracy drops quite heavily as proven. Since the scenario for a spacecraft landing as already explained may implies situations for which the scene seen by the camera will be almost planar, this is a notable drawback.
One may point out that according to the previous results five-point algorithm seems to work with a lower accuracy with respect to the Normalized 8-Point algorithm. Despite this is partially true, it will be shown that if the features given in input to five-point algorithm are of high quality and widely spread across the image then results obtained are better than ones coming from normalized 8-point. Moreover, five-point algorithm works efficiently almost independently of the scene configuration.

For what concerns EPnP, the choice is almost obvious in a context in which a sparse $3D$ map of the environment is available, and this is exactly the case for a Visual Odometry with mapping like implementation, which is the objective of this work. EPnP works with high accuracy in almost all the situations (small baselines, planar and non-planar scenes) and is also efficient in time computationally speaking. It should be anyway pointed out that EPnP here works so well because the problem is imposed with an ideal geometry fashion: $3D$ points are exact and so their projection on the image plane, with no outliers. This is a situation impossible to obtain in a real application, conditions will be slightly worse with some noise, and so an effective drop of performance is expected.

3.5 REFINEMENT METHODS

Motion estimation by itself, independently from the fact that is done exploiting $2D$ to $2D$ or $3D$ to $2D$ correspondences, does not give a sufficient accuracy in trajectory reconstruction. This is because it is an error prone step for different reasons. First of all, features extracted and tracked from the images are affected by noise. This basically means that the epipolar geometry described by the fundamental ma-

trix F or the essential matrix E is not exactly verified, namely: $x^{'}Ex \simeq 0$ or $x^{'}Fx \simeq 0$. As a consequence rotation matrix and translation vector extracted from these will be erroneous at a certain level. Moreover, these motion informations are concatenated together in order to reconstruct the whole trajectory, thus also concatenating together the errors, that consequently grows exponentially as the trajectory develops. All of this results in an increasing drift of the trajectory in time which is classical of the VO or V-SLAM probelms. In order to counteract this effect, limit the errors in reconstruction and improve accuracy to an acceptable level, several approaches have been developed across the years.

While the first techniques where born as off-line refinement methods for image post-processing in computer vision and were computationally quite heavy, development of new CPU processors along with a growth of the available computational power in the recent years have made possible to apply this techniques also to real-time applications as Visual Odometry and SLAM problems. Two methodologies have become predominant: filtering methods, that fuse the information from all the images with a probability distribution, and non-filtering methods, that retain some optimization to exploiting only some important keyframes [15]. For the first category the most diffused filtering approach is exploitation of the Extended Kalman Filter (EKF) along with a probabilistic description of the problem. For the second category instead, the most interesting technique is the Bundle Adjustment (BA). BA is an optimization method designed to minimize a function which represent the error in reprojecting 3D points whose 2D coordinates are known on the images taking as constraint the geometry of the scene. BA is nowdays the most efficient and diffused technique in the field of optical navigation for VO and V-SLAM applications, but has been developed and heavily improved only in recent years. Kalman filtering instead has been first implemented to solve the SLAM problem for mobile robots, and successively has become the first technique to be applied to solve optical V-SLAM.

In order to give a proper overview to the reader, here both techniques will be described, even if the one of interest for the Navigation Algorithm is Bundle Adjustment. An overview of Kalman filering, starting from the earliest robotic application to end up with optical navigation will be therefore given before analysing the BA problem, which will be described along with its different techniques and implementations. At the end, an overview of the different libraries available for implementation of Bundle Adjustment will be given.

3.5.1 *Filtering Approaches: EKF*

The Kalman filter was invented in the 1950's by Rudolph Emil Kalman, as a technique for filtering and prediction in linear systems. It is a method applicable only to systems represented with continuous states and uses moment representation with mean μ and covariance Σ to calculate momentary estimates of the state (beliefs). In order to obtain Gaussian estimations of the state given some measurements (posteriors), state and measurements probability must be linear with Gaussian noise, and initial state estimate must be normal distributed. These assumptions of linear state transitions and linear measurements with Gaussian noise are rarely met in practice. For this reason Extended Kalman Filter has been developed: EKF overcomes the linearity assumption making use of non-linear state and measurements probability functions. This way Extended Kalman Filter calculates an approximation of the true belief and represent this approximation with a Gaussian [49].
The EKF has become in recent years the most popular tool for state estimation in robotics. Its strength lies in its simplicity and in its computational efficiency. Efficiency which is related to the fact that it represents the belief by a multivariate Gaussian distribution, which can be thought of as a single guess, annotated with an uncertainty ellipse. In many practical problems, Gaussians are robust estimators. An important limitation of the EKF arises however from the fact that it approximates state transitions and measurements using linear Taylor expansions. This approximation may result too simplistic for some cases in which transition and measurements have complex behaviour [49].
The first classical application of EKF was to solve the SLAM problem in robotics for trajectory and map reconstruction, the so called EKF-SLAM [50]. The key concept behind the algorithm, which applies also for optical navigation, is to estimate the posterior over the momentary pose along with the map: $p(x_t, m|z_{1:t}, u_{1:t})$, where m represent the map of n features, u_t the motion model and p is the posterior. This only involves estimation of variables that persist at time t, and current poses are obtained integrating past poses only one at a time. Initial pose is taken to be to the origin of coordinate system, and incremental maximum likelihood is used to determine correspondences. To do this, the following assumptions must hold: map shall be composed by features, motion and measurements shall be described by Gaussians distributions and not seen features shall not be considered as informations [49]. $3D$ coordinates of all the features are estimated along the way and for this reason are included in the state vector. EKF algorithm observing a feature does not just improve the position of the feature itself, but also that of other features as well which are simultaneously seen by the camera. In the same way, seeing more

times a feature reduce its uncertainty in position which consequentially reduces the uncertainty of the estimated pose. Due to the often unrealistic assumption of Gaussian noise, spurious measurements may lead to false features placed in the map. To solve this problem, different approaches exists, one of which is to build a sort of pre-map with provisional keypoints which are not used for pose estimation and that are added to the true map only once observed many times with consequent reduced uncertainty.

Considering in detail EKF for the V-SLAM problem many issues arises. A typical application is the one described in [51] and[28], in which authors shown for the first time that it is possible to make real-time localization and mapping with a monocamera. Overall state of the system $\mathbf{x}$ is represented as a vector partitioned into the state x_v of the camera and the states y_i of entries in the map. State vector is accompanied by a single covariance matrix P which represents uncertainty in all the quantities inside it. During motion a predictive step uses a *motion model* to predict camera pose and how uncertainty has grown, when feature measurements are then obtained, a *measurement model* describes how uncertainty in pose and map can be reduced [51]. Despite this method has basically introduced for the first time a real-time solution for EKF-SLAM with monocameras, its application was restricted to small environments with a limited number of features. This is because the filtering approach made this way imply a computational cost which grows as $O(n^2)$ as the map grows, with n number of map points. Moreover, the process works on a single CPU thread and each time EKF has to filter the current step and all the previous ones. It results obvious how the computational burden increases in time during camera movements, being a non acceptable limiting factor for applications on real large environment with long open trajectories and thousands of features to process (as a landing trajectory for a satellite could be). It is important to consider also the fact that filtering needs to be initialized with a correct state, thing which is usually done manually pointing the camera to look at a priori known set of features. This approach for bootstrap is of course not feasible for a spacecraft application, in which Navigation Algorithm must be completely autonomous. Alternative approaches exploit EKF filtering in different ways; an interesting method being the one used by [31], already described in Section 1.3, which uses a separate thread dedicated to the map with respect to tracking and exploits Extended Kalman Filter to define the depth of triangulated feature coordinates.

Despite being some decades ago the breaking method for visual SLAM problem solution, and despite bringing some interesting solutions to optical navigation problems, EKF and other filtering approaches have many drawbacks which does not fit well with the

problem here considered, moreover they are nowdays outperformed by non-filtering approaches as Bundle Adjustment both in terms of accuracy and computational performances, as proven in [52]. EKF filtering shall not be anyway completely discarded: even if not directly applied to the navigation system to solve for pose of the camera in fact, it still remains a valid option for data fusion with other sensors of the spacecraft (e.g. an Inertial Measurement Unit).

3.5.2 *Bundle Adjustment*

"Bundle Adjustment is the problem of refining a visual reconstruction to produce jointly optimal $3D$ structure and viewing parameter (camera pose and/or calibration) estimates. Optimal means the parameter estimates are found by minimizing some cost function that quantifies the model fitting error, and jointly that the solution is simultaneously optimal with respect to both structure and camera variations" [53]. It can be seen as a large sparse geometric parameter estimation problem, with parameters being the 3D feature coordinates, camera poses and calibrations. Almost any predictive parametric model can be handled (any model that predicts the values of some known measurements on the basis of some continuous parametric representation of the world), along with different possible camera models. Problem may be posed as explained in [1]. Considering to have a set of $3D$ points X_j viewed by a sequence of camera with projection matrices P^i, denoting x_j^i coordinates of the j-th point seen by the i-th camera, supposing x_j^i and X_j known, problem is to find P_i such that:

$$x_j^i = P^i X_j \tag{3.72}$$

Eq. 3.72 will never be exactly satisfied due to presence of noise, consequently a Maximum Likelihood (ML) solution is searched assuming that the measurement noise is Gaussian: Objective is to estimate projection matrices $\hat{P}^i$ and $3D$ points $\hat{x}_i^j$ which project exactly to $\hat{x}_j^i$ as $\hat{x}_j^i = \hat{P}^i \hat{X}_j$ and also minimize the image distance between the reprojected point and detected (measured) image features x_j^i for every view in which the feature appears[1]:

$$\min_{\hat{P}^i \hat{X}_j} \sum_{ij} d(\hat{P}^i \hat{X}_j, x_j^i)^2 \tag{3.73}$$

where $d(x, y)$ is the geometric image distance between point x and y. This cost function (Eq. 3.73) is non-linear and may be optimized with non-linear least-square algorithms. Among the possible, Levenberg-Marquardt (LM) is the most diffused one, in particular for real time applications [31][41][54], due to its ability to converge promptly from a wide range of initial guesses using an effective damping strategy. For a complete description of the Levenberg Marquardt algorithm see

Appendix A.2.
Since each camera has 11 degrees of freedom and each 3D point 3 degrees of freedom, a reconstruction involving n points over m views requires minimization over $3n + 11m$ parameters. If the Levenberg–Marquardt algorithm is used (as is usually the case) to minimize Eq. 3.73, then matrices of dimension $(3n + 11m) \times (3n + 11m)$ must be factored (or sometimes inverted). Consequently, as the number m of views and n of features increase, this becomes extremely costly, and eventually impossible to do in real-time. The most demanding steps in particular, are the ones related to the computation of derivatives of the cost function with respect to camera and $3D$ point parameters [54]. To cope with this, core feature of any bundle adjuster is to take advantage of the sparsity which arises because parameters for scene features combine to predict measurements. Anyway, this is usually not sufficient for BA to be applied in real-time, and a widely diffused solution is to use only a subset of significant keyframes. A classical application of this is the so called *windowed Bundle Adjustment*, in which BA is applied to a subset of keyframes in a sliding window of fixed dimension which moves forward as new frames arrive in input. An alternative solution is the one first proposed in [30], and used also in [31][41], in which multiple core of the CPU are exploited to run the algorithm and BA is basically used on a different thread with respect to tracking thus releasing any real-time computational constraint. Bundle Adjustment for a real time application, is thus to be considered a tool for optimizing both the rotation and translations extracted from the motion estimation step and the 3D map. It is the last step of many navigation algorithms in robotics and does not only increase the accuracy of the camera trajectory, but also prevents error build up, with one iteration on just the most recent view resulting in a significant decrease of failure rate, as demonstrated in [54].
BA, being an optimization problem, can be completely customized, and different versions of it may be applied depending on the application context. Supposing to keep fixed the Levenberg-Marquardt as ML estimator, there are basically three possibilities of application of the same bundle adjustment, depending on which parameters need to be optimized:

- *Full BA*: All points along with camera poses are optimized, camera intrinsic may be included.
- *Motion BA*: Only the camera pose is optimized.
- *Structure BA*: Only $3D$ points are optimized.

These are anyway just given as guidelines, since as already state BA can be parametrized and adapted in any different way depending on

the application. For a classical state of the art application one can think of a full BA working for the first steps when map is initialized, which then becomes a motion only BA for tracking alternated to structure only BA each time the map is updated with new features.

3.5.3 *Application and Available Libraries*

It has been chosen to exploit Bundle Adjustment as the last step of the developed Navigation Algorithm, in order to improve both accuracy of the trajectory retrieved via motion estimation step and generated $3D$ map. This step is currently implemented but still not achieving satisfactory results, therefore implementation will not be proposed in this work. The choices made for the implementation are anyway here proposed.
BA is a complex topic, and development of a full optimizer takes too much effort, being also out of the scope of this thesis. For this reason, it has been decided to rely on an already built external software. Despite being BA an "hot" topic in computer vision in the last years, at the moment there are not that many developed dedicated libraries. Among the possible choices, three have been considered: g2o [55], SSBA [56] and SBA [57].

- **g2o** is a library developed for pose graph optimization. An exhaustive description of pose graph is given in [58]. Briefly, in robotics, pose graph is a way of formulating SLAM problem in which graph are used whose nodes represent poses of the robot at different point in times and whose edges represent constraints between the poses (measurements). Once such graph is constructed, BA is applied to find a configuration of the nodes which is maximally consistent with the measurements. g2o has been considered because efficiently applied in [32]; despite being quite attractive being the most efficient and most customizable method among the ones considered (it has been developed specifically for SLAM problems, while the other two not), it still remains quite complex to implement and requires too much effort in time to obtain a satisfactory and usable configuration. For this reason it has been discarded, but still remains the first choice for a future development of the optimization step.

- **SSBA**, the second library considered, is specifically developed for Structure from Motion (SFM) problems. SFM is an offline reconstruction method developed to build a consistent structure of an observed object from multiple camera observations. Despite SFM being similar in procedure to a VO or V-SLAM problem, it has the important main difference that is an offline method,

being free from any real-time computational constraint. Library implements a sparse Levenberg-Marquardt based non-linear least squares optimizer with bundle adjustment implementation. It has an interface which is quite simple with respect to g2o, and requires only a conversion of the data generated through OpenCV [59].

- **SBA**, the last library considered, was one of the first BA software package placed in public domain [57]. Still implementing a LM-based solver for Bundle Adjustment as SSBA, it has a predefined customizable structure with different drivers developed for "expert" and "simple" users. Despite being in some way similar to SSBA, the interesting thing about this software is that a wrapper library specifically developed to interface it with OpenCV (CVsba) has been developed by a group of Spanish researcher.

Discarding g2o, being both SSBA and SBA at the same level for a first step application of BA as an opinion of the author, it has been chosen to use the SBA library along with the OpenCV wrapper "CVsba" to be implemented on the navigation system.

4

OPTICAL NAVIGATION SYSTEM

This chapter describes the current development status of the Optical Navigation algorithm and all the tests performed on synthetic images simulating landing trajectories on the Moon. First, a characterization of a general landing trajectory for a spacecraft is described, along with the synthetic sequence of images used as benchmark for the tests. The general configuration of the algorithm is then presented. To conclude, different configurations of the Navigation algorithm are presented along with the results obtained on the simulated trajectories. Benefits and limits for each configuration are discussed.

4.1 SPACECRAFT LANDING TRAJECTORY

Main objective of this work is the development of a vision-based navigation system for planetary landing. Before going in detail with the system itself, it is therefore proper to characterize the operational scenario in which this will work, that is a typical landing trajectory for a spacecraft. Supposing the spacecraft starts the landing maneuver from a parking orbit around a target celestial body with no atmosphere, the following phases are identified:

- **Deorbit**: The spacecraft performs a braking maneuver to move from the parking orbit to an elliptic trajectory with pericentre located at $20 - 50$ km from the body surface.

- **Main Brake**: Once the spacecraft is at the pericentre of the elliptical orbit, engines are started and an intense brake is performed to inject spacecraft on an hyperbolic trajectory towards the selected landing site. This phase ends at the so called *High Gate*.

- **Approach**: This is the most delicate phase, in which an AGNC system would work at full regime. Spacecraft actively maneuvers according to the Guidance and Navigation subsystem, which estimates the engine impulses necessary to move to the landing spot selected by the HDA subsystem, knowing the relative position with respect to the surface. This Phase ends at the so called *Low Gate*.

- **Landing**: At low altitude from the landing site, spacecraft slows down to almost null vertical velocity and follows trajectory for a soft touchdown.

Optical Navigation starts from the beginning of the main brake maneuver, until the end of the approach phase (*Low Gate*). Indeed, theoretically ON can operate as long as the camera is pointing towards the target body and an image with distinguishable features is available. Fig. 4.1 shows a graphical illustration of the last phases of the described trajectory.

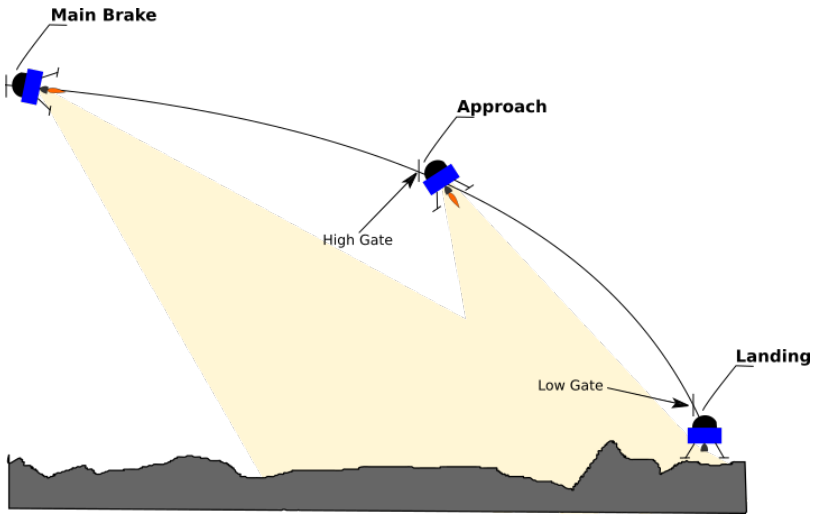


Figure 4.1: *Typical landing trajectory.*

4.2 MOON DATASET BENCHMARK

In order to validate and asses the performance of the Navigation System, tests on synthetic sequences of images simulating Lunar landing trajectories have been made. The synthetic images used have been obtained from high resolution Lunar DEM from the Lunar Reconnaissance Orbiter Camera (LROC) dataset [60]. These have an initial variable resolution between 2 and 5 m/px and have been used as starting point for generating the synthetic sequences. Small scale details have been then added including fractal noise, craters and boulders, increasing resolution up to 0.3m/px [61, 10]. Craters and boulders have been added respecting their real statistical distribution on Lunar surface and taking into account their real formation process [10]. In the end, the camera frame has been rendered in POV-ray [62] with realistic illumination conditions assuming a pinhole camera model with characteristics as from Table 4.1.

Two landing trajectories have been used for the tests, *Main Brake* and *Approach*. *Main Brake* is a constant thrust horizontal brake maneuver over the Moon Planck crater. Characteristics of the trajectory are given in Table 4.2, while Fig. 4.2 gives a graphical representation of the trajectory.

Camera model	Pinhole camera
Image resolution	1024×1024 px
angle of view	60°
Color	8-bit gray scale

Table 4.1: *Camera parameters used for the synthetic images*

Resolution	2 m/px
Horizontal development	35049×6507 m
Altitude range	$-154.3 \div 1380.6$ m
Duration	114 s

Table 4.2: *Approach trajectory characteristics.*

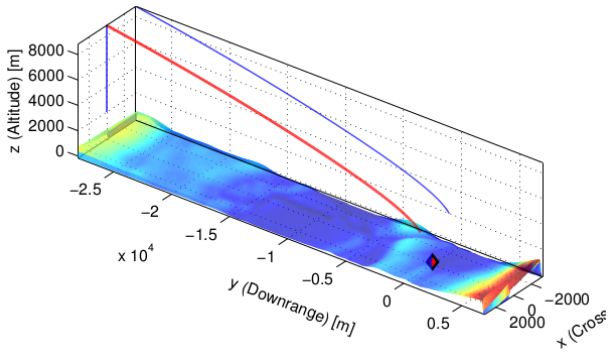


Figure 4.2: *Graphical representation of the Main Brake landing trajectory.*

Approach is a variable thrust approach phase maneuver representing the last part of the landing over the Moon Planck crater. Trajectory characteristics are given in Table 4.3, while representation of the trajectory is shown in Fig. 4.3.

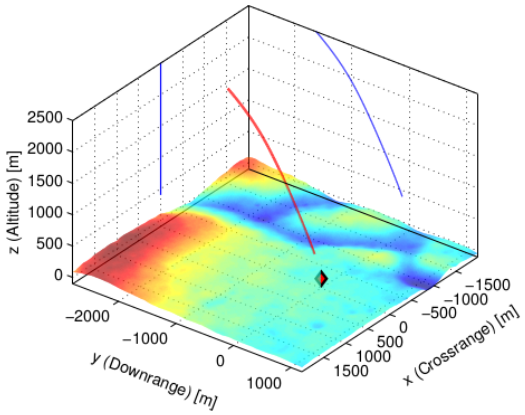


Figure 4.3: *Graphical representation of the Main Brake landing trajectory.*

Resolution	0.49 m/px
Horizontal development	4000×4000 m
Altitude range	$-115.7 \div 155.4$ m
Duration	55 s

Table 4.3: *Main Brake trajectory characteristics.*

For the *Approach* trajectory synthetic sequences of images exists for three different attitude modes of the spacecraft: camera fixed downwards, camera at 30° forward pointing and camera nozzle pointing. The same attitude modes are used for *Main Brake*, except for the nozzle pointing mode that is substituted by a sequence with variable attitude in which spacecraft performs a pitch maneuver from downward looking to 30° pointing.
Fig. 4.4 shows a short sequence of images from the approach trajectory with downward looking camera as illustrative example.

Figure 4.4: *Example of image sequence from the Approach trajectory.*

Trajectories can be characterized as follows:

- **Main Brake**: This trajectory has a large scale environment, with an almost pure translation as dominant motion. Since the scale is quite high, environment is seen as almost flat and with low details. Scale is almost constant for the whole motion, except during the final descent in which it starts changing quite fast.

- **Approach** This trajectory has a medium scale environment, with a steeper motion coupled to a little forward one. The change in scale from the beginning until the end is constant and quite high, with surface rich of details, but still observed as almost flat by the camera, especially for the first frames.

Considering then the different attitudes:

- **Downward pointing**: This represent a sort of standard reference configuration, features are seen only shifting on lines by the camera with a gradual zooming action. No rotations are present, being the attitude fixed.

- **30° Pointing**: Features are seen from this configuration with a different angle, resulting in a radial motion of them between consecutive frames which is expected to be easier to be interpreted by the navigation system. Attitude is fixed with no rotations.
- **Downward to 30° pointing**: The two previous configurations are here combined, with a pitch maneuver connecting the two attitude. This is a fast rotation around one axis almost decoupled by translations, a very challenging situation to be managed by a vision based algorithm.
- **Nozzle pointing**: This is the most challenging case, being pointing constantly changing, with a constant rotation about x axis occurring while trajectory develops. This makes rotations and translations difficult to distinguish, a challenging situation for an optical navigation algorithm.

Due to the different combination of kind of motion, scale variation and distribution of the salient features on the images, these sequences globally represent a quite challenging benchmark for the application of a Visual Odometry algorithm as the one proposed, being therefore an optimal optimal tool for validation.

4.2.1 *The KITTI Dataset*

The described scenario for the Lunar dataset is something not usually encountered in computer vision, the application field in which VO is born, where motion of the camera is usually almost planar and the observed environment results to have a small scale characterized by near objects and a major image depth. To better asses the behaviour of the developed navigation system under different imaging conditions and with a different scenario, it has been therefore decided to make tests also on a benchmark specifically developed for computer vision, that is the KITTI dataset [63]. This dataset comprises different trajectories obtained from a car equipped with a camera moving through a residential district and is a really challenging benchmark for monocular vision due to fast rotations, areas with lot of foliage, which make feature tracking/matching more difficult, and relatively high car speed.
The sequence here used is the first one from the dataset, named "00". Some frames from the sequence are shown in Fig. 4.5, while Fig.4.6 shows the nominal trajectory reconstructed with GPS data.

Benchmark characteristics may be resumed as follows:

- **KITTI sequence 00**: small scale feature rich scenario, with high depth images and a forward motion which makes features seen between frames moving radially across the screen. Lots of turns and changes in velocity are present.

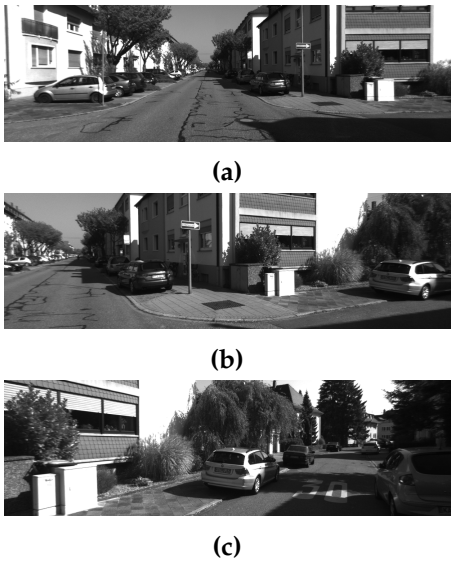


Figure 4.5: *Example frames extracted from the KITTI dataset benchmark, sequence "00", showing a right turn of the car.*

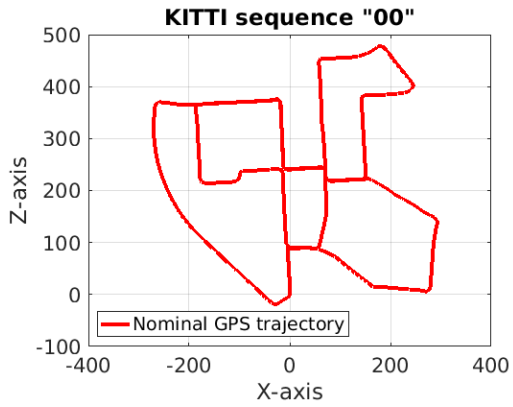


Figure 4.6: *Ground truth GPS trajectory for the KITTI dataset sequence "00".*

4.2.2 *Reference Frames*

A brief overview of the reference frames used across this chapter for trajectory reconstruction is hereafter expounded in order to make everything clearer to the reader.
Considering the monocular camera, notation used is the one exposed in Chapter 2. The reference system used for the *image plane* is the one adopted by OpenCV, centred on the top left corner of the image. Camera $3D$ frame has then been set accordingly as illustrated in Fig. 4.7.

Considering this camera coordinate system, the reference frame of the first camera, corresponding to the one of the first input image, has been set to be the *world coordinate frame* for trajectory reconstruction. Since each relative frame pose for adjacent cameras obtained from the motion estimation step is expressed with respect to the previous one

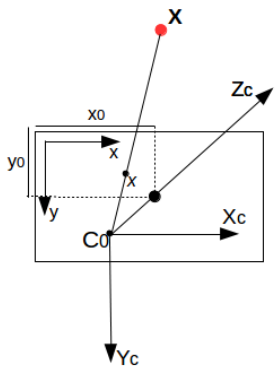


Figure 4.7: *Camera and image plane reference systems adopted. **X** is a generic 3D world point, **x** is its projection on the image plane. x_0 and y_0 are the principal point offset.*

(i.e. R and t obtained at step $k+1$ bring from frame at $k+1$ to frame at k), these are concatenated together in the following way to obtain absolute pose:

$$R_{k+1} = R_{k-1} \cdot R_k \tag{4.1}$$

$$t_{k+1} = t_k - R_{k+1} \cdot t_k \tag{4.2}$$

where $k-1$, k and $k+1$ are indexes for three consecutive frames. This means that after concatenation each new camera pose is expressed with respect to the first frame, that is the *world coordinate frame*. Fig. 4.8 gives a representation of this for a generic trajectory.

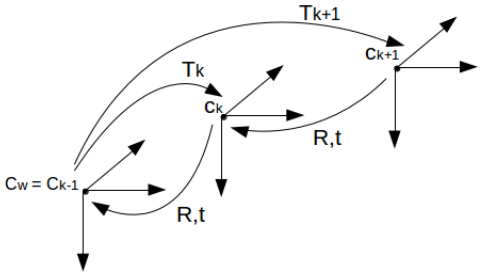


Figure 4.8: *Example of reference frame used for trajectory reconstruction, first frame is also set to be world reference frame. Relative pose R,t of adjacent cameras and absolute poses T are shown.*

Since no information or measurement from other instrument is given to the navigation algorithm, trajectory is reconstructed up to scale relative to the observed surface, which means no information of the absolute position of the spacecraft can be retrieved.

Lunar dataset benchmark has also to be considered, since ground truth trajectories are expressed in a different coordinate system then the one above described for the camera. Reference frame is in fact

centred on the landing point (which is not shown in figures since landing trajectory is not considered here) with the z axis pointing upwards and the x axis orthogonal to the trajectory development plane. z axis indicates therefore the *altitude*, x axis the *crossrange* while y axis indicates the *downrange* (which according to this notation will be always negative). This axes configuration can be easily observed in Fig. 4.3. For the sake of simplicity, this reference frame has been chosen to be the one used for trajectory comparison in the following sections for the different tested sequences.

4.3 TEST PLAN

Performances of the navigation system have been assessed considering the quality of the reconstructed trajectories with respect to the ground truths available from the Lunar dataset and KITTI benchmark. Attitude has been also considered for the Lunar dataset, expressed in terms of quaternions, and compared with the true one. For what concerns KITTI sequence in particular, only planar trajectory reconstruction has been considered with no calculation of the errors. This choice has been made since this sequence has been used only to have a qualitative observation of the behaviour of the algorithm under a different scenario.

Before any comparison it is important to consider that trajectory is reconstructed up to scale by the vision based navigation algorithm. Since no external measurement has been introduced to solve such an ambiguity, scaling has been introduced in post-processing in order to compare the scaled reconstructed trajectory with the ground truth.
For the Lunar dataset, reconstructed trajectory expressed in world camera frame has been first rotated to the dataset reference frame, then scaling has been applied. Relative scale has been retrieved as the ratio between the norm of the vectors representing the full length of the ground truth and reconstructed trajectory. Reconstructed trajectory has been then expressed the reference system described in Section 4.2.2 used by the dataset by applying a translation of the origin.
Trajectories comparison has been done considering separately each axis and their relative motion development in time, computing also for each one errors in terms of position difference with respect to the ground truth. Considering attitude, everything has been left expressed in camera reference frame for the sake of simplicity (most of the attitudes tested are fixed). Attitude behaviour in time has been expressed in terms of quaternions and have then been compared with the true one available from ground truth.
For what concerns KITTI dataset, since GPS ground truth is already available for each frame, scaling has been computed and directly ap-

plied during reconstruction. Trajectories are already expressed in the same reference frame and just the planar reconstruction is considered.

Tests have been structured according to Table. 4.4. F, E and H represent the different configurations of the navigation system tested, each one with parameters equally set and fixed. Every configuration has been run first on the *Main Brake* trajectory three different sequences: downward pointing, 30° pointing and downward to 30° pointing. The same has been then performed for the *Approach* trajectory with its three sequences: downward pointing, 30° pointing and nozzle pointing. To complete the tests, each configuration has been in the end run on the the KITTI sequence.

Trajectory	*Sequence*	*Configurations*		
		F	**E**	**H**
Main Brake	*Downward*	-	-	-
	30°	-	-	-
	Down to 30°	-	-	-
Approach	*Downward*	-	-	-
	30°	-	-	-
	Nozzle	-	-	-
KITTI	*"00"*	-	-	-

Table 4.4: *Navigation algorithm test plan.*

4.4 OPTICAL NAVIGATION SYSTEM CONFIGURATION

The general configuration of the vision-based navigation algorithm can be seen as a skeleton on which different building blocks may be interchanged. This configuration is represented in Fig. 4.9, where for each functional block of the Visual Odometry scheme the different considered tools are reported. Up to now, the only fixed blocks are the first one, *feature detection and description*, with ORB selected as the best approach; and the second one, *Lucas-Kanade feature tracking*. All the other blocks can be interchanged giving rise to different algorithm configurations, each one with its pros and cons and a different behaviour tested on the Lunar dataset. The triangulation and mapping blocks have been put apart since they represent a tool which can be used or not depending on the fact that Bundle Adjustment or EPnP are used. At the current development phase anyway, EPnP and BA are both implemented but still not functional, so will not be considered in the following.

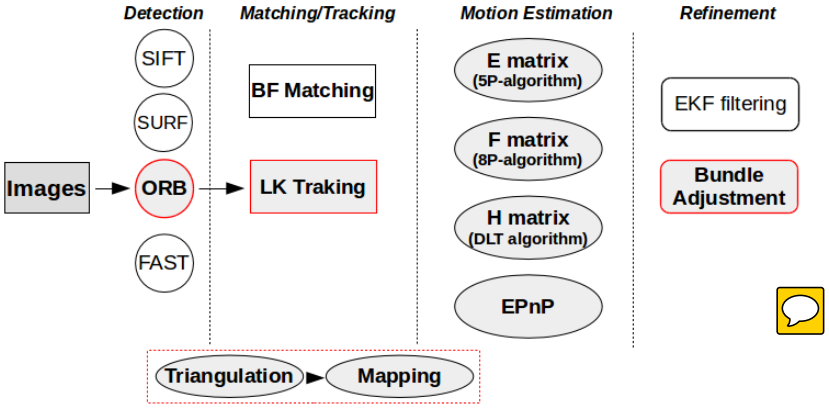


Figure 4.9: *Available tools for each building block of the VO algorithm, red highlighted blocks represent a fixed choice.*

4.4.1 *General Configuration*

Current developed configuration exploits Lucas-Kanade feature tracking of detected ORB features. The structure of the algorithm is independent from the exploitation of the H, F or E matrices for the motion estimation step. The map is not considered and the solution coming from the motion estimation step is directly used for trajectory reconstruction without use of any refinement technique. Functional scheme of the algorithm is reported in Fig. 4.10.

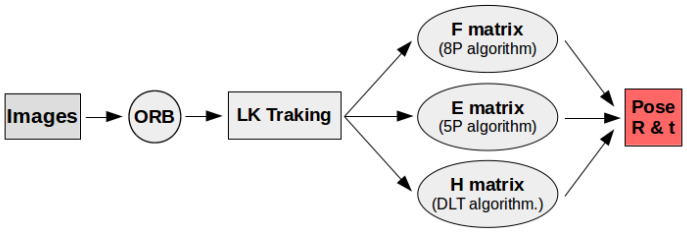


Figure 4.10: *Functional scheme of the first tested configuration.*

The algorithm works with the following steps:

- First, ORB features from the first image are extracted
- Second, image from the camera is taken in input. LK tracks features previously extracted on the new image. All the features gone outside the image or found with high uncertainty are eliminated.
- Essential/Fundamental/Homography matrix is retrieved within a RANSAC scheme. For the fundamental and essential matrix case, a check on the determinant of the matrix is done. If the quality of the matrix is too low (i.e. determinant is not zero), the

frame is discarded and not used for trajectory reconstruction. Successive frame is called and procedure repeated until an high quality matrix is obtained. In each case, features classified as outliers by RANSAC are deleted.

- Once a high quality F,E or H is available, motion estimation is done and R & t are retrieved. A check on the determinant of the rotation matrix is made. If check fails frame is rejected and procedure repeated from the previous step with the successive frame. Once coherent R and t are extracted algorithm can be considered bootstrapped and trajectory reconstruction begins, with R & t concatenated with the initial frame pose set as $[\mathrm{I} \mid \mathbf{0}]$.

- Features of the secondly processed frame become the current features which are given in input to LK for tracking with a new image coming from the camera. New features are tracked on the incoming image, if their number drops below a fixed threshold, then tracking is considered failed and an ORB detection is triggered on the previous image and LK tracking restarted.

- If enough features are tracked then correspondences with the previous frame are available and E, F or H are retrieved within the RANSAC scheme. For fundamental and essential matrix same check as for initialization is done and frame is skipped if matrix have a low quality.

- R & t are extracted from the estimated matrix. Again a check on the rotation matrix determinant is performed and frame eventually rejected.

- New R and t are concatenated together with the previous ones, all expressed with respect to the first fixed frame which is also the world reference one.

Procedure goes on until frames coming from the camera finishes. ORB detection is re-triggered every time the number of features drops below a predefined threshold, that is like the informations on trajectory are preserved but tracking restart from scratch each time. During the motion estimation step, independently from the method used, cheriality check is done to choose the right pose of the camera. To do this, feature correspondences are simply triangulated exploiting the linear method described in Chapter 3, and only the configuration with the highest number of points in front of both camera is retained.
Algorithm works extracting 300 features per frame at a frame rate of 0.5 Hz. This rate is lower with respect to the maximum available from the Lunar dataset (which is 30 fps), and has been chosen in order to guarantee a wider baseline between couple of frames, which results in a clearer motion for the camera point of view, and a lower computational burden for an ideal on-board hardware. Using a lower frame

rate also increment the available time to operate between frames, thus increasing real-time operations possibility. This configuration of the navigation algorithm is among the simplest possible to perform VO, but if correctly implemented gives remarkably good performance.

4.4.2 *Features Detection*

Features detection is the first step of the VO navigation chain. From the trajectory reconstruction point of view, this is the most sensible one which influences all of the subsequent blocks results. If a wrong or imprecise detection occurs, the whole algorithm may fail in reconstructing a coherent trajectory.
The feature detector employed is the already described Oriented FAST and Rotated Brief (ORB). The ORB algorithm used is the one implemented in the OpenCV-3.1.0 *features2d* library. This implementation given offers the possibility to extract the features from an input image and is completely tunable in all its parameters. Despite this, application of a detection algorithm directly on the whole image may be a too simplistic process to be used for navigation. There are several reasons in fact for which such a choice may lead to a lack of information and a wrong "description" of the observed scene.
The first thing to consider is that to achieve real-time performance and to be implemented on a spacecraft embedded CPU, the number of features extracted by ORB cannot be too high, otherwise the computational burden may grows too much. This leads to an upper limit threshold of about 300 features for current state of the art CPU according to [18]. As a consequence of this enforcement, an expectable effect arise: since only corners with the higher FAST score are retained by ORB, these will tend to concentrate on corners-rich zones of the image, leaving some zones not covered resulting in a non uniform distribution all over the image. This may be considered as a loss of information for the motion estimation step and in particular for the map build up (if a map is used). An example of this can be seen in Fig. 4.11 for an image of the *Approach* dataset: being the left part of the scenario occupied by a sort of large canyon with lots of details, all the features tend to concentrate there, leaving extensive portion of the image uncovered.
Because of this reason, it has been decided to split the image in sub-windows and to extract ORB features on each one, merging all of them together at the end of the detection step by simply adjusting their coordinates. Configuration with 4, 16 and 64 sub-windows have been tested. Increasing the number of sub-windows did not increased the performance. Exploiting sub-windows gives a better uniform distribution of the features on the image augmenting the possibility of correct motion estimation and eventually contributing to give a

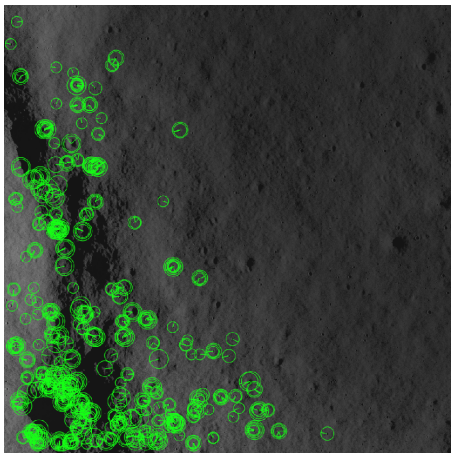


Figure 4.11: *ORB features detected from a frame of the Approach Lunar landing dataset. As can be observed, all features tend to concentrate in a single corner-rich zone of the image.*

uniform sparseness of the eventually reconstructed 3D map.

4.4.2.1 *Test on Sub-Windows*

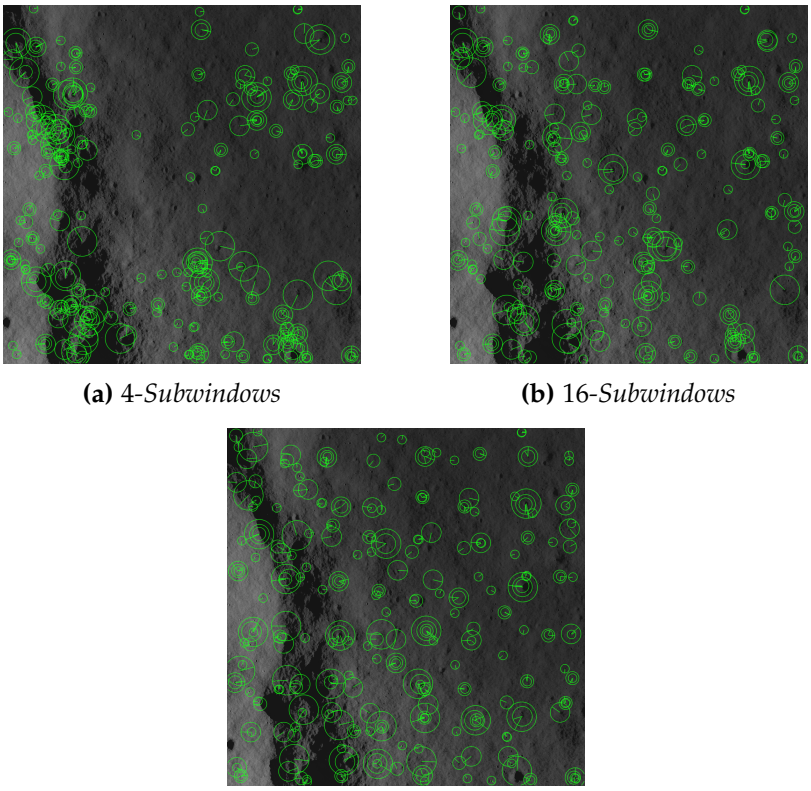
Tests to establish which configuration of the ORB feature detector gives the better performance have be done using 4, 16 and 64 sub-windows on the 1024×1024 images of the *Approach* dataset. ORB parameters for each of the sub-window configuration have been kept fixed and set according to Table4.5, where edge threshold is the minimum distance

Number of features	300
Scale factor	1.5
Pyramid layers	4
Edge threshold	0 px
Patch size	20 px
FAST threshold	20

Table 4.5: *ORB feature detector parameters setting.*

from the image border in pixels for which no features are searched, it is set to zero since search of the features is made on sub-windows and using such a threshold on each one would mean a loss of useful portion of the frame for detection. Patch size is the dimension of the patch used for rBRIEF descriptor. This parameters set is also the one used in all the tests done with the complete navigation algorithm on the Lunar dataset.

Speaking about computational time, the results obtained are shown in Table 4.6.



(a) 4-*Subwindows* **(b)** 16-*Subwindows*

(c) 64-*Subwindows*

Figure 4.12: *Image from the Approach Lunar dataset with ORB features detected.*

Configuration	mean Time [ms]
Complete image	20
4-Subwindows	20
16-Subwindows	21
64-Subwindows	26

Table 4.6: *Computational time required for each feature detection configuration.*

Time is almost equal for each configuration independently from the number of windows used: this was expected, since detection is done in more sub-windows but of smaller size and searching inside each one less features. The slight increase in time for the 64 sub-windows case is probably due to the chosen implementation, since a moving window has been used within a cycle, and an higher number of iterations are needed to cover the whole image.
For what concerns features distribution, Fig. 4.12 shows the results obtained for each of the configuration. It can be seen that for the 4 sub-windows case (Fig. 4.12a), features are not equally spread on the image and wide zones are still left empty. Better results are obtained using 16 and 64 sub-windows (Fig. 4.12b, Fig. 4.12c) with the whole image covered. In the end, the 64 sub-windows case has been selected to be

used for the Navigation Algorithm. Choice has been made because with respect to the 16 sub-windows case, even if the image is equally well covered, features using an higher number of sub-windows are more homogeneously distributed and more separated from each other. This configuration guarantees the highest accuracy for the subsequent estimation steps of the algorithm.

4.4.3 *LK Features Tracking*

Lucas-Kanade features tracking is the second step of the navigation chain. ORB features extracted from the first frame are searched on successive images in order to have at each step a set of correspondences with the previous one. This gives rise to the following scenario: given a set of ORB features for a first frame, for each successive frame, number of these features seen again in the image will be known along with their new position. Tracking is of course not infallible, and some features, even if present again in the new image, may not be tracked. Moreover due to the given motion, features will cross the image and soon or later will be out of the field of view. This means that after some frames the initial number of features will drop and a re-detection of ORB features will be necessary.
LK algorithm used is the sparse iterative one exploiting pyramidal approach already introduced in Chapter 3. The parameters setting used are listed in Table 4.7.

Window size	21×21 px
Pyramid levels	3
Eigen threshold	0.001
Max. iterations	30
Min. movement	0.01

Table 4.7: *Lucas-Kanade algorithm parameters setting.*

Window size is the dimension of the searching window in the new frame, eigen threshold is used to filter out low contrast features, while minimum movement represent the minimum displacement which the search window has to have between successive iterations to terminate the search. Tracked features comes out along with a vector of status giving informations of which features have been tracked and with which quality. This measure is immediately used to eliminate all points tracked with low precision or gone outside the image.

4.4.4 *Motion Estimation*

Three different methods have been considered for the motion estimation step: 8-Point algorithm for fundamental matrix estimation, 5-Point algorithm for essential matrix estimation, and DLT algorithm for homography matrix estimation. Each of these methods gives rise to a possible configuration of the navigation system which has been tested according to the test-plan defined in Section 4.3. Results obtained are hereafter reported and commented.

4.4.4.1 *Configuration with Fundamental Matrix*

This configuration exploits the fundamental matrix estimation for the motion step making use of the 8-Point algorithm. Once the fundamental matrix is retrieved, this has to be converted to an essential matrix exploiting Eq. 2.15 for motion estimation. Once E is available then it is possible to retrieve the relative motion between frames. As the name of the algorithm suggests, it requires at least 8 features tracked between each frame to properly work. Such a condition is anyway always verified since the tracking threshold is fixed to an higher level. This configuration of the algorithm has been proved to not work well on almost all the sequences of the dataset and even under different settings of the algorithm itself. For this reason, in the following only results for the reconstructed *Main Brake* and *Approach* trajectories for the downward pointing attitude sequences are presented. Motivations leading to the wrong behaviour are then discussed.

MAIN BRAKE TRAJECTORY

Fig. 4.13 shows the reconstructed trajectory for the Main Brake dataset with downward looking camera, while Fig. 4.14 shows the reconstructed attitude. Motion development for each axis as a function of time is reported along with the error with respect to nominal trajectory in Fig. 4.15.

Even if the reconstructed trajectory trends resembles qualitatively the actual path, trajectory immediately drifts after a very few frames (a small portion of the trajectory on the top left of the image is aligned) and becomes erratic. Attitude reflects the trajectory behaviour, with lots of oscillations about the nominal values.

This wrong behaviour is due to the exploitation of the fundamental matrix with the 8-Point algorithm for the motion estimation. As already stated and proven in Chapter 3, 8-Point algorithm does not work for planar or nearly planar configurations, and this is exactly the case. *Main Brake* trajectory in particular stresses this condition since Moon surface is seen from an high altitude and appears almost flat as seen from the camera. Also changing ORB settings and trying to

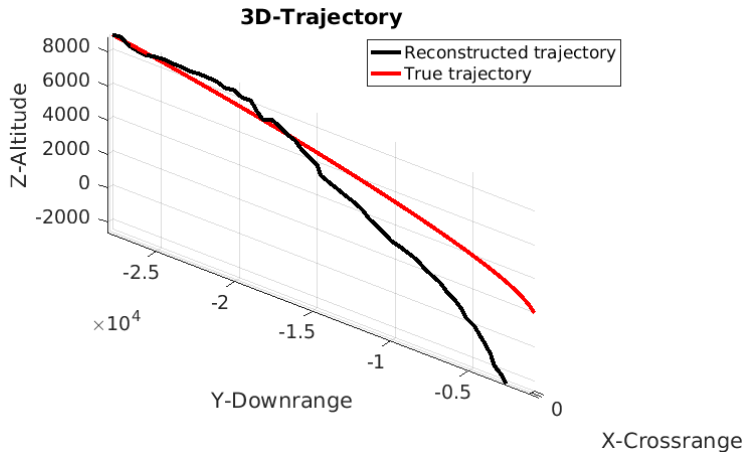


Figure 4.13: *Reconstructed Main Brake trajectory using F for motion estimation. It is clear how trajectory immediately drifts and is wrongly reconstructed. This is due to the 8-Point algorithm not working well for this kind of features distribution.*

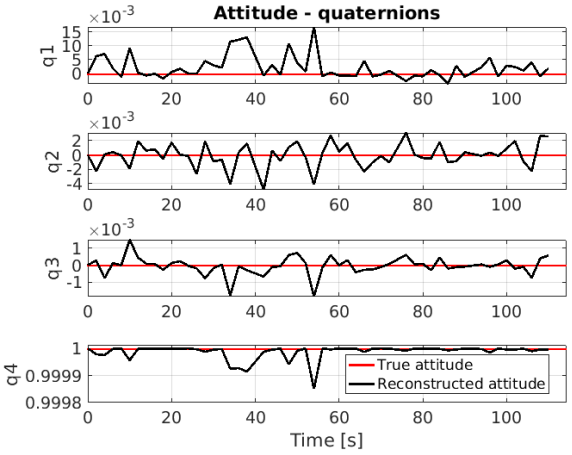
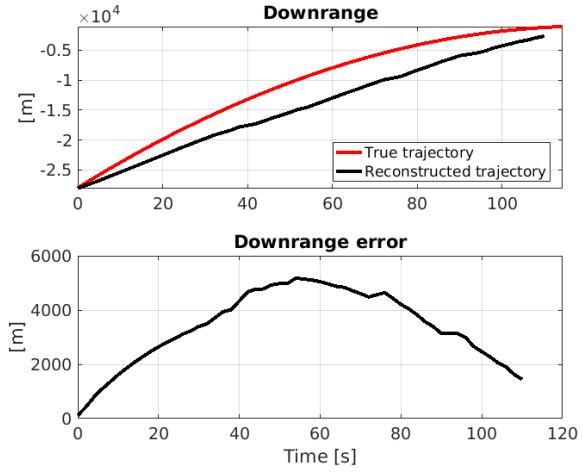
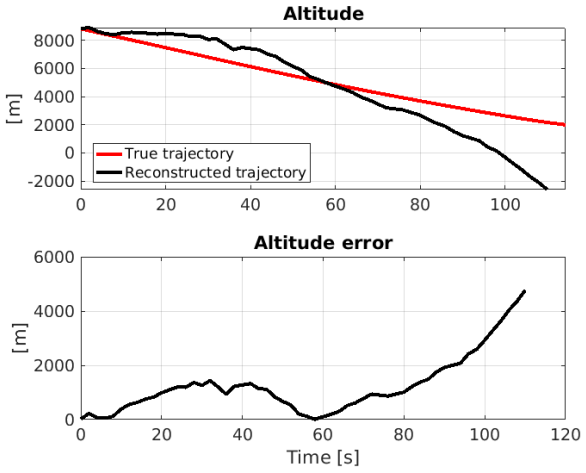


Figure 4.14: *Reconstructed Main Brake attitude using F for motion estimation.*

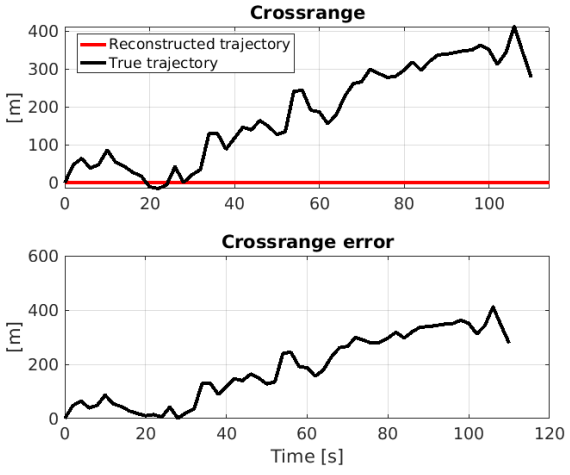
work on the frame rate and others parameters, trajectory reconstructed exploiting F always results not acceptable. Tests done with the other attitudes (camera 30° pointing and camera downward to 30° pointing) are affected by the same wrong behaviour and therefore not reported.



(a)



(b)



(c)

Figure 4.15: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

APPROACH TRAJECTORY

Reconstructed *Approach* landing trajectory with the downward looking attitude is reported in Fig.4.16. Accuracy is quite low, with an erratic behaviour which increasingly drifts from the ground truth.

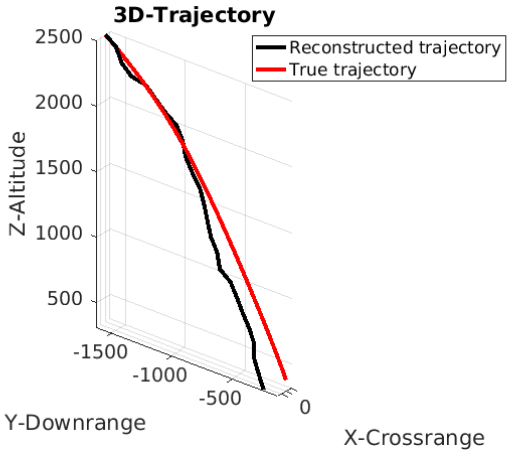


Figure 4.16: *Reconstructed Approach trajectory using F for motion estimation. Also in this case trajectory immediately drifts and is wrongly reconstructed.*

Reconstructed attitude behaviour can be observed in Fig. 4.17, with quaternions oscillating around the nominal values with good accuracy. *Approach* trajectory offers a condition different from the one given by *Main Brake*; with a higher level of details of the ground and a reduced scale of the problem. This should in theory be a more affordable scenario for the 8-point algorithm, and this is verified comparing results with the ones obtained for *Main Brake* trajectory.

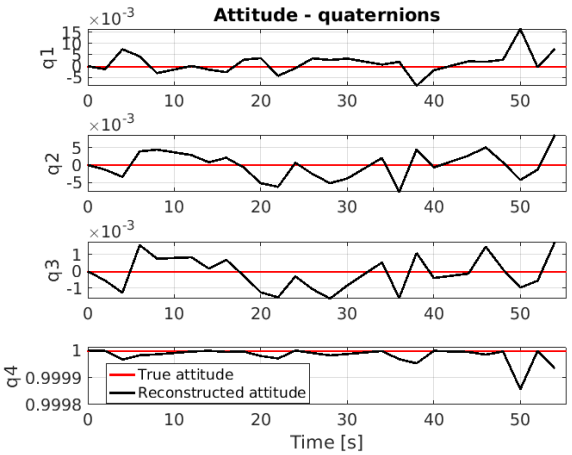
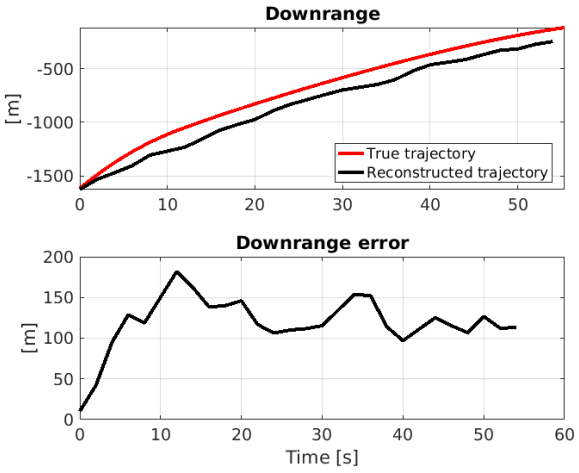


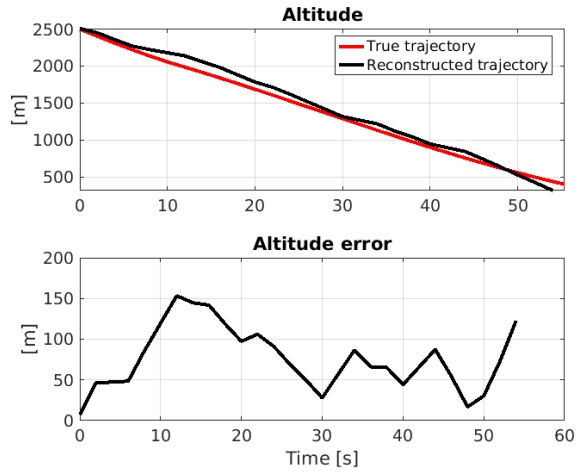
Figure 4.17: *Reconstructed Approach attitude using F for motion estimation. Since attitude is fixed quaternions do not change in time.*

Downrange, altitude and crossrange behaviour in function of time and their errors are reported in Fig. 4.18. Considering the smaller

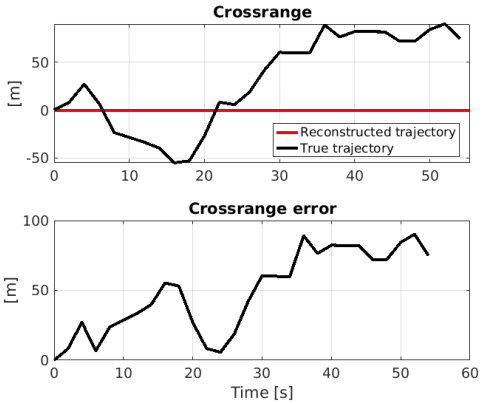
scale of the Approach environment the errors are quite high, with oscillations due to the erratic behaviour of the trajectory.



(a)



(b)



(c)

Figure 4.18: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

KITTI DATASET

KITTI dataset offers a completely different situation both in terms of motion and kind of images to test the algorithm. Since sequence is really long and drift becomes important after a certain point for a VO application with no refinement, not the whole sequence has been used but just the first part of it (about 1000 frames). Fig. 4.19 shows the result obtained. Errors have not been computed since this test is

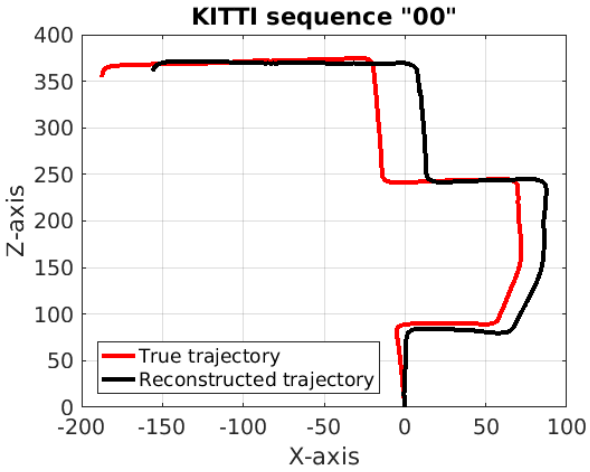


Figure 4.19: *Reconstructed Approach trajectory using F for motion estimation. Also in this case trajectory immediately drifts and is wrongly reconstructed.*

given just to have an immediate comparison with the previous results. While on the Lunar dataset the trajectory was completely rough with lots of wrong changes in directions resulting in a overall high drift, here the whole trajectory is reconstructed smoothly and with sufficient accuracy, being the accumulated drift and an erroneously evaluated rotation matrix at the beginning of the trajectory the only errors. This may seem surprisingly at first glance, but this situation is strictly connected to the kind of motion and to the feature distribution on the images, which in this case have a small scale being near the camera and completely surround it.
This test highlights the fact that the algorithm configuration is functional, but is not able to work properly for a planetary landing scenario as the one from the Lunar dataset.

4.4.4.2 *Configuration with Essential Matrix*

This configuration makes use of the "5-point" algorithm for essential matrix E estimation which is then directly used to retrieve motion. This method is computationally more efficient than the 8-Point one and works well both for planar and non-planar configurations. Algorithm has been tested on all the sequences for the *Main Brake* and *Approach* trajectories with good results and is currently the most robust

working configuration.

MAIN BRAKE

Results obtained with the navigation algorithm run on the *Main Brake* trajectory dataset are presented. The three different sequences with different attitude have been used, starting from the downward pointing case.

Downward pointing

For the downward looking camera sequence, Fig. 4.20 shows the obtained reconstructed trajectory. Fig. 4.21 shows the respective errors in downrange, crossrange and altitude, while Fig. 4.22 reports the reconstructed attitude. Trajectory is reconstructed with good accuracy,

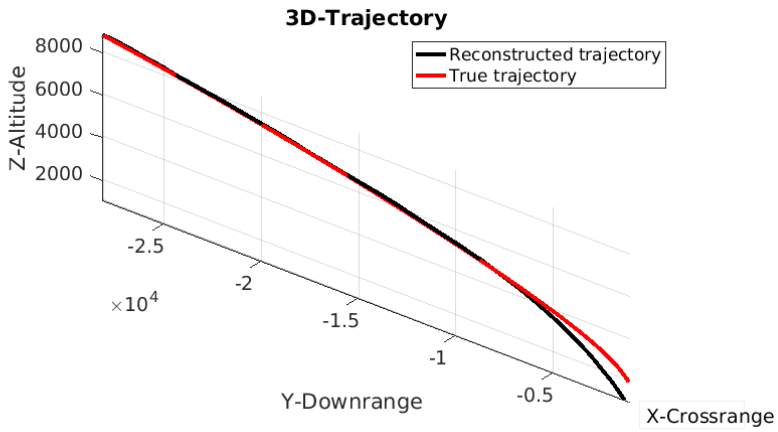
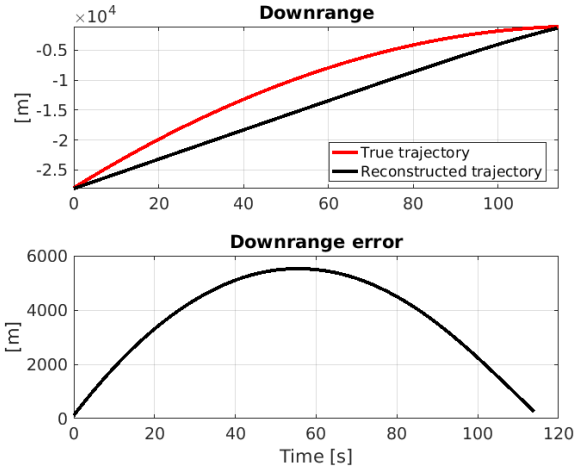
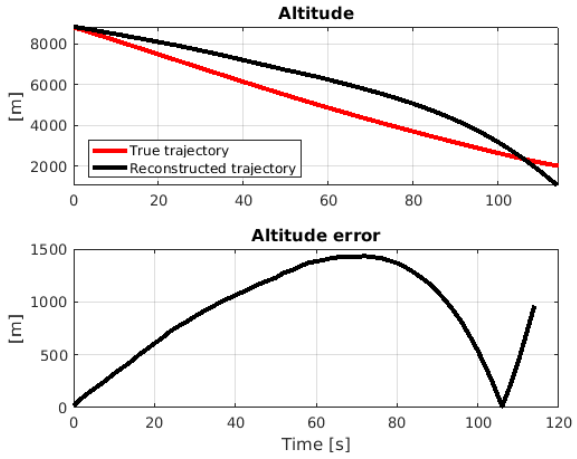


Figure 4.20: *Reconstructed Main Brake trajectory with downward looking attitude using E for motion estimation. Trajectory is estimated quite well, with major drift occurring in last portion.*

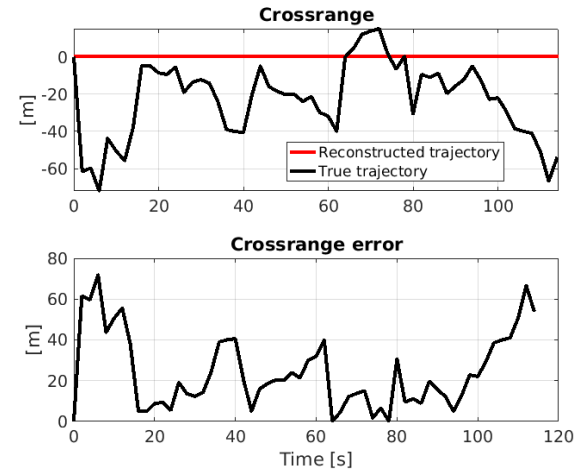
with visible drift occurring only in the last section, when the camera moves near the surface, that is when the sequence is already long and thus errors have accumulated, and there is a change in scale with respect to the initial and mid phases. Despite the smoothness of the reconstructed trajectory, looking at the errors (Fig. 4.21) it can be seen how a quite high drift occurs since the beginning both for altitude and downrange, while crossrange oscillates but is bounded in an acceptable region. Errors for cross and downrange are basically due to the different reconstructed curvature of the trajectory, because of the errors in rotation/translation extraction. The fact that the whole scale of the problem is really big should be anyway taken in consideration. These errors are of course too big for real navigation, but show that globally the trajectory as been reconstructed with good approximation, even without knowing scale and with the most simple configuration



(a)



(b)



(c)

Figure 4.21: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

possible of the VO algorithm. Concerning with attitude, Fig. 4.29 shows good results with an high accuracy, curves are almost smooth with low oscillations, meaning that the constant attitude is maintained.

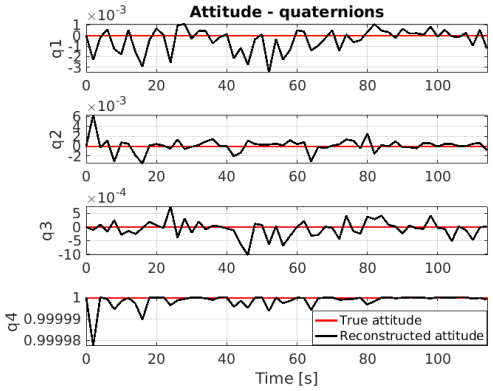


Figure 4.22: *Reconstructed Main Brake attitude for the downward pointing case using E for estimation. Since attitude is fixed quaternions do not change in time.*

30° Pointing attitude

30° pointing attitude is considered, with Fig. 4.23 which shows the obtained reconstructed trajectory. Fig. 4.24 shows the respective errors in downrange, crossrange and altitude, and Fig. 4.25 reports the reconstructed attitude. Being the camera inclined 30° downward

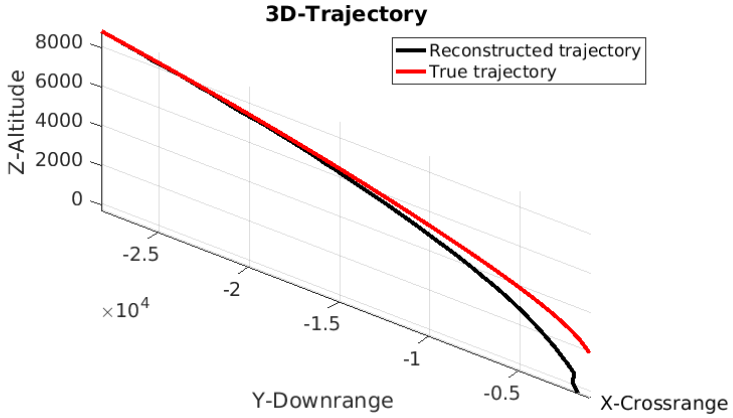


Figure 4.23: *Reconstructed Main Brake trajectory with 30° pointing attitude using E for motion estimation. A final step due to erroneous reconstruction is easily observable.*

with respect to the trajectory, this condition makes observation of the scenario more favourable from the trajectory reconstruction point of view, mainly because of the kind of motion of the features between

consecutive frames, which tend to move from the center to the borders of the image. 5-Point algorithm is expected to work well under this condition, but, as can be observed in Fig. 4.23, a final error in reconstruction occurs. This may probably due to a wrong cheriality check done during motion estimation. Overall drift is also a bit higher with respect to the downward looking case. Errors of Fig. 4.24 have a behaviour similar to the observed one for downward looking attitude sequence, with overall drift reaching the same magnitudes except for the last portion of trajectory when the erroneous reconstruction occurs. Crossrange also, shows a bit higher drift. For what concerns attitude (Fig. 4.25), accuracy remains high with only an erroneous peak in the final part of the reconstruction where the erroneous pose is retrieved.

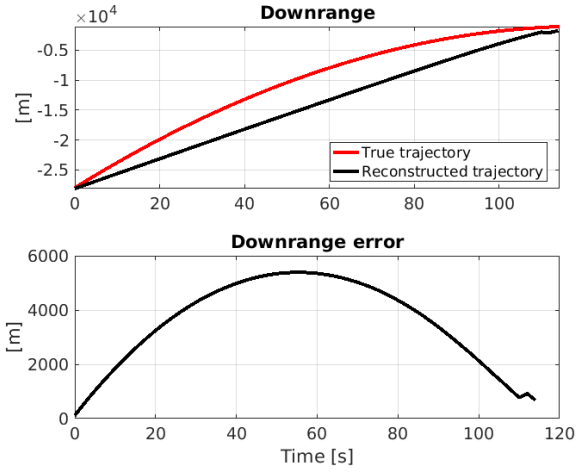
Pitch maneuver: downward to 30° pointing

Test on trajectory with an attitude change is quite interesting to prove how navigation algorithm reacts to such a challenging maneuver. Fig. 4.26 shows the obtained reconstructed trajectory. It is clear that the pitch maneuver done during the first phases, which is basically a fast rotation from down to 30° pointing around x axis, is misinterpreted as a turn, making the whole trajectory erroneously drifting from the actual one. It can be seen anyway that despite the initial erroneous rotation, trajectory is reconstructed with a coherent shape. This error in the reconstruction is mainly due to the fact that features are seen really far from the camera (large scale scenario) and that rotation is done quite fast independently from forward motion. Motion estimation step misunderstands the rotation around the x axis in part as a translation, and this is why trajectory is reconstructed with a turn. This situation is critical from the visual navigation point of view, and even more complex algorithm run with optimization may fail in recognize such a kind of motion.

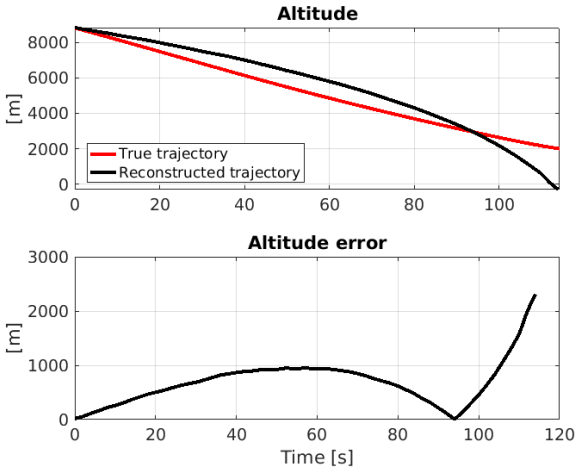
Fig. 4.27 shows the attitude behaviour, with the high initial curve corresponding to the wrongly interpreted pitch maneuver. Computed errors on each reference axis are here not given, being not representative.

APPROACH

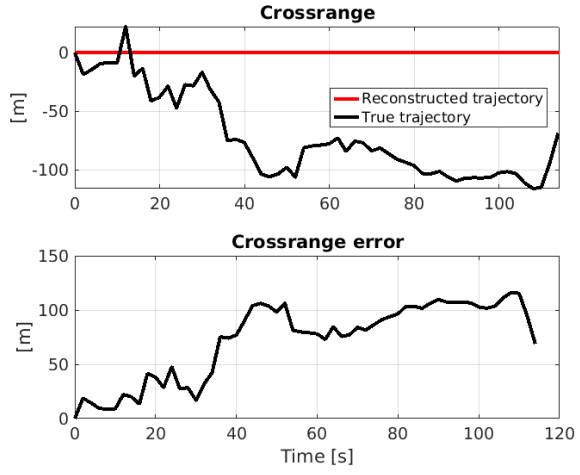
Considering the *Approach* trajectory, as before navigation algorithm has been run on the different sequences with different attitudes. First one considered is the downward looking, then the 30° pointing one and to conclude, the particular case of nozzle pointing camera. It has to be considered that this trajectory has a completely different scale, giving rise to reduced errors which not necessary means a lower drift.



(a)



(b)



(c)

Figure 4.24: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

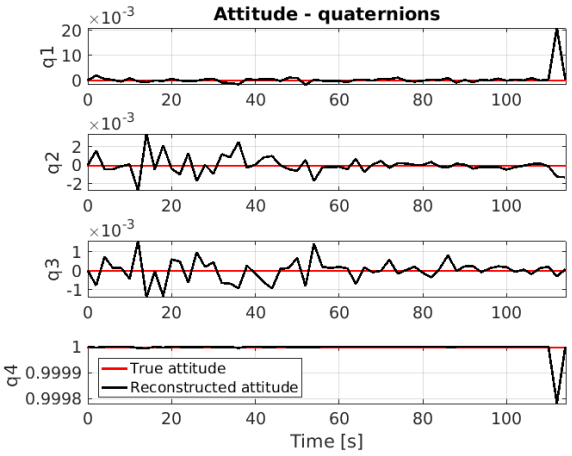


Figure 4.25: *Reconstructed Main Brake attitude for the* 30° *pointing case using E for estimation. Since attitude is fixed quaternions do not change in time.*

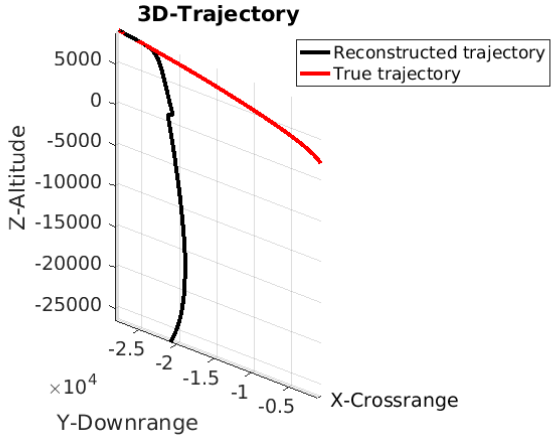


Figure 4.26: *Reconstructed Main Brake trajectory for the pitch maneuver using E for motion estimation. Three step due to erroneous altitude reconstruction are easily observable.*

Downward looking attitude

Fig. 4.28 shows the reconstructed trajectory with the downward point attitude. Accuracy is quite high with no wrongly estimated poses, which makes the trajectory smooth and only affected by an increasing drift which has its maximum at the end of trajectory.

For what concerns the attitude, Fig. 4.29 shows that it is retrieved with good accuracy, with small order oscillations which does not influence the overall accuracy of reconstruction.

Fig. 4.30 shows downrange, crossrange and altitude behaviour in time and reports their relative error with respect to ground truth. Crossrange drift is quite contained, while despite the apparent good accuracy of the reconstructed trajectory errors for both downrange

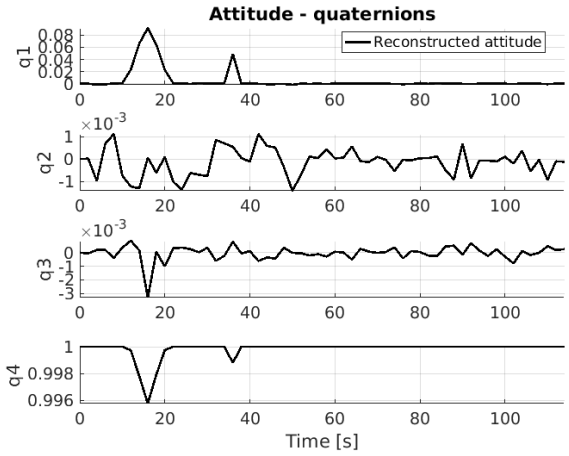


Figure 4.27: *Reconstructed Main Brake attitude for the pitch maneuver sequence using E for estimation.*

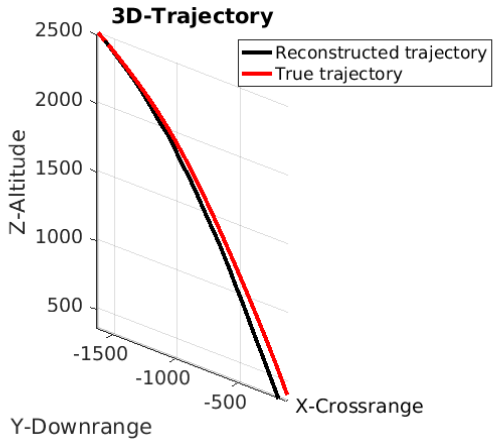


Figure 4.28: *Reconstructed Approach trajectory with downward looking attitude using E for motion estimation. Trajectory is correctly shaped with minor drift.*

and altitude have an increasing behaviour at the beginning which then stabilize in the middle to decrease in the final portion of trajectory. This trend may be interpreted once again as due to the different reconstructed curvature of the trajectory, which is due to imprecisions in the retrieved pose of the camera because of the ambiguity between rotations and translation which arises for this kind of scenario.

30° Pointing attitude

Reconstructed trajectory obtained for the camera pointing 30° downward sequence is shown in Fig. 4.31. The different attitude pointing gives an increase in accuracy with respect to the down pointing case, trajectory is in fact coherently reconstructed with no gross errors and a small drift compared to the previous. Downrange, crossrange and altitude behaviour are given in Fig. 4.32, showing a relative error

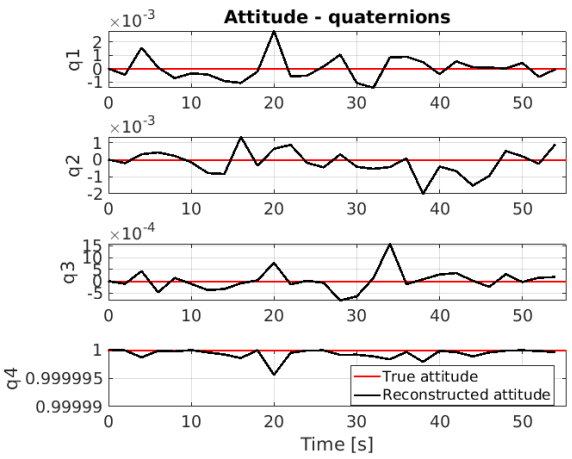


Figure 4.29: *Reconstructed Approach attitude for the downward pointing case using E for estimation.*

behaviour quite similar to the one obtained for the downward looking case, also in terms of maximum drift. Final error in altitude is however lower and also downrange error despite the first increase drops down to a lower level with respect to previous case. In practice, the first part of the trajectory is reconstructed similarly to the downward pointing case while accuracy is increased in the last portion of it, showing a lower drift.

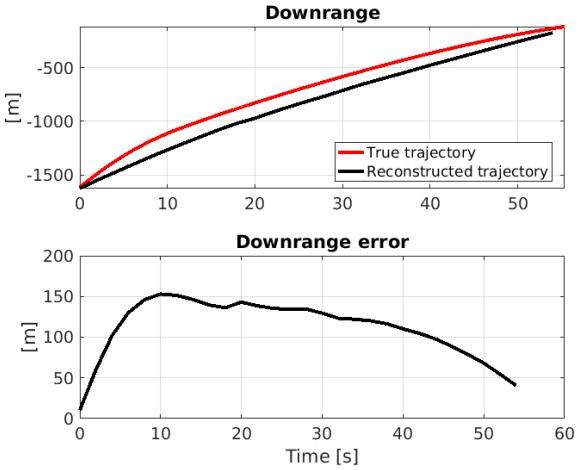
Attitude behaviour shown in Fig. 4.33 is accurate with only minor oscillations, showing that correctly retrieved rotation matrices have been obtained.

Nozzle pointing

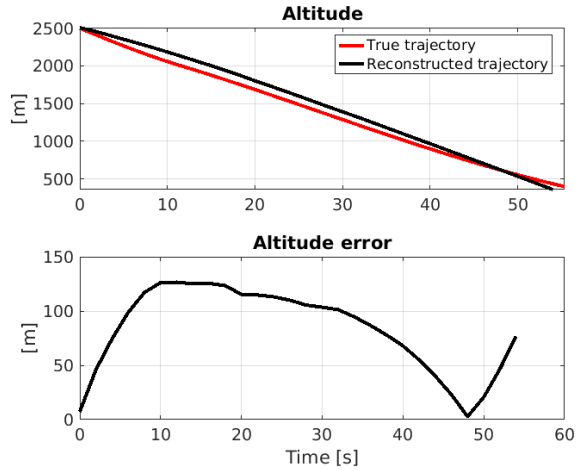
Reconstructed trajectory for the nozzle pointing attitude sequence is reported in Fig. 4.34.

Estimated motion is clearly wrong, but shows an interesting behaviour which has been already described for the Main Brake downward to 30° pointing attitude case, that is translations are misinterpreted as rotations and vice-versa, camera rotation to maintain attitude tangent to trajectory combined with forward motion, is interpreted as an altitude loss plus an opposite rotation making the whole trajectory being completely wrongly retrieved. This problem is probably intrinsic to the coupled geometry of the scenario and motion of the spacecraft, and cannot be solved even with refinement techniques. Even [31], state of the art algorithm described in Chapter 1, is not able to correctly reconstruct this kind of trajectory and gives similar errors.

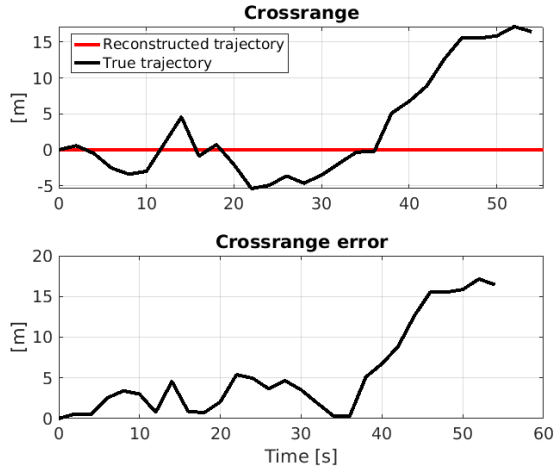
Fig. 4.35 shows the quaternion behaviour in time, underling the wrongly retrieved rotation matrices which makes the camera tracking failing.



(a)



(b)



(c)

Figure 4.30: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

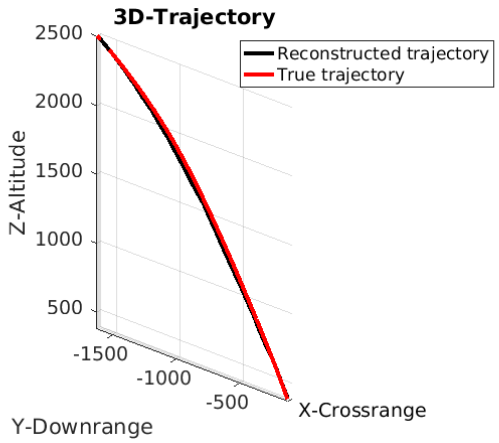
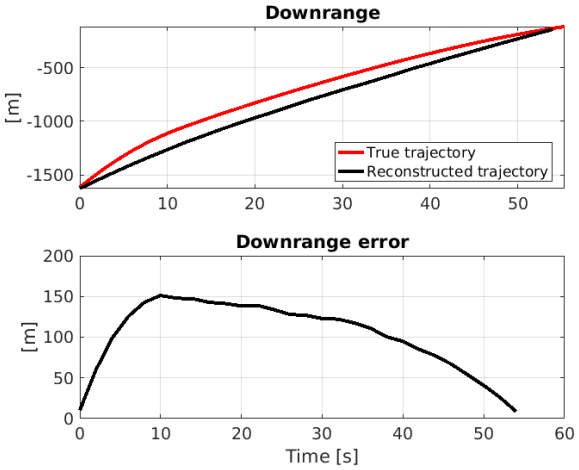


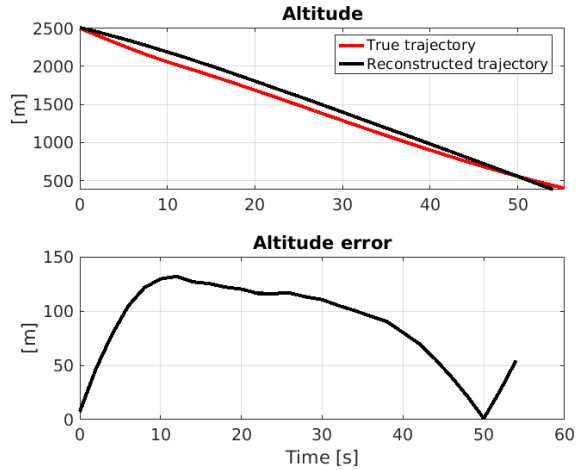
Figure 4.31: *Reconstructed Approach trajectory for the* 30° *downward pointing sequence using E for motion estimation.*

KITTI Dataset

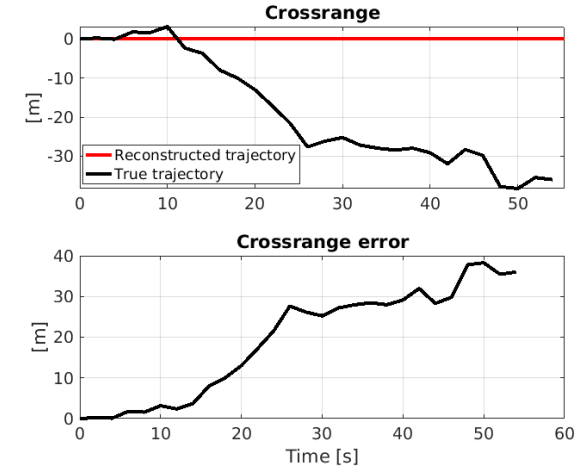
Results for the KITTI dataset sequence "00" obtained with the 5-Point algorithm for essential matrix estimation are given in Fig. 4.36. Only first 1000 frames have been used as for the tests on fundamental matrix configuration. As can be seen, trajectory is reconstructed until the end with surprisingly high accuracy, with only a minimal drift occurring for the last straight piece of road. This shows the remarkably high potential of the 5-Point algorithm if applied to certain kind of images. When the scale scenario is relatively small with features widely distributed and the environment is rich of element surrounding the camera not far away from it in particular, performance are maxima. This situation anyway, even if challenging for visual navigation, is quite far from the Lunar dataset scenario as already described.



(a)



(b)



(c)

Figure 4.32: *Downrange, Altitude and Crossrange behaviour in time of the reconstructed trajectory along with their relative errors.*

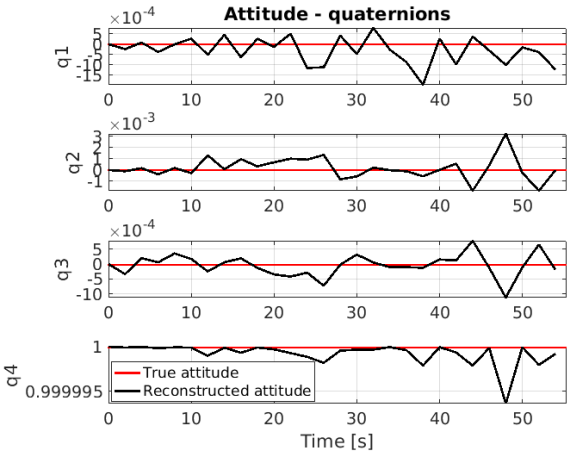


Figure 4.33: *Reconstructed Approach attitude for the downward pointing case using E for estimation.*

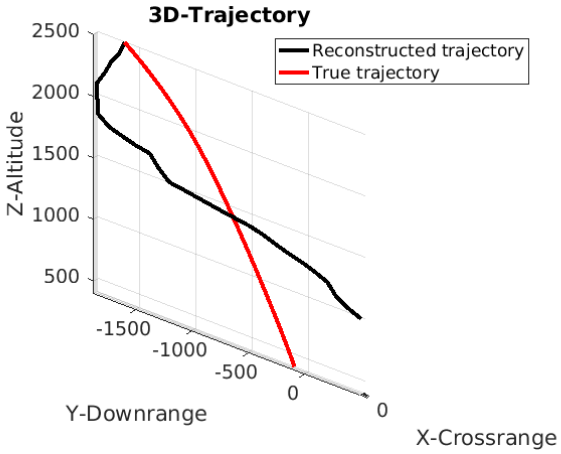


Figure 4.34: *Reconstructed Approach trajectory for the nozzle pointing attitude using E for motion estimation. Trajectory is completely wrong estimated, with rotations misinterpreted as translations and vice-versa.*

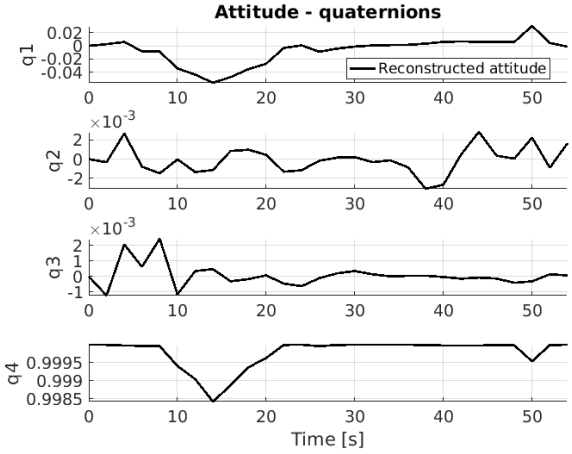


Figure 4.35: *Reconstructed Approach attitude for the downward pointing case using E for estimation.*

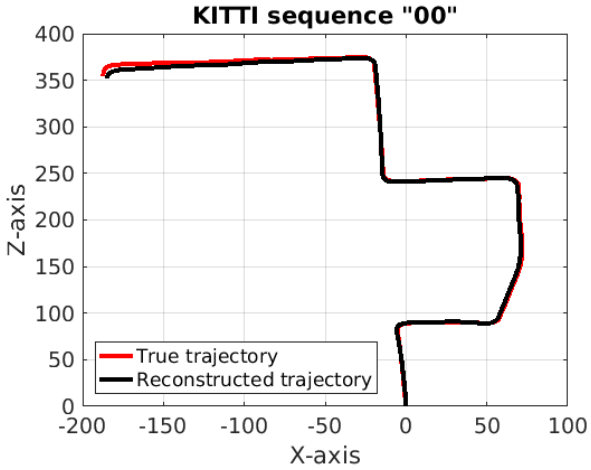


Figure 4.36: *Reconstructed partial KITTI trajectory (first* 1000 *frames) for sequence* 00 *using E for motion estimation.*

4.4.4.3 *Configuration with Homography Matrix*

Homography to relate two scenes can be used if those are planar or at least partially planar. This can be considered as first approximation the case for both *Main Brake* and *Approach* trajectories due to their properties, as already explained. Homography matrix and motion are retrieved with the algorithms presented in Chapter 3. As for the fundamental and essential matrix configurations, algorithm has been run on all the sequences of the Lunar dataset. KITTI dataset instead, here is not considered, since there is a priori known no possibility that an homography matrix relating two images of the type present in this sequence can exist.

MAIN BRAKE

Main Brake trajectory is the first used for tests and the one which presents the conditions closer to the optimal ones for the homography application. As previously done, tests have been done for the three different sequences available starting from the standard downward pointing case, proceeding then with the 30° attitude and concluding with the pitch maneuver one.

Downward pointing Trajectory obtained with the algorithm run on the downward pointing sequence is reported in Fig. 4.37, which shows a behaviour quite smooth until the last portion of the trajectory. The first error is an inconsistent vertical descent, then the last portion of trajectory, which is steepest with respect to the previous, is completely wrongly estimated as a vertical descent. Both errors occurs when the camera moves closely to the surface and the possibility of not verifying anymore planarity condition increases. For the second error in particular, this occurs when trajectory starts becoming steeper, and the algorithm misinterprets this sudden change in altitude as a vertical descent. Considering the attitude in Fig. 4.38, quaternions shows a highly coherent behaviour making exception for the last portion of trajectory, where peaks corresponding to the wrongly retrieved rotations are clearly visible in $q1$ and $q4$.
Fig. 4.39 shows the downrange, crossrange and altitude behaviour along with their errors. In each of the plots accuracy is quite good but behaviour concludes with a large discrepancy in correspondence to the erroneous reconstructed last piece of trajectory.

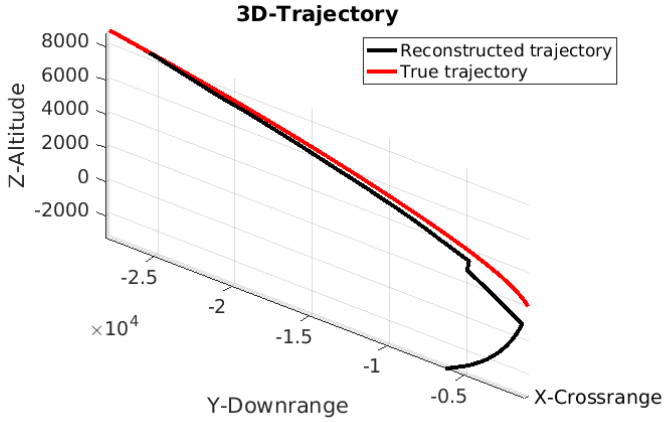


Figure 4.37: *Reconstructed Main Brake trajectory with downward looking attitude using H for motion estimation. A clearly erroneous retrieved rotation can be seen in the last portion of trajectory.*

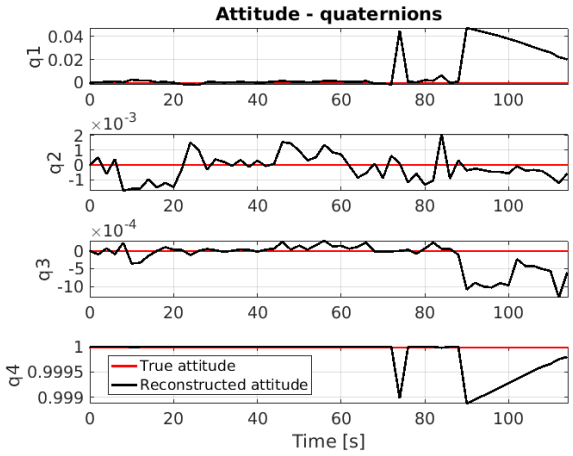


Figure 4.38: *Reconstructed Main Brake attitude for the downward pointing case using H for estimation. The wrong retrieved rotations are clearly visible in q1 and q4.*

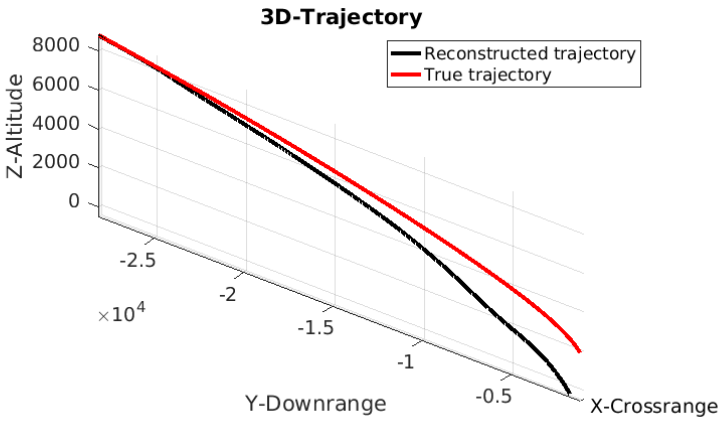
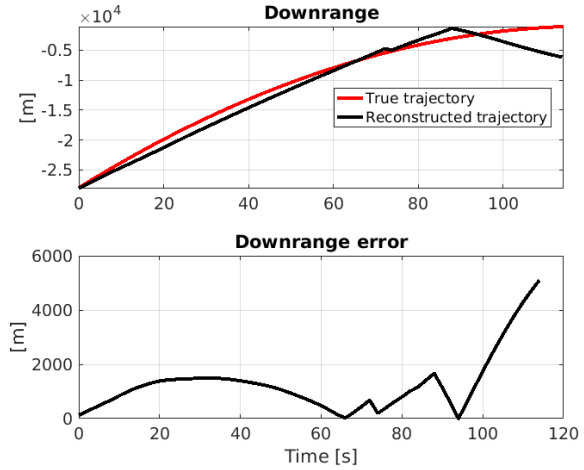
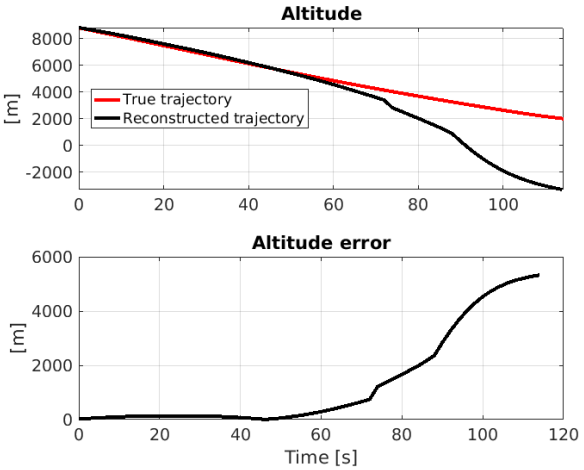


Figure 4.40: *Reconstructed Main Brake trajectory with downward looking attitude with algorithm working at* 30 Hz. *The ambiguity in translation/rotation confused as vertical descent is avoided.*



(a)



(b)

(c)

Figure 4.39: *Downrange, Altitude and Crossrange of the reconstructed trajectory as a function of time along with relative errors.*

The two errors have occurred because of a wrongly estimated homography due to a misinterpreted scene. Problem arises because of the challenging situation encountered in the sequence, and the only way to cope with this ambiguity is to increase the frame rate. Fig. 4.40 shows the obtained trajectory with a 30 Hz frame rate as reference. The ambiguity previously shown is completely avoided, and trajectory is reconstructed with good accuracy even if a quite high drift is present in the final portion. An application of this kind is anyway not possible at the moment, being this frame rate too high to be managed in real-time by the system.

30° Pointing attitude

For the 30° downward camera pointing sequence, this algorithm configuration is not able to coherently reconstruct the trajectory, as shown in Fig. 4.41. Trajectory is anyway really smooth and not affected by gross errors, showing an interesting erroneous behaviour which has already been encountered with the essential matrix configuration: forward motion is completely mistaken as a rotation, and as a result the trajectory is reconstructed as if the camera is rotating around the scene. This condition is intrinsically related to the kind of images and motion which brings algorithm to an ambiguity condition, and is really difficult to be avoided. Downrange, crossrange and altitude errors are not presented since not significant. For what concerns the attitude in Fig. 4.42, quaternions follow exactly the behaviour of the trajectory, underling the misinterpreted rotation which in reality should be a translation.

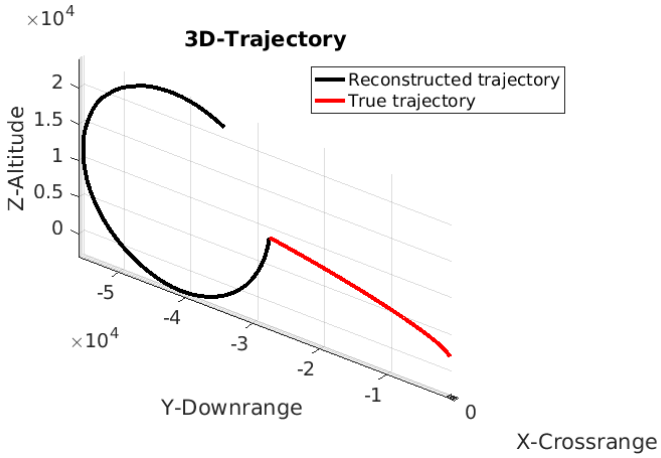


Figure 4.41: *Reconstructed Main Brake trajectory with 30° pointing attitude using H for motion estimation. Forward motion is completely misinterpreted as a rotation of the camera.*

Fig. 4.43 shows the reconstructed trajectory obtained augmenting the frame rate to 30 Hz as previously made for the downward pointing

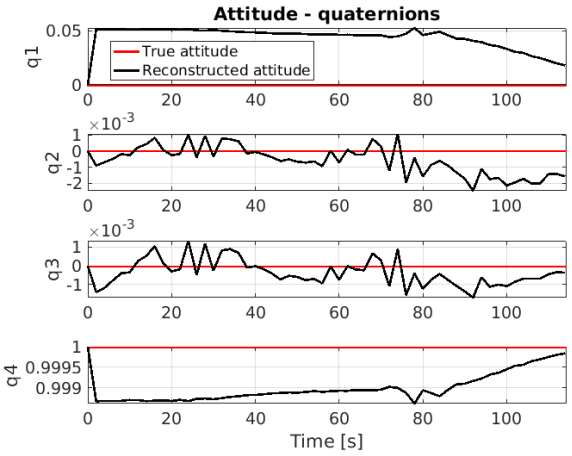


Figure 4.42: *Reconstructed Main Brake attitude for the* 30° *pointing case using H for estimation. Quaternions behaviour exactly underline the misinterpreted rotation.*

sequence. This solution gives better results, with trajectory which has at least a more coherent behaviour. In the final phases anyway, the misinterpreted translation issue still arise making the trajectory completely drift from the true one.

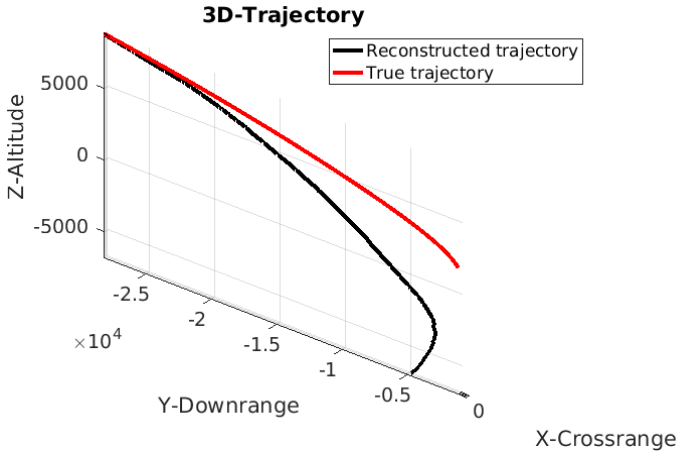


Figure 4.43: *Reconstructed Main Brake trajectory with* 30° *pointing attitude using H, with system working at* 30 Hz*. Misinterpreted translation still occurs at the of the trajectory.*

Pitch maneuver: downward to 30° pointing

The system run on this *Main Brake* sequence is not able to reconstruct the trajectory and shows the same errors observed on the previous one. Here the thing is even more accentuated, since the reconstructed trajectory starts coherently when camera is downward pointing, and as soon as the pitch maneuver occurs, this is completely misinterpreted with a completely wrong translation vectors, and when finally

the camera reaches the 30° pointing, forward motion starts to be interpreted as a rotation. Downrange, crossrange and altitude errors,

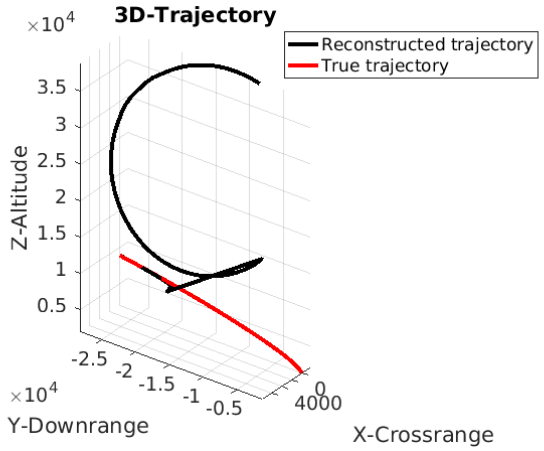


Figure 4.44: *Main Brake trajectory for the pitch maneuver using H for motion estimation. Algorithm has not been able to reconstruct the trajectory.*

along with quaternions behaviour, are not reported for this case since they do not give any additional information.

APPROACH

Considering the *Approach* trajectory, as done for the previous configurations, algorithm has been run on the three different sequences with downward, 30° and nozzle camera pointing.

Downward looking attittude

Despite the different scenario given by the *Approach* sequences, the same problem encountered for the *Main Brake* ones has been encountered. Fig. 4.45 shows how reconstruction is coherent only for the initial frames, then forward motion is interpreted as a rotation and trajectory completely drifts. Errors shown in Fig. 4.47 are lower in magnitude with respect to the *Main Brake* case, but this is only due to the global lower scale of the true trajectory.
Attitude behaviour represented in Fig. 4.46, shows quaternions following coherently and smoothly the reconstructed trajectory behaviour, an explicative fact that the error is not due to a wrongly extracted translation or rotations, but intrinsic of the scenario itself.

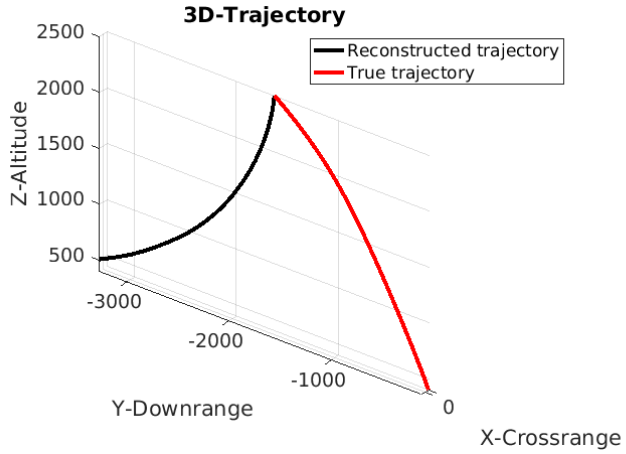


Figure 4.45: *Reconstructed Approach trajectory with downward looking attitude using H for motion estimation. Forward motion is wrongly interpreted as a rotation.*

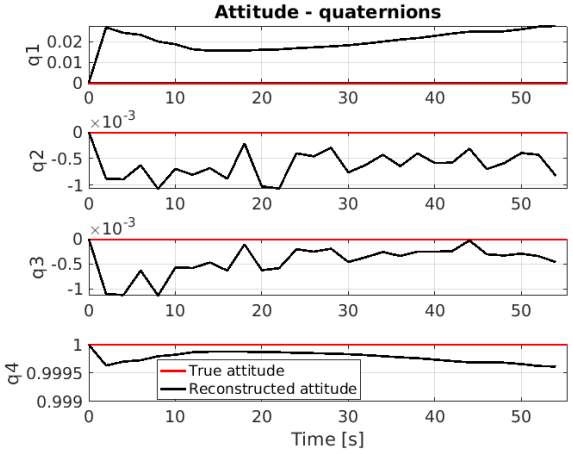
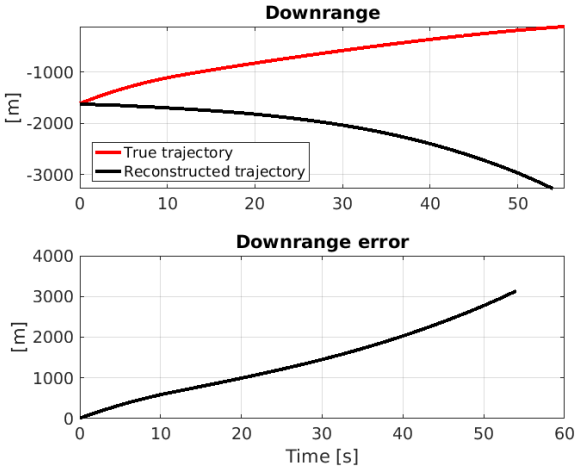
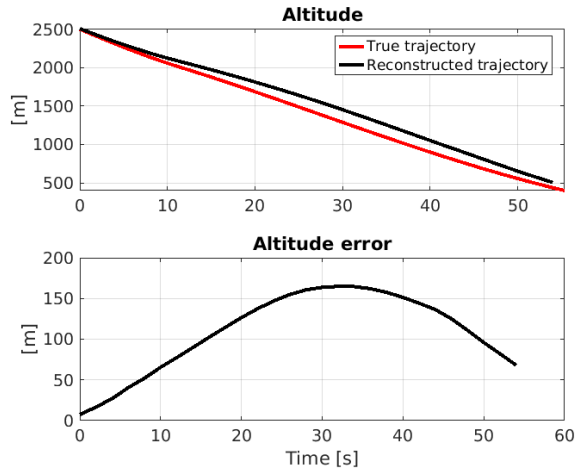


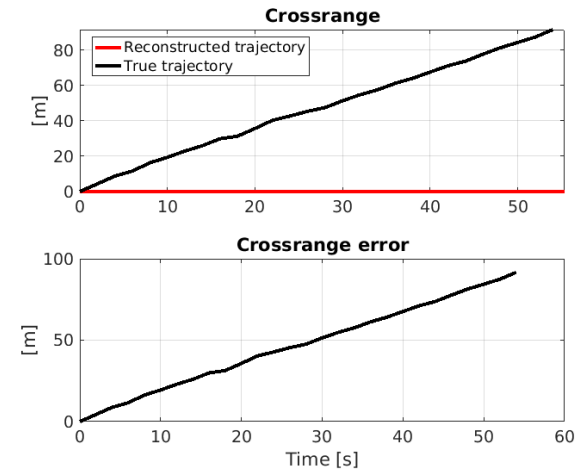
Figure 4.46: *Reconstructed Approach attitude for the downward pointing case using H for estimation. Since attitude is fixed quaternions are expected not change in time.*



(a)



(b)



(c)

Figure 4.47: *Downrange, Altitude and Crossrange of the reconstructed trajectory in function of time along with relative errors.*

30° Pointing attitude

Results obtained for the 30° pointing attitude are still affected by the misinterpreted forward motion problem. Fig. 4.48 shows the obtained trajectory, which shows a behaviour completely similar to the one obtained for the downward looking camera (Fig. 4.45). Attitude

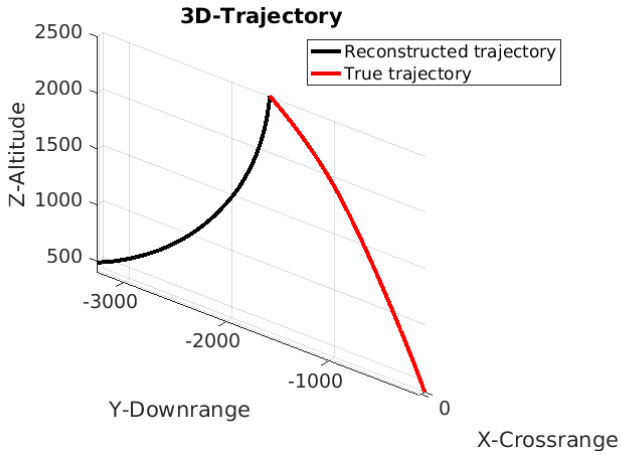


Figure 4.48: *Reconstructed Approach trajectory for the* 30° *attitude sequence using H for motion estimation.*

behaviour is reported in Fig. 4.49. Quaternion $q1$ clearly shows the misinterpreted rotation around crossrange axis.

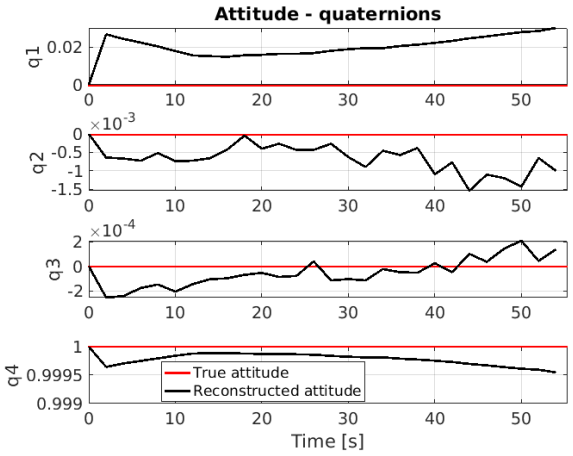


Figure 4.49: *Reconstructed attitude for the* 30° *pointing sequence using H for estimation.*

Nozzle pointing

Reconstructed trajectory for the nozzle pointing attitude case is reported in Fig. 4.50. This variable pointing condition shows an interesting behaviour: for the initial piece of trajectory motion is interpreted as a rotation but on the opposite side with respect to the behaviour obtained with the previous attitudes; as long as the path becomes

more vertical then, reconstructed motion starts to become more coherent with the true one. This behaviour underlines how sensible the algorithm is to the different pointing conditions, with a slight variations of attitude resulting in a wrong retrieved translation/rotation. As already explained, this is a direct consequence of the kind of motion coupled with the scenario seen by the camera (and thus feature distribution). Fig. 4.51 Fig. 4.51 shows the attitude behaviour,

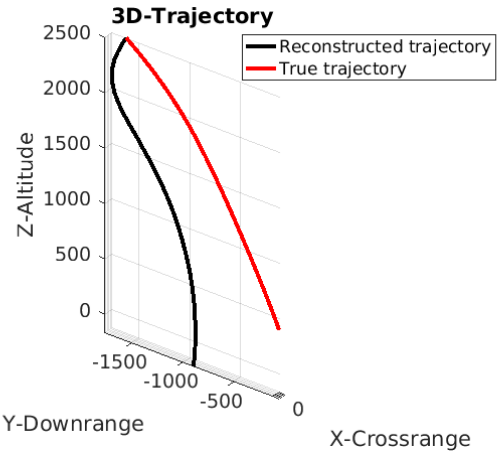


Figure 4.50: *Reconstructed Approach trajectory for the nozzle pointing attitude using H for motion estimation.*

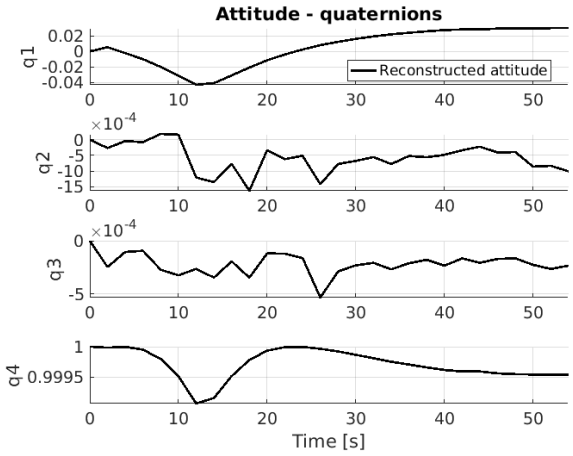


Figure 4.51: *Reconstructed attitude for the nozzle pointing approach sequence using H for estimation.*

with quaternions following the reconstructed trajectory path. Even if incorrect, the fact that quaternions have a coherent behaviour with retrieved motion shows once again that the problem is not in the rotation matrix retrieve but intrinsic of the homography matrix itself which interprets a wrong projectivity between the seen frames.

4.4.5 *Results Overview*

Tests done with the different motion estimation blocks for the current navigation system configuration have highlighted many different pros and cons of each method considered, along with common problems caused by intrinsic characteristics of the landing scenario. First, 8-Point algorithm for fundamental matrix estimation does not work properly on any of the sequences, independently from the setting used. On the other side the fact that trajectory of the KITTI dataset has been properly reconstructed with the same algorithm demonstrates that problems are not related to algorithm itself but instead to the particular scenario faced during landing. Since the algorithm has been developed for planetary landing, it results obvious that the configuration which exploits fundamental matrix for motion estimation has to be discarded.
The other tested configuration, exploiting essential matrix, has been proven working robustly both on Approach and Main Brake trajectories, independently from orientation of the camera. At the current development status this configuration gives the higher performance both in terms of computational burden and accuracy in reconstructed trajectory. The main problems encountered are related to wrongly estimated translations and misinterpreted motions. This problem is anyway intrinsic to the scene observed by the camera coupled with its motion, being something related to projective geometry which cannot be changed, with the only possibility to overcome this problem being to find a way to adapt the algorithm to this condition.
For what concerns the last tested configuration, the one which exploits the homography matrix, interesting results have been obtained for the downward pointing camera with the *Main Brake* trajectory. The other reconstructed sequences are heavily affected by the wrong interpretation of forward motion which is retrieved as a rotation, a problem similar to the one encountered with essential matrix configuration, but which is more important here. Exploitation of homography with downward looking camera is anyway interesting because of the accuracy obtained in the reconstructed trajectory, despite the the final errors when camera moves near the surface.
Using the essential matrix for frame to frame motion estimation appears as the most robust solution among the tried configurations. An interesting alternative might be the one described for bootstrapping of algorithm presented in [32], which proposes an adaptive configuration based on estimation of how planar is the surface seen by the camera. One can think of using homography as long as the landing trajectory is in the main-brake phase, far from the surface when this appears as flat, and switch to essential matrix when moving nearer, in the approach phase, when planarity condition is no more verified.
The proposed architecture represent just a basic configuration for a

VO application, and further steps shall be done to make the algorithm more robust and greatly increase its accuracy in order to make it useful for navigation purposes. Independently from this, results obtained are quite satisfactory making the actual configuration a good benchmark to be used for further development. Table 4.8 gives a simple overview of the obtained results with each configuration:

Trajectory	*Sequence*	*Configuration*		
		F	**E**	**H**
Main Brake	*Downward*	×	√	∼
	30°	×	∼	×
	Down to 30°	×	×	×
Approach	*Downward*	×	√	×
	30°	×	√	×
	Nozzle	×	×	×
KITTI	*"00"*	√	√	//

Table 4.8: *Navigation algorithm: Test results.*

With × symbol indicating a wrongly reconstructed trajectory, ∼ symbol a trajectory almost coherently reconstructed but with some errors, and √ a correctly reconstructed trajectory.

4.5 DISCARDED CONFIGURATION: FEATURE MATCHING

This configuration was the first one initially developed and exploits features matching between couple of frames to obtain the set of correspondences needed by the 2D to 2D motion estimation step to work. This step can then be done using the 8-point, 5-point or DLT method to retrieve respectively fundamental, essential or homography matrix as done with the tracking, which are finally used for motion estimation. Map and refinement techniques are not used and the retrieved relative camera pose is directly used to reconstruct trajectory step by step. The functional scheme of this architecture is reported in Fig. 4.52.

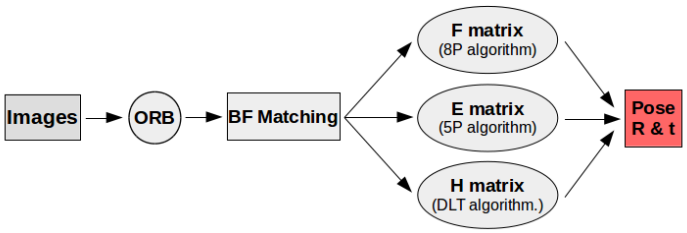


Figure 4.52: *Functional scheme of the algorithm first tested configuration.*

Navigation algorithm set this way works with the following steps:

- First image from the camera is given in input, ORB features are extracted
- Successive image from the camera is given and ORB features are extracted, matching process is then applied according to method suggested by Lowe [38] and a set of features correspondences is obtained.
- From the set of feature correspondences, E, F or H matrix is retrieved within a RANSAC scheme. If fundamental or essential matrix is used, a check on the determinant of the matrix is performed to assess quality of the obtained matrix(i.e. a determinant far from 0 indicates a wrong matrix); if quality is low frame is rejected and process restarts from the previous step, otherwise process continues.
- Once a good F, E or H matrix is retrieved, motion estimation reconstruct rotation matrix R and translation vector t. A check on their consistency is done, if check is negative pose is rejected.
- Algorithm is initialized with the pose between first two frames. An update of the variables is done: the last considered frame becomes the old one to do matching with, and the same for the last extracted features.
- New frame coming from the camera is taken and ORB features are extracted and matched with the previous frame.
- Motion estimation is done as described in the previous steps and obtained pose is concatenated with the already retrieve ones to reconstruct trajectory.

Process goes on iteratively each time a new frame from the camera is given in input: detected features from image one are matched with features from image two, features from two are then matched with features from image three and so on until the frames finish. This process is very simple but needs the feature extraction step to be performed with each new incoming frame.
This configuration of the algorithm is really sensible to the features used, and even a change on one of the ORB parameters may results in a completely erroneous trajectory reconstruction. Working with the parameters used in the previous section is not feasible, since obtained trajectory are too much drifted and full of errors. To show therefore useful results, parameters given in Table 4.9 are used.

Frame rate used is always 0.5 Hz, but the number of sub-windows used for feature detection is reduced to 16. The higher number of features extracted is necessary because with features matching more than a half of the obtained matches are discarded by Lowe's method [**?**] applied to retain only inliers, but a conspicuous number of matches

Number of features	960
Scale factor	1.5
Pyramid layers	4
Edge threshold	20 px
Patch size	20 px
FAST threshold	20

Table 4.9: *ORB feature detector parameters setting.*

is needed by 5-Point algorithm within the RANSAC cycle to work properly. As a consequence to the choice of using more features, the 64 sub-window case is not suitable to work because of problems with the edges between sub-windows, in practice the maximum number of features which can be retained for each sub-window is too low, and for this reason the 16-subwindows configuration has been used. For what concerns the frame rate instead, using matching and working with an higher one is not possible: higher frame rates up to 30 Hz have been tested and in each case forward motion was completely misinterpreted as an altitude loss or a rotation.

4.5.1 *Motion Estimation*

In the following, only the three tests performed on the most meaningful sequences are given, which are the Main Brake with downward pointing camera sequence, the Approach with downward pointing camera one and the KITTI car sequence. Essential matrix and then homography have been used for the motion estimation step, while 8-point algorithm to retrieve fundamental matrix has been proven not be able to work properly with this configuration and therefore is not considered.

4.5.1.1 *Configuration with Essential Matrix*

This configuration of the algorithm exploits 5-point method for estimation of the essential matrix. As the name suggest, at least five point correspondences are needed, giving a minimum threshold value for correct matched features to be obtained during the matching step.

MAIN BRAKE TRAJECTORY

Reconstructed trajectory obtained with the navigation algorithm run on the *Main Brake* trajectory for the downward pointing camera sequence is shown in Fig. 4.53. Accuracy in reconstruction is good

for the first part, then trajectory starts drifting and there are three main errors. Errors are similar to the ones obtained with tracking and homography matrix, forward descending motion is interpreted only as a vertical descent by the algorithm, which estimates a wrong translation.

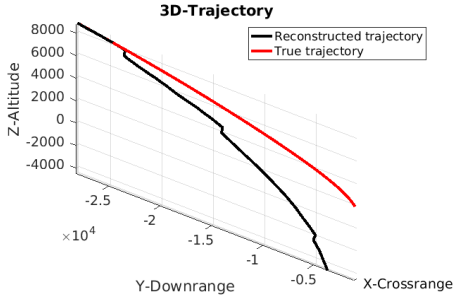


Figure 4.53: *Reconstructed Main Brake trajectory with downward looking attitude using E for motion estimation. There are several points in which forward motion is erroneously interpreted as an altitude loss.*

Considering attitude, it can be seen how three peaks are present in quaternions $q1$ and $q4$ corresponding to the three misinterpreted motion. Since the erroneous pose is retrieve only between a couple of frames, behaviour then comes back to follow the ground truth one.

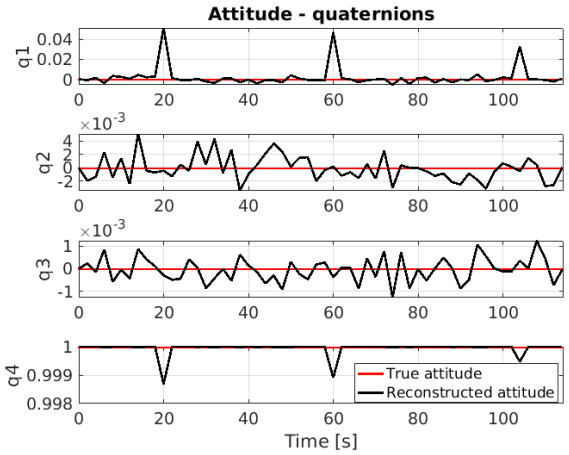
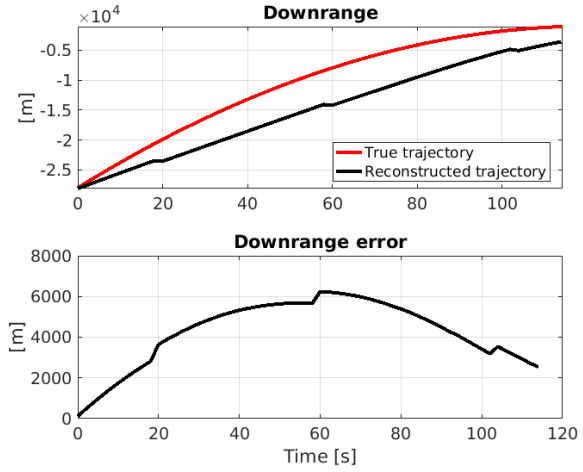
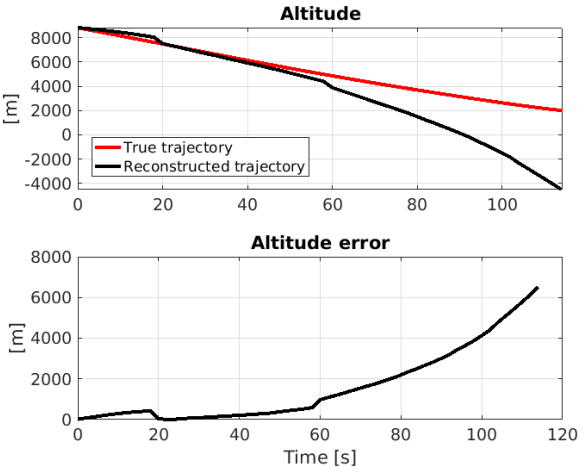


Figure 4.54: *Reconstructed Main Brake attitude for the downward pointing case using E for estimation. Since attitude is fixed quaternions do not change in time.*

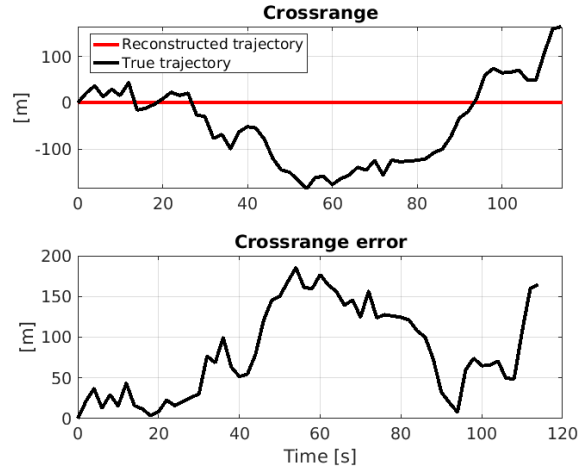
Fig. 4.55 shows the downrange, altitude and crossrange behaviour as a function of time. Overall drift is in some way contained for the cross range; concerning altitude, it remains bounded at first, then grows after the first erroneous retrieved pose. Downrange drift instead, grows since since the beginning at first to come back near ground truth behaviour at the end. This is done to difference in curvature obtain for the reconstructed trajectory.



(a)



(b)



(c)

Figure 4.55: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

APPROACH TRAJECTORY

Results obtained for Approach trajectory reconstruction are reported. Navigation algorithm has been run on the downward pointing camera sequence. Fig. 4.28 shows the reconstructed trajectory. Accuracy is good for almost all the motion, with an error occurring in the final part: when the camera is near the ground small forward descending motion is misinterpreted as an altitude loss, giving rise to the step which can be observed in figure. In Fig. 4.30, errors for downrange, crossrange and altitude are reported, showing the trajectory drift in times. Globally, errors are quite small due to the good accuracy in trajectory reconstruction and due to the smaller scale of the scenario seen by the camera. To conclude, Fig. 4.29 reports the attitude behaviour, which shows only a bigger oscillation in correspondence of the wrong altitude loss estimation.

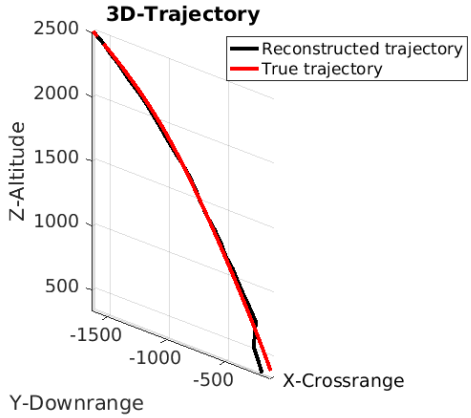


Figure 4.56: *Reconstructed Approach trajectory for the downward looking camera sequence using E for motion estimation. A wrong pose estimation occurs in the end part.*

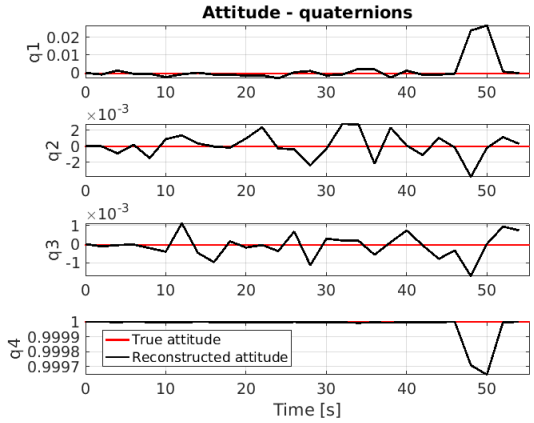
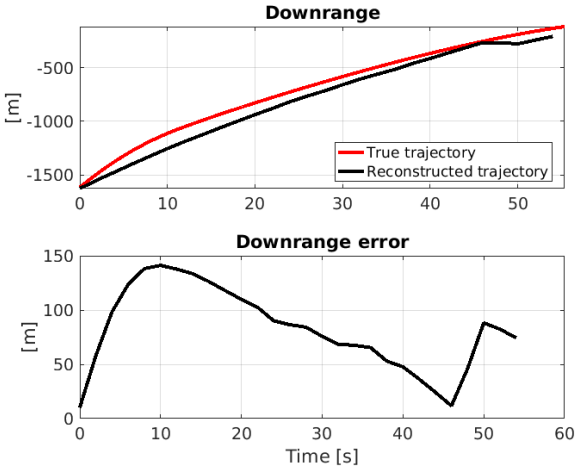
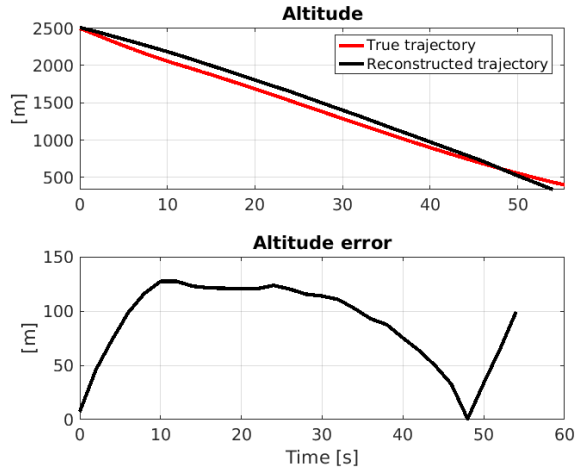


Figure 4.57: *Reconstructed Approach attitude for the downward pointing sequence using E for estimation.*



(a)



(b)

(c)

Figure 4.58: *Downrange, Altitude and Crossrange of the reconstructed trajectory as a function of time along with relative errors.*

KITTI DATASET

Algorithm has been run on the first part of sequence 00 of the KITTI dataset as done for the configuration of Section 4.4.1. Reconstructed trajectory is shown in Fig. 4.59.

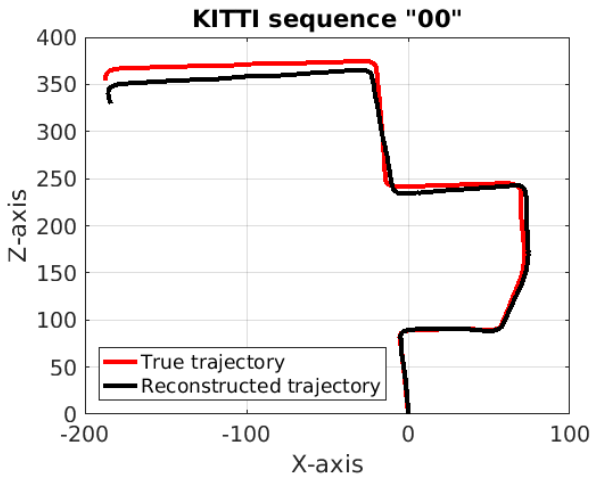


Figure 4.59: *Reconstructed partial KITTI trajectory (first* 1000 *frames) for sequence* 00 *using E for motion estimation.*

Overall obtained accuracy in recosntruction is quite high, with no gross errors due to wrongly estimated pose rotations. It is observable only a progressive drift which is maximum in correspondence of the last turn. Once again the algorithm has prove to manage with no problem such a kind of scenario, despite instead the difficulties in coherently reconstructing the Lunar landing trajectories.

4.5.1.2 *Configuration with Homography Matrix*

Algorithm configuration exploiting homography for the motion estimation has been run on both Main Brake and Approach trajectories different sequences with changed attitude. Results for the sequence with downward looking camera are the most representative and the only one here presented.

MAIN BRAKE TRAJECTORY

Reconstructed Main Brake trajectory for the downward pointing camera sequence is reported in Fig. 4.60. Trajectory is reconstructed smoothly, but two main errors are present, the first of which occurs during initialization: image of the Main Brake sequence is partially black for the first frames since rendered that way, thus it is initially possible to detect only a minor number of features, resulting in an

easier possibility to fall in the ambiguity with forward motion misinterpreted as a loss of attitude. Second error is instead near the end of the path, when camera moves near the surface, and completely wrong translations are retrieved making the trajectory completely drift. Fig. 4.62 shows the downrange, crossrange and altitude behaviour in time along with their relative errors. Progressive drift in time can be observed, while for downrange the final wrong reconstruction is particularly evident. Quaternions behaviour is shown in Fig. 4.61. In $q1$ and $q4$ the initial misinterpreted altitude loss can be observed, and the same for the final part, evidenced by the sudden step.

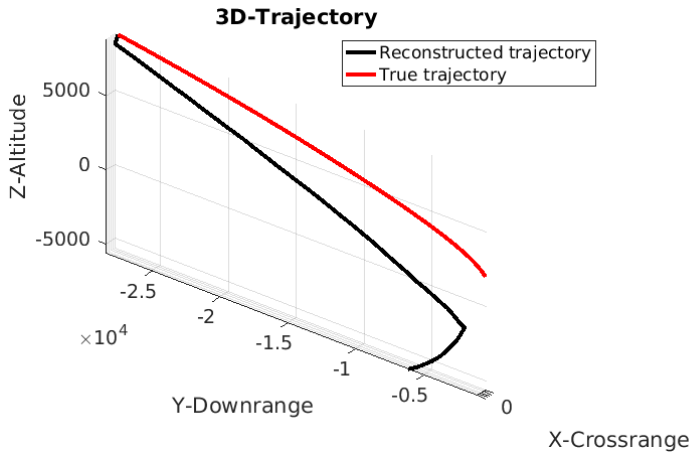


Figure 4.60: *Reconstructed Main Brake trajectory with downward looking attitude using H for motion estimation.*

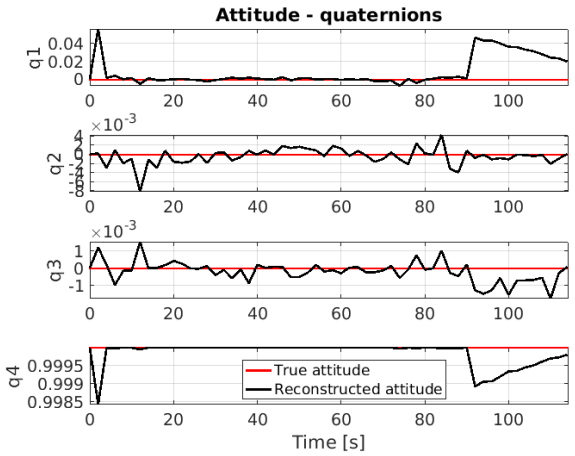
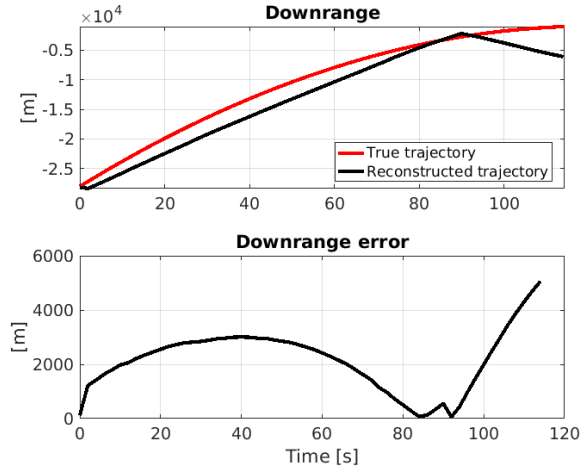
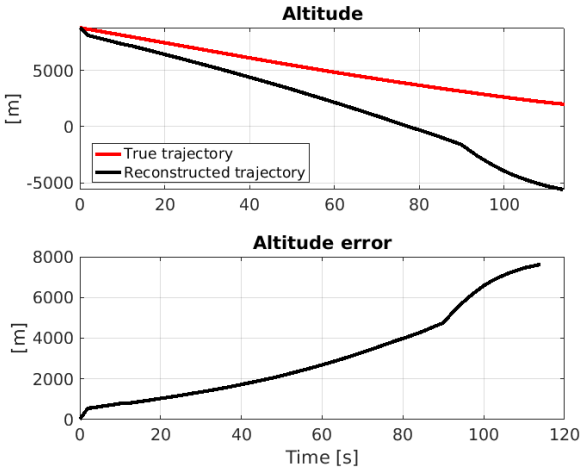


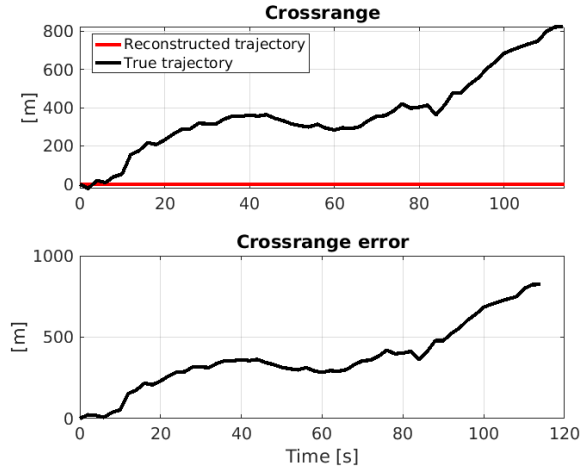
Figure 4.61: *Reconstructed Main Brake attitude for the downward pointing sequence using H for estimation. The two error in reconstruction are clearly visible.*



(a)



(b)



(c)

Figure 4.62: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with their relative errors.*

APPROACH TRAJECTORY

Approach trajectory reconstructed with homography matrix is shown in Fig. 4.63. Despite the whole curve is smooth and without gross errors, the drift from real motion is quite high. In a similar way as for the configuration exploiting feature tracking, the forward motion is misinterpreted by homography as a rotation around camera x axis and this causes the whole trajectory drift. This situation underlines that homography does not works properly for this kind of scenario when the altitude is low and the scale reduced.

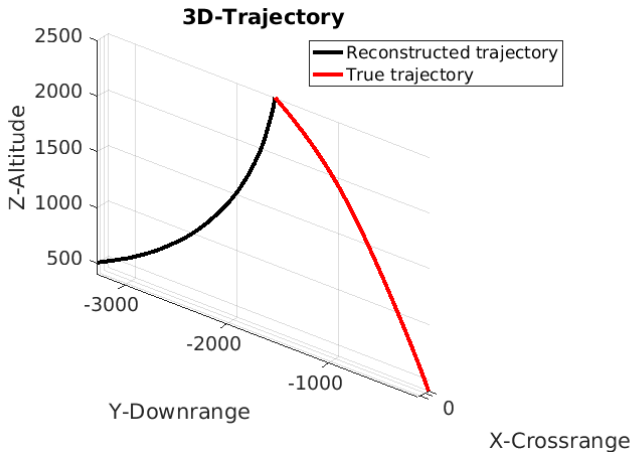


Figure 4.63: *Reconstructed Approach trajectory sequence with downward looking camera using H for motion estimation. Forward motion is again wrongly interpreted as a rotation.*

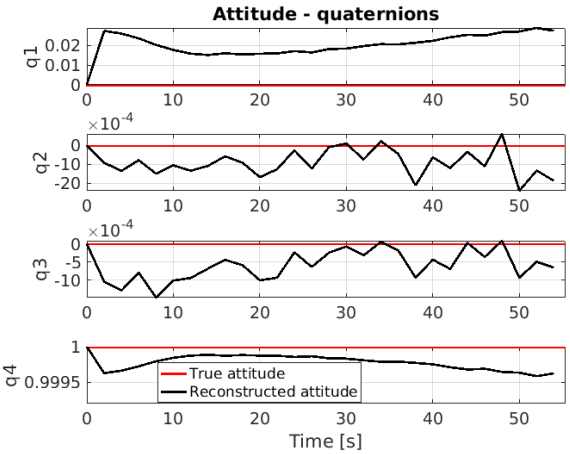
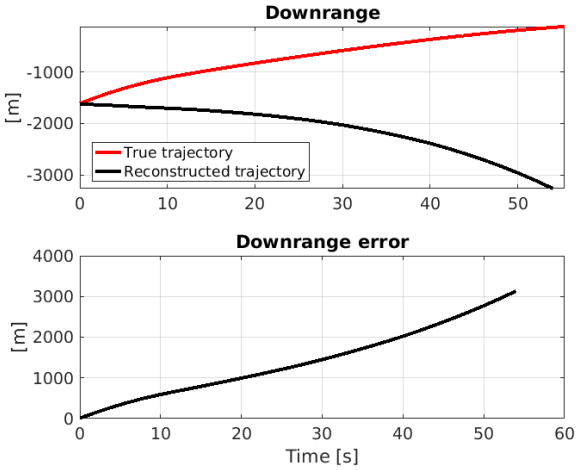
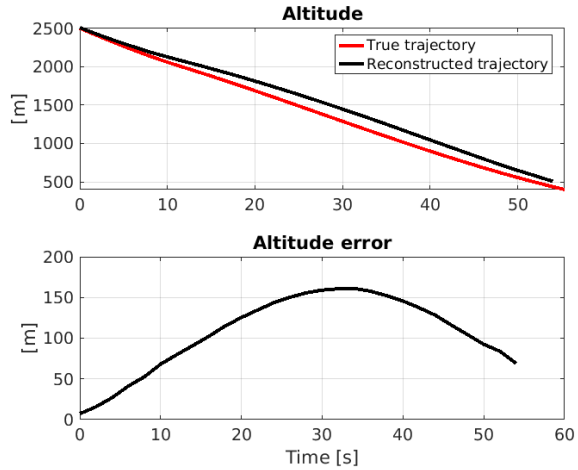


Figure 4.64: *Reconstructed Approach attitude for the downward pointing sequence using H for estimation. The wrongly interpreted rotation can be easily visualized for $q1$ and $q4$.*

Fig. 4.65 shows the dowrange, crossrange and altitude behaviour along with errors. As could be expected the major drift occurs for



(a)



(b)

(c)

Figure 4.65: *Downrange, Altitude and Crossrange of the reconstructed trajectory along with relative errors.*

the downrange, which is completely wrongly reconstructed by the algorithm. For what concerns the attitude, quaternions behaviour in time is shown in Fig. 4.64. From $q1$ and $q4$ it can be easily observed the misinterpreted rotation around the x axis.

4.5.2 *Overview*

Despite the good quality of the obtained results, showing the capability of the system configuration exploiting matching to work with the two Lunar landing trajectories and with the KITTI dataset one, this scheme has been discarded for different reasons. System built this way first of all has demonstrated an high sensitivity to detected features, being not robust to any change in the ORB parameters setting. For the essential matrix configuration, the setting presented is basically the only one which makes algorithm obtaining sufficiently good results. Moreover, no other image frame rate then the one used can be managed by the algorithm. Homography matrix configuration instead, has shown an higher flexibility, being able to cope both with higher frame rates and different ORB settings. Despite this, using features matching with ORB feature detection at each step results in an high computational cost, which coupled with the high number of features required results to be difficult to be used in real-time and which does not justify its choice against the feature tracking configuration. Moreover, working with new features at each frame makes also harder to predict which features have been seen not only in the last but also in the previous frames, an information which is quite useful (if not indispensable) both for a map construction or a Bundle Adjustment implementation. It has to be anyway underlined that the matching method shall not be completely discarded, since it can be an extremely useful tool to be used for future developments of state of the art mapping implementations as the one proposed in [32].

5

CURRENT STATUS AND FUTURE DEVELOPMENT

At the current development status, algorithm configuration is not yet fixed, but some of its building blocks have been defined, yielding to a first Visual Odometry configuration capable of giving acceptable trajectories reconstruction, even if not with the sufficient accuracy to be directly used for navigation. Fig. 5.1 shows the actual situation.

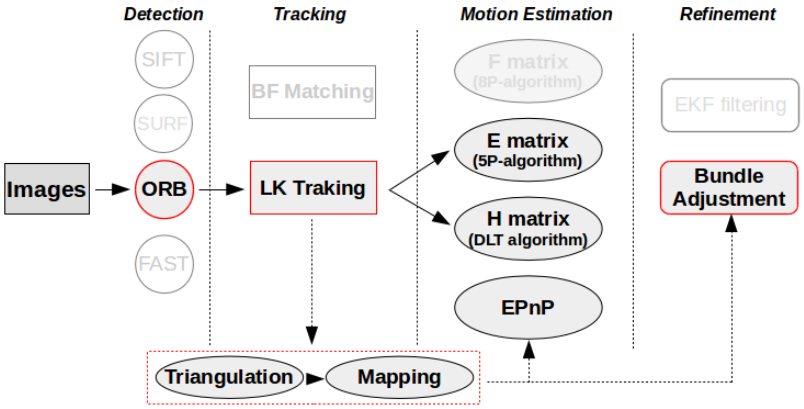


Figure 5.1: *Current implemented and available tools, red highlighted blocks represent a fixed choice, dashed arrows indicate blocks to be developed.*

ORB features and Lucas-Kanade tracking have been individuated as the best choice against other alternative methods for this application. 8-point, 5-point and DLT methods have been extensively tested under different situations, starting from synthetic cloud of points to conclude with different trajectories (Approach, Main Brake and KITTI dataset), also using different camera pointing conditions. EPnP has been tested on synthetic point clouds; application on synthetic images and implementation on navigation algorithm have been developed but are yet not complete and here not shown. The same holds for the mapping, which has been implemented and tested using methods for triangulation described in Chapter 3, but that is not yet good enough to be used to properly reconstruct trajectory and is therefore here not shown. Optimization techniques, in particular Bundle Adjustment, are under testing and further development are necessary to make them functional in real-time on the algorithm.

Building a complete Vision-based Navigation algorithm involves a lot of work due to the multidisciplinary nature of the argument and due to the great number of different tools to be developed and

implemented. What done up to now represent just the first phase of the work, basically the skeleton to be used as a starting point for a more sophisticated and detailed development.

5.1 FUTURE DEVELOPMENT PROGRAM

Since the optical navigation problem has been extensively studied and examined in its details, and given the wide theoretical background considered, final objective navigation algorithm configuration to be obtained is known. A developed program is therefore in the following proposed, organized in phases, aimed at defining the project in its completeness. At the end of each presented phase extensive test of the navigation algorithm shall be done on the Lunar dataset in order to asses and validate performance improvement. *Phase-0* is represented by the work done in this thesis and is here shortly reported for completeness.

PHASE 0 Theoretical background, definition of the project objectives and of the available tools. Test and validation of the different algorithms and methods under different scenarios, implementation of one or more starting configurations. Trade-off between the results obtained and definition of a basic configuration.

PHASE 1 Enhancement of the configuration defined in Phase-0, refinement of the structure of the algorithm. Some tips may be the following:

- ORB features as implemented now are efficient but an improvement in the way sub-windows are managed may be done to save computational time, moreover an adaptive implementation which rises the number of features in the other sub-windows when in one not enough of them are detected may increase robustness of the whole algorithm.
- LK traking as implemented works well, but at the moment each time number of features drops, the process is completely reset starting from new features. This is not efficient, since despite the drop in number, lots of features are still being correctly tracked. An implementation winch retains still tracked features and extracts new ORB one adding these to the already implemented would be better, increasing globally the number of features continuously tracked.
- Due to navigation algorithm intrinsic demonstrated sensitivity to changes in the parameters settings, a deeper sensitivity analysis on all the parameters shall be done in order to find an optimal configuration which maximize performance.

PHASE 2 In this phase three main development step shall be done:

- **Mapping**: Final objective is to have a sparse coherent map representing the observed environment by the camera made of $3D$ features. Map shall be constantly updated and optimized in order to increase accuracy. A policy to discard wrongly triangulated points and outliers has to be implemented.
- **Bundle Adjustment**: BA shall be implemented in the algorithm and made properly work in motion only and full configuration. Objective is to have an implementation in which full BA works to optimize the built map while sliding window BA for motion only optimization is used to increase trajectory accuracy.

Once both mapping and optimization are developed, implementation in multi-thread processing of the navigation system is the next step. This is a fundamental objective in order to enhance performance while still be able to work in real-time.

PHASE 3 Once *Phase-2* is completed an almost final configuration of the algorithm shall be available and functional, able to robustly work on spacecraft landing scenarios. A further development step is implementation of data fusion with other sensors:

- **IMU**: There are two basic ideas to make such fusion: the most simple one is to simply make a data fusion of camera and IMU data with an EKF filter. Something more efficient would be instead to make the navigation algorithm rely on measurements from the IMU to enhance its attitude and motion informations.
- **Laser Altimeter**: A further optional step would be integration of data also from a laser altimeter, that would greatly increase accuracy in features triangulation and thus in the map reconstruction.

PHASE 4 Once data integration with other sensors is done, the final and last development step would be integration with the other subsystem of the AGNC chain. Vision based navigation shall supply important informations to the Guidance and to the Hazard Detection and Avoidance subsystems, sharing informations of the computed trajectory and 3D map. (e.g. sharing of the sparse map with HDA, which can exploit it to obtain slopes and roughness of the observed scene to better choose the landing spot).

PHASE 5 Final development phase is represented by extensively testing of the Visual Odometry Navigation Algorithm on the experimental facility under development at Department of Aerospace, Science and Technology (DAER) at Politecnico. Facility is composed by a robotic

arm carrying the sensors suite to simulate lander dynamics, a 3D planetary mock-up, an illumination system, control and test computers. This system is designed to verify both hardware and software up to TRL 4 [61]. This step is anyway not trivial, since an adaptation of the algorithm to be integrated with the hardware has to be done. The main factor to be considered is integration of the algorithm with the monocular camera, which sends images to it in real time.

It has to be considered that all the outlined phases represent a guideline: at each step, after testing, any needed changes may be applied if some of the outlined improvement does not give efficient results.

6

CONCLUSIONS

Development of an autonomous Optical Navigation System for planetary landing working with a monocamera has been proposed in this work. A wide and extensive theory background has been considered and expounded along with the possible tools to be used to construct the vision based navigation system. Both the space field and the computer vision one applied to robotics have been considered. Available studied tools have been analysed and tested on synthetic generated cloud of points in order to assess their behaviour and make a trade-off between them. Current Navigation Algorithm configuration works extracting salient features from the incoming images, which are then tracked frame by frame. This gives rise to a set of bi-dimensional correspondences which is exploited by a dedicated algorithm to retrieve pose, that is rotation and translation, between couple of frames. Results extracted this way are concatenated together to reconstruct spacecraft trajectory in time up to a scale factor. Testing and performance assessment of the navigation system with different possible configurations has been performed on synthetic image sequences of spacecraft Lunar landing trajectories. Moreover, in order to validate the system under a different scenario and test its behaviour in a condition more similar to the one for which its constituent algorithms were developed, tests have been done also on an image dataset developed for computer vision of a moving car equipped with camera. Navigation algorithm has shown good performance on these sequences, being able to reconstruct almost all trajectories with good accuracy and respecting the requirements defined in Chapter 1. Sequences that have not been properly reconstructed, represent a quite challenging benchmark for the navigation system and have highlighted a problem which is intrinsic of the considered scenario: observing an almost planar surface from a large distance easily lead vision based navigation to misinterpret rotations and translations. Nevertheless, results obtained have also demonstrated the capability of algorithms born to be used in computer vision and usually applied to completely different kind of environments to efficiently work also on a planetary landing scenarios. To conclude the work, a detailed development program describing the work to be done in order to complete and enhance performance of the navigation system has been given.

BIBLIOGRAPHY

[1] Richard Hartley and Andrew Zisserman, *Multiple view geometry in computer vision*, Cambridge university press, 2003.

[2] Edward Rosten and Tom Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006*, pp. 430–443. Springer, 2006.

[3] "Chandrayaan Programme website," http://www.chandrayaan-i.com/index.php/chandrayaan-2.html, Last visit: 10th Feb. 2016.

[4] "Viking Programme website," http://nssdc.gsfc.nasa.gov/planetary/viking.html, Last visit: 23th Jun. 2016.

[5] "Mars Science Laboratory Programme website," http://mars.nasa.gov/msl/, Last visit: 24th Jun. 2016.

[6] "ExoMars Programme website," http://exploration.esa.int/mars/, Last visit: 24th Jun. 2016.

[7] T Miso, Tatsuaki Hashimoto, and Keiken Ninomiya, "Optical guidance for autonomous landing of spacecraft," *IEEE Transactions on aerospace and electronic systems*, vol. 35, no. 2, pp. 459–473, 1999.

[8] "Rosetta Programme website," http://sci.esa.int/rosetta/, Last visit: 23th Jun. 2016.

[9] "Osiris-Rex Programme website," http://www.asteroidmission.org/, Last visit: 23th Jun. 2016.

[10] Lavagna Lunghi, "Autonomous vision-based hazard map generator for planetary landing phases," 2014.

[11] Yang Cheng, Mark Maimone, and Larry Matthies, "Visual odometry on the mars exploration rovers," in *2005 IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 2005, vol. 1, pp. 903–910.

[12] Francesco Castellini, David Antal-Wokes, R Pardo de Santayana, and Klaas Vantournhout, "Far approach optical navigation and comet photometry for the rosetta mission," in *Proceedings of the 25th International Symposium on Space Flight Dynamics (ISSFD), Munich, Germany*, 2015.

[13] R Pardo de Santayana and M Lauer, "Optical measurements for rosetta navigation near the comet," .

[14] PH Smith, B Rizk, E Kinney-Spano, C Fellows, C d'Aubigny, and C Merrill, "The osiris-rex camera suite (ocams)," in *Lunar and Planetary Science Conference*, 2013, vol. 44, p. 1690.

[15] Davide Scaramuzza and Friedrich Fraundorfer, "Visual odometry [tutorial]," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 4, pp. 80–92, 2011.

[16] Hugh Durrant-Whyte and Tim Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[17] G Capuano, M Severi, ED Sala, R Ascolese, C Facchinetti, and F Longo, "Compact and highperformance equipment for vision-based navigation," in *63rd International Astronautical Congress (IAC)*, 2012.

[18] K. Hornbostel M-Dunstan, "Image processing chip for relative navigation for lunar landing," in *GNC 2014: 9th international ESA conference on Guidance, Navigation & Control Systems*, 2014.

[19] Friedrich Fraundorfer and Davide Scaramuzza, "Visual odometry: Part ii: Matching, robustness, optimization, and applications," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 2, pp. 78–90, 2012.

[20] Scott A Striepe, Chirold D Epp, and Edward A Robertson, "Autonomous precision landing and hazard avoidance technology (alhat) project status as of may 2010," 2010.

[21] Timothy G McGee, Thomas B Criss, Paul Rosendall, Adrian Hill, Wen-Jong Shyong, Nishant Mehta, Cheryl Reed, Greg Chavers, Michael R Hannan, and Chirold Epp, "Aplnav: development status of an onboard passive optical terrain relative navigation system," in *Proceedings of AIAA Guidance, Navigation, and Control Conference. Kissimmee Florida: AIAA*, 2015.

[22] Stephen R Steffes, Stephan Theil, Michael Dumke, David Heise, Marco Sagliano, Malak A Samaan, Erik Laan, Murat Durkut, Tom Duivenvoorde, David Nijkerk, et al., "Sinplex: A small integrated navigation system for planetary exploration," 2013.

[23] Stephen Steffes, Michael Dumke, David Heise, Marco Sagliano, Malak Samaan, and Stephan Theil, "Target relative navigation results from hardware-in-the-loop tests using the sinplex navigation system," 2014.

[24] Andrew E Johnson, Yang Cheng, James Montgomery, Nikolas Trawny, Brent E Tweddle, and Jason Zheng, "Design and analysis of map relative localization for access to hazardous landing sites on mars," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 0379.

[25] V Simard Bilodeau, S Clerc, R Drai, and J de Lafontaine, "Optical navigation system for pin-point lunar landing," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 10535–10542, 2014.

[26] Leena Singh and Sungyung Lim, "On lunar on-orbit vision-based navigation: terrain mapping, feature tracking driven ekf," in *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, Hawaii.(Cited at pages 20, 21, 22, 25, 29, and 120.)*, 2008.

[27] Cedric Cocaud and Takashi Kubota, "Autonomous navigation near asteroids based on visual slam," in *Proceedings of the 23rd International Symposium on Space Flight Dynamics, Pasadena, California*, 2012.

[28] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, 2007.

[29] David Nistér, Oleg Naroditsky, and James Bergen, "Visual odometry for ground vehicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[30] Georg Klein and David Murray, "Parallel tracking and mapping for small AR workspaces," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 2007, pp. 225–234.

[31] Christian Forster, Matia Pizzoli, and Davide Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

[32] Raul Mur-Artal, JMM Montiel, and Juan D Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *Robotics, IEEE Transactions on*, vol. 31, no. 5, pp. 1147–1163, 2015.

[33] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.

[34] Martin A Fischler and Robert C Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[35] Chris Harris and Mike Stephens, "A combined corner and edge detector.," in *Alvey vision conference*. Citeseer, 1988, vol. 15, p. 50.

[36] Hans P Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover.," Tech. Rep., DTIC Document, 1980.

[37] Jianbo Shi and Carlo Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, 1994, pp. 593–600.

[38] David G Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[39] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool, "Surf: Speeded up robust features," in *Computer vision–ECCV 2006*, pp. 404–417. Springer, 2006.

[40] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[41] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2564–2571.

[42] Simon Baker and Iain Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[43] Gary Bradski and Adrian Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*, " O'Reilly Media, Inc.", 2008.

[44] Jean-Yves Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, pp. 4, 2001.

[45] David Nistér, "An efficient solution to the five-point relative pose problem," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 6, pp. 756–770, 2004.

[46] Ezio Malis and Manuel Vargas, "Deeper understanding of the homography decomposition for vision-based control," 2007.

[47] Richard I Hartley and Peter Sturm, "Triangulation," *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[48] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, pp. 155–166, 2009.

[49] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic robotics*, MIT press, 2005.

[50] Tim Bailey and Hugh Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[51] Andrew J Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 1403–1410.

[52] Hauke Strasdat, José MM Montiel, and Andrew J Davison, "Visual slam: why filter?," *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[53] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon, "Bundle adjustment—a modern synthesis," in *Vision algorithms: theory and practice*, pp. 298–372. Springer, 1999.

[54] Chris Engels, Henrik Stewénius, and David Nistér, "Bundle adjustment rules," *Photogrammetric computer vision*, vol. 2, pp. 124–131, 2006.

[55] G Grisetti, H Strasdat, K Konolige, and W Burgard, "g2o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation*, 2011.

[56] Christopher Zach, "Robust bundle adjustment revisited," in *European Conference on Computer Vision*. Springer, 2014, pp. 772–787.

[57] Manolis Lourakis and Antonis Argyros, "The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm," Tech. Rep., Technical Report 340, Institute of Computer Science-FORTH, Heraklion, Crete, Greece, 2004.

[58] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[59] Daniel Lélis Baggio, *Mastering OpenCV with practical computer vision projects*, Packt Publishing Ltd, 2012.

[60] "LROC website," `http://wms.lroc.asu.edu/lroc/rdr_product_select`, Last visit: 7th Jul. 2016.

[61] Marco Ciarambino, Paolo Lunghi, Luca Losi, and Michele Lavagna, "Development, validation and test of optical based algorithms for autonomous planetary landing," 2016.

[62] "Persistence of vision pty. ltd. (2004). persistence of vision (tm) raytracer. persistence of vision pty. ltd., williamstown, victoria, australia.," http://www.povray.org/.

[63] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

Appendices

A

APPENDIX A: MATHEMATICS

A.1 SINGULAR VALUE DECOMPOSITION (SVD)

The following discussion is obtained as brief summary from [1], the interested reader is encouraged to consult the book for more details.

Singular Value Decomposition (SVD) is a classical method for matrix decomposition typically used for the solution of over-determined system of equations.
Given a square matrix A, the SVD is the factorization of A as:

$$A = U \cdot D \cdot V^T \tag{A.1}$$

where U and V are orthogonal matrices, and D is a diagonal matrix with non-negative entries in descending order. SVD exists also for non square matrix, the most interesting case being the one when there are more rows m than columns n. In this case a decomposition of matrix A $m \times n$ exist and is $A = UDV^T$, with U $m \times n$ matrix with orthogonal columns, D $n \times n$ diagonal matrix and V $n \times n$ orthogonal matrix.
The diagonal entries of matrix D in the SVD are non-negative and are known as *singular values* of matrix A. These are not the eigenvalues of A, but are related to them, it can be demonstrated in fact that:

$$A^T A = V D^2 V^{-1} \tag{A.2}$$

which is the defining equation for eigen values, meaning that the entries of D^2 are the eigen values of $A^T A$ and the columns of V its eigenvectors. In short, the singular values of A are the square-roots of the eigenvalues of $A^T A$.

A brief discussion of the computational time effort needed for SVD decomposition is also appropriate. Whole computational cost for SVD is quite high depending on the number of entries of the matrix to be decomposed. Most of this time is required by the estimation of the U matrix and this is particularly true for systems of equations with many more rows than columns. There are anyway situations in which calculation of U can be avoided since solution of interest may be found in matrix V (e.g. linear triangulation method). For this reason it is important to consider the application for which SVD has being used in order not to waste computational time for useless operations.
Different algorithm implementations exists and can be found both in Matlab® and OpenCV-3.1.0 for the SVD decomposition and are here

not reported.

A.2 LEVENBERG-MARQUARDT

Levenberg-Marquardt method can be considered as a variation of the Gauss-Newton method. It is basically an implementation of Gauss-Newton with a steepest descent technique. When the current solution is far from the correct one, the algorithm behaves as a steepest descent method, slow but guaranteed to converge. When the correct solution is close to the correct solution, it becomes a Gauss Newton method [57].
Description here proposed is taken from [57] and [1]. Let f be a functional relation which maps a parameter vector $\mathbf{p}\epsilon\mathbb{R}^m$ to an estimated measurements vector $\hat{\mathbf{x}} = f(p)$, $\mathbf{x}\epsilon\mathbb{R}^n$. Given an initial estimate p_0 and measurement $\mathbf{x}_0$, objective is to find $\mathbf{p}^+$ that minimizes the square distance $\varepsilon^T\varepsilon$, with $\varepsilon = \mathbf{x} - \mathbf{x}^T$.
f is linearly approximated in the neighborhood of p with a Taylor expansion:

$$f(\mathbf{p} + \delta_p) \approx f(\mathbf{p}) + \mathbf{J}\delta_p \tag{A.3}$$

with $\mathbf{J}$ Jacobian matrix $\dfrac{\partial f(\mathbf{p})}{\partial \mathbf{p}}$.
Being a non-linear optimization method, Levenberg Marquardt is iterative. At each optimization step it finds the $\partial\mathbf{p}$ that minimizes $||\mathbf{x} - f(\mathbf{p} + \delta\mathbf{p})|| \approx ||x - f(\mathbf{p}) - \mathbf{J} \times \delta_\mathbf{p}|| = ||\epsilon - \mathbf{J}\delta\mathbf{p}||$.
Minimum is obtained when $\mathbf{J}\delta_\mathbf{p} - f$ is orthogonal to the column space of $\mathbf{J}$, this yield to the so called *normal equations*:

$$\mathbf{J}^T\mathbf{J}\delta_\mathbf{p} = \mathbf{J}^T\epsilon \tag{A.4}$$

This is the normal equation classical of Gauss-Newton, LM solves a variation of this one known as *augmented normal equation*:

$$(\mathbf{J}^T\mathbf{J} + \lambda I)\delta_\mathbf{p} = \mathbf{J}^T\epsilon \tag{A.5}$$

for some $\lambda > 0$. This strategy is called *damping*, and λ are the damping terms.
If the updated parameter vector $\mathbf{p} + \delta_\mathbf{p}$ with $\delta_\mathbf{p}$ from Eq. A.4 gives a reduction of the error ϵ, update is accepted and process iterated with a reduced damping term. Otherwise, damping term is increased, augmented normal equations are solved again and process is repeated until a reduction of ϵ. Damping term λ is adjusted at each iteration to assure a reduction of the error, if damping is set to a large value, then $(\mathbf{J}^T\mathbf{J} + \lambda I)$ in Eq. A.5 becomes nearly diagonal and the LM update step $\delta_\mathbf{p}$ is near the steepest descent direction. If the damping is small, LM step approximates the quadratic step of a linear problem.
Algorithm terminates when magnitude of the gradient in the normal

equations drops below some threshold ε_1, when the change in magnitude of $\delta_{\mathbf{p}}$ drops below some threshold ε_2 or when a maximum number of iterations is reached.
Note that if a covariance matrix Σ_x for the measurements vector $\mathbf{x}$ is available, it can be incorporated in the LM algorithm generating the *weighted normal equations*:

$$\mathbf{J}^T\Sigma_x^{-1}\mathbf{J}\delta_{\mathbf{p}} = \mathbf{J}^T\Sigma_x^{-1}\epsilon \tag{A.6}$$