Politecnico di Torino

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

MSc in Mechatronic Engineering

Robot Navigation using a Single Camera

Supervisors

Prof. Basilio Bona

Prof. Matteo Matteucci

Dott. Luca Carlone

Author

Ludovico Orlando Russo

Academic Year 2012/2013

A mio nonno Orlando,
che mi ha sempre spinto a pensare a modo mio.

Abstract

Computer vision approaches are increasingly present in robotic systems, since they allow to obtain a very good representation of the environment around the robot by using low power and cheap sensors. In particular, computer vision approaches have shown they can compete with standard and costly solutions based on laser scanners when dealing with the problem of simultaneous localization and mapping (SLAM), where the robot aims at exploring an unknown environment by building a map of the environment and localizing itself in that map. Aim of this Master Thesis is to analyze a solution able to perform SLAM using a common monocular camera sensor, and implementing this solution in a working prototype using well known open source frameworks and libraries for robotic systems.

Sommario

Gli approcci nel campo della computer vision stanno diventando sempre più presenti nei sistemi di robotica, in quanto permetto di ottenere un rappresentazione molto buona dell'ambiente circostante al robot usando sensori semplici ed economici. In particolar modo, la visione ha dimostrato di poter competere con le classiche e costose soluzioni, basate su sensori laser scanner, al problema dello SLAM (*simultaneous localization and mapping*), in cui un robot ha lo scopo di esplorare un ambiente non noto, creando una mappa di questo ambiente e localizzandosi all'interno di questa mappa allo stesso tempo. Lo scopo di questa Tesi di Laurea Magistrale è quello di analizzare in dettaglio una particolare soluzione al problema dello SLAM che prevede l'utilizzo di una telecamera monoculare come unico sensore, oltre ad implementare questa soluzione in un prototipo funzionante usando note tecnologie e frameworks open source disponibili nel campo della rotorica di servizio.

Acknowledgments

I have to thank first of all my professor and supervisor Basilio Bona and my co-supervision Luca Carlone who have introduced me to the world of robotics and gave me the opportunity to work on this interesting topic, and also my co-supervisor from Politecnico di Milano Matteo Matteucci who gave me several advices about this work. A special thanks goes certainly to Marco Gaspardone and Telecom Italia for supporting a big project of which this Master Thesis is a little part. I'm very grateful to Stefano Rosa for his help on technical issues about robotics and programming and for his very important moral support when I needed it, and to all the other past and present members of Robot Research Group of Politecnico di Torino I have the opportunity to meet in this periods.

Moreover, I have to thank all people who have surrounded me during the last year but also during all the last 10 years of study, I would never have obtained this results without your help and support. I want to thank my friends from high school, especially Uccio, Sergio, Vera, Bianca and Silvia, for having put up with me during five years and later, and also my professors, in particular profs Erminia Santacroce, Fabrizio Perniola and Mario Cassiano (if you will read this thesis, remember: now I hate computing derivatives). Thanks to Verres, three years in that sad place was nothing than the friendships I have gained in it: Simone, Alex, Andrea and Samuele. Thank you all for been a great company for coffee breaks, nighttime "studying", pool party and other "illegal" stuff ...

Thanks to PoliTO and ASP for all the things that these institutions gave me during this years: new friends, new projects and *tanti paperi*. I would like to acknowledge all my friends and colleagues from these schools, and in particular all people involved in PARLOMA and my special *champagne* team. Thanks to Gabriele Tiotto, Marco Indaco and Paolo Prinetto for believing in this project. Thanks to President Chiara and her politics powers, thanks to Alice and her food support during the ASP lessons, thanks to Giorgio for his help with a particular mattress and thanks to Giuseppe, who taught me that `++i` should be better than `i++`, sometimes. And finally, thanks to Davide Tinaso, for having remembered me every day how long ASP weeks are.

A special thanks goes also to all my friends, Maria Letizia, Roberto, Valerio, Federica, Sara, Giorgia and all the other ones from Collegio Einaudi, who have fed and put up with me during the last 2 years, without you Bobbio would never be such funny (maybe).

Finally, *last but not least*, this work would never been possible without the help of several people around me during this last period. Thanks to Silvia, who has been giving me everything since five months ago, and to her relatives and the whole region of Basilicata (they know why). And thanks to my mother Elisa, my father Alfredo, my sister Valentina and my brother Vincenzo, my grandparents (in particular Nonna Vitina), and all my relatives who have surrounded me since I can remember.

Contents

Chapter 1

Introduction

A robot may not injure a human being or, through inaction, allow a human being to come to harm

The First Law of Robotics

In mobile robot applications, a robot moves in and interacts with the environment to perform some specific tasks, e.g., monitoring certain values of a place, localizing and collecting some objects and so on. In all these applications, knowing some information about the environment where the robot is located and also some information about the position of the robot in the environment itself could strongly help perform the task required to the robot, especially if the task itself and/or the environment are very complex.

For example, imagine a robot located in a large server farm committed to monitor the temperature of the servers by means of a thermal sensor, such as thermo-cameras or similar devices, in order to alert an operator if (and where) the temperature is too high, i.e., above a certain threshold, in a specific place. The task performed by the robot, i.e., monitoring the temperature of the environment and sending an alert message if necessary, is not so complex. On the other hand, the dimension of the environment could make impossible to perform this task without some knowledge of the environment itself. Trivially, the robot can not be sure to cover the entire area if it does not know information about shape and extension of that area or it can not localize itself in that area.

Consider now a robot moving in a small or medium environment committed to recognize and collect some objects randomly positioned in that environment: an example could be a robot able to automatically collect tennis balls after a match. While the task can be easily performed reactively, i.e., when the robot locates an object, it reaches and collects it and then looks for the other ones, it is obviously that storing some information (e.g., the positions of all the objects detected by the robot) about the environment while the robot is performing a subtask, can dramatically speed up the entire process. If the robot detects two objects and stores the position of the second one while it is busy in reaching the first one, after that the first one is collected, the robot can directly move towards the second one without wasting time in looking for it.

The two examples show how localization and mapping techniques are strong tools that can improve the performances of navigation algorithms and that are, often, necessary to perform a specific task. Usually, navigation is performed by means of powerful approaches based on laser scanner sensors which are precise but very expensive devices. However, in the last years some new approaches based on vision sensors have been proposed as solutions to navigation problem. Vision

approaches are still slightly robust with respect to classical approaches, however they will become certainly more and more used in the near future.

1.1 Robot Navigation: an overview

For a mobile device, the ability to navigate in its environment is important. Avoiding dangerous situations such as collisions and unsafe conditions (temperature, radiation, exposure to weather, etc.) comes first, but if the robot has a purpose that relates to specific places in the robot environment, it must find those places. Robot navigation means the ability of a robot to determine its own position in the surrounding environment and then to plan a path towards some goal location. In order to navigate in its environment, the robot or any other mobile device requires representation, i.e., a map of the environment, and the ability to interpret that representation.

Aim of this Section is to provide a general overview on the problem of robotics navigation, to introduce some terms which are used in this Master Thesis and to discuss common issues regarding the problem.

Navigation can be defined as the combination of the three fundamental competences: *localization*, *mapping* and *path planning*. Localization is the ability of a robot to establish its own position and orientation within a given map which represents the environment surrounding the robot, Mapping refers to the capacity of a robot to build a representation (map) of the environment in which it moves. Finally, under path planning are labeled all the solutions able to plan the trajectory of the robot within an environment in order to perform a certain task.

According to the last definition, navigation requires a robot to interact with the surrounding environment, or at least to measure it; hence characteristics of the environment have a strong impact on navigations. In particular, environments can be classified as *static environments* or *dynamic environments*. In static environments the only time-variable quantity (state) is the pose of the robot. In other words, only the robot can move in this class of environments, while all the other objects are assumed to stay in the same location forever. On the other hand, in dynamic environments objects (such as people or doors) can move and properties (light conditions) can change. Of course, navigation in dynamic environments is more difficult than navigation in static environments. In the first ones the map can be assumed to be immutable, while in the second class this assumption does not hold true, and the robot needs, in some way, to update the map using on line information. Due to this consideration, localization and mapping problems can not often be split in two different problem, also because usually the robot does not know anything about the environment. In this cases, solutions to mapping and localization have to be performed at the same time. This big problem is known as *Simultaneous Localization and Mapping* (SLAM). In SLAM the robot deals with the necessity of building a map of the environment while simultaneously determining the location of the robot within this map.

Classical SLAM approaches are based on *laser ranging systems* (laser scanners from now), that are accurate active sensors. Its most common form operates on the time of flight principle by sending a laser pulse in a narrow beam towards the object and measuring the time taken by the pulse to be reflected off the target and returned to the sender. Laser scanners allow the robot to have a measurement of the section of the environment in front of it. By merging the various sections obtained while robot is moving, an algorithm can measure the displacements of the robot and, at the same time, build the map of the environment.

Recently, some new approaches able to perform SLAM using vision sensors has been proposed. The problem of performing navigation by means of vision sensors are known as *Visual Navigation*.

The sensors used in visual navigation can be classified according to the type of information they return: a *monocular camera* is a commercial camera, able to return a stream video, i.e., a temporal

succession of frames; a *Depth sensor* is a system able to return a *depth map* of the surrounding environment, i.e., an image of the environment in which to each pixel a measure of the distance (or depth) of that pixel is associated. This type of images are also labeled as RGB-D (where D means depth) or 2.5D images. Examples of depth sensor are stereo systems or the Microsoft Kinect and similar systems.

Classical SLAM approaches based on laser scanners consist in very powerful solutions able to estimate very efficiency the map and the robot location. However they presents two big limits: the cost of the laser sensor and incapacity to work in open spaces. On the other hand, visual navigation approaches are still in a researching phase but are not affected by the first issue. Moreover, approaches based on cameras are able to work both in indoor and outdoor environments.

Another issue regarding navigation in general and SLAM in particular is the difficulty to obtain good performance in *isotropic environments*, i.e., environments which present high grade of symmetry. In this case, robots are not able to merge information without ambiguity, so that similar parts of the environment may appear to the robot as same the place. This issue is very common in visual approaches based on features, where the robot performs navigation by tracking a limited set of distinguishable key points in the environment. Of course, if two key points are very similar to each other, the robot is not able to distinguish them and the algorithm may fail.

1.2 Visual Navigation: related works

Visual navigation is a broad term grouping techniques and algorithms based on visual sensors to provide the following services:

- localization in a known environment,
- map building and localization in an unknown environment (visual SLAM),
- path planning.

In this Section only algorithms related to localization and visual SLAM problems, which are the main topic of this Master Thesis, are considered.

Visual navigation approaches are all based on a vision system, that could be a single camera or a system able to produce a depth-map of the scene, such as a stereo system or a depth camera system. The majority these approaches is based on the concept of features matching, that is a technique able to match a set of key points in different images of the same scene taken from different points of view. By measuring the displacements of these key points it is possible to reconstruct the motion of the camera and its positions in the scene. However, features matching is not a simple and fast job and often mismatches occur. For this reason, at the current state of the art, visual navigation algorithms fail in not ideal conditions, such as dynamic or big environments, uniform environments, in which the robot can not found a sufficient number of interest points, and are not robust enough in presence of big displacement of the robot and occlusions.

Approaches performing visual navigation proposed in literatures can be also classified according to the quantity and type of the information stored in the map: *dense*, *sparse* and *semantic* solutions.

Dense visual navigation algorithms use a dense, i.e., continuous, map of the environment. They are robust solutions but require a lot of computational power due to the huge amount of information to be processed. For this reason, they work in real time only if optimized to GPU processors.

Sparse map algorithms are based on a map in which only a small number of points in the environment are stored. This class of algorithms requires less computational efforts because they locate in memory only the most significant key points representing the map. For this reason, they are good candidates for a real time visual SLAM implementation. Despite that, they are less robust

due to the fact that a single mismatch in features matching could be disastrous. Moreover, this kind of algorithms is candidate to build very complex map made by semantic objects instead of simple key points.

The idea behind semantic approaches is very similar to the one behind sparse solutions, but in this case high level information are associated to the reconstructed points in the map, in order to generate more relevant results. In this way, a single point in the map does not simply represent a recognizable feature in the image, but can be associated to a real object such as the center of a table, a door, etc.

Aim of this Section is to present an overview on the most important algorithms and solutions solving visual navigation and visual SLAM problems, enhancing their benefits and limits and explaining the choice of implementing in particular the Mono-SLAM algorithm instead of other solutions.

1.2.1 Dense Approaches

Dense visual navigation methods are approaches able to reconstruct the environment in a dense points cloud in which all the pixels in the various frames are reconstructed. Since all the information stored in the images are used, these approaches provide a good grade of robustness but require a lot of computational power, so that usually their are not able to satisfy real time constraints. For this reason, the majority of these solutions is used for off line processing of the stream video or is implemented using cloud computing solutions, where the main routines are not executed on the robot but remotely on very powerful machines.

An interesting approach belonging to this class is *Dense Tracking and Mapping* (DTAM) method[2], that is a real time camera tracking and dense scene reconstruction algorithm based on a monocular camera system. To track camera in real time, dense whole image alignment against the scene model is performed, on the other side, the model is updated and expanded using the images coming from tracked camera. Hence, the system is fully self-supporting after an initialization phase. To meet real time constraints, the algorithm is fully implemented in GPU, moreover, the scene updating is not performed every frame but only when the camera is moving very slowly.

In dense approaches based on single camera, as DTAM, reconstruction from a 2D image requires a lot of additional computational efforts, hence the majority of these solutions is based on sources of information providing directly depth map images such as the Microsoft Kinect. As in the Kinect Fusion methods.

Kinect Fusion is a dense method of indoor environment mapping and localization proposed by Izadi et al. in [3, 4]. It allows to build up accurate map of indoor scenes in real-time, using only a moving Kinect depth camera and common graphics hardware. Using only depth data, the system continuously tracks the 6 degrees of freedom (DOF) pose of the sensor using all of the live data available from the Kinect sensor, and integrates depth measurements into a global dense 3D model. Differently from the standard methods based on features extraction and matching, the Kinect Fusion approach works directly with surfaces, by means of the *iterative closest point* (ICP) [5] algorithm for accurate and computationally efficient registration of 3D shapes, curves and surfaces. Moreover, since the approach uses only depth information from the sensors, it is able to work also in dark environments, in contrast with lots of camera based systems whose performances are strongly related to light condition. To reach real time constraints, the algorithm is optimized to be executed by a GPU and implemented in CUDA language. Using a GPU implementation is the only solution to elaborate such amount of data with real time constraints, but it limits the machines on which the algorithm can be executed.

3D information of the environment are stored in a predefined volume which limits the reconstruction region of the algorithm. This volume is then subdivided uniformly into a 3D grid of voxels. The predefined volume in which the algorithm works limits the algorithm itself to reconstruct only environments of small dimensions, such as an indoor room.

In [6] Whelan et al. present an interesting work, called *Kintinuous*, able to extend the Kinect Fusion algorithm to work in large scale environments. The algorithm simply allows the predefined volume region to vary dynamically according to camera motion and stores the parts of the points cloud that leave this region in a triangular mesh representation of the environment. A very similar approach was developed at the same time in [7].

1.2.2 Sparse Approaches

All approaches which rely on a sparse map composed by a small number of selected interested point are labelled under the name of sparse approaches. This class of methods is candidate for real time applications since they require very limited computational power. On the other hand, these methods are much less robust than the dense counterpart and require sophisticated solutions to improve their robustness. Moreover, the required computational resources available in embedded systems are still too few to implement this algorithm without using particular solutions such as dedicated hardware and optimized code. Note that the sparse approach methods are all based on *features matching* algorithms, that are methods which aim at finding correspondences of the same point projected on different images taken from different points of view. Features matching approaches will be discussed later in this document.

Thinking about strong real time requirements such as augmented reality, the *Parallel Tracking and Mapping* (PTAM) [8] algorithm, based on a monocular camera system, was proposed. It extends the DTAM algorithm previously described. The main idea consists in splitting tracking and mapping in two different phases. Tracking is performed each time a new frame is available, by comparing the new frame itself with the map using feature descriptors and matching techniques; to speed the algorithm up, the slightly robust *FAST* features are used. On the other side, the map updating phase is performed only when the current camera position estimation is precise enough; in this case, the current frame is used to update the map. The PTAM algorithm requires a initialization phase in which the same features are seen from different points of view to initialize the map. The most important limit of the algorithm is the impossibility to handle occlusions.

A powerful solution belonging to this class of approaches is the *Mono-SLAM* algorithm proposed in [1, 9]. In this approach, the map and the camera pose are stored in a stochastic multidimensional variable and the system evolution is estimated using the *Extended Kalman Filter* (EKF) algorithm. By using a stochastic approach, the algorithm is able to strongly speed-up the features matching phase, since the location of the interesting point in the new frame is searched only in the most probable region in which it should lie. This approach is known as *active search* paradigm. In addition, the algorithm does not need an initialization phase as PTAM and is enough robust to handle occlusions. The most important limitation of that algorithm is the fact that, when the map becomes too big, the EKF processing phase becomes too low to satisfy real time constraints. However, several solutions have been proposed to solve this problem.

1.2.3 Semantic Approaches

Semantic approaches are an extension of sparse approaches in which the map is built by using objects which store semantic high level information of the environment, instead of simper interest points. These approaches bring the two main benefits of the two classes of methods previously discussed, inasmuch they use a sparse map, that requires limited computational power, but are able

to maintain a good grade of robustness since they use high level features and more sophisticated algorithms, such as objects recognition. However, for their complexity, these solutions are still at a theoretical phase of development, in fact currently no algorithm able to automatically extract high level information from an unknown environment exists, since all the semantic approaches are based on objects recognition algorithms, which require an initial training phase during which they "learn" to recognize predefined objects. For this reason, semantic approaches are much less powerful with respect to the other two classes since they need to known the objects which lie in the environment. For instance, a navigation algorithm developed to recognize tables and to perform SLAM by using tables in the environment, will fail in an environment with no tables inside.

A very first solution which extends a sparse approach (Mono-SLAM) to a "semantic" approach is the work proposed in [10], which performs a monocular SLAM algorithm using string lines as features, in addition to points. The algorithm is able to recognize, match and reconstruct both point and lines and, according to the authors, the system is able to obtain good results using either line features alone, or lines and points combined.

A second extension of the Mono-SLAM algorithm to a semantic approach is proposed in [11, 12], where Castle et al. combine the Mono-SLAM approach with an algorithm able to recognize planar objects. For each recognized object, three points of its contour are added to the state of the EKF filter, and the covariance matrix of the three points is set in order to respect geometric constraints. Moreover, if the dimensions of the objects are available, the scale of the scene (which in all monocular approaches are unknown) can be easily deduced.

Recently, a very interesting work has been proposed by Salas-Moreno et al. in [13], consisting in a complete semantic method based on a Kinect system and objects recognition approaches. The *SLAM++* built maps directly at the "object oriented" level by jumping over low level geometry processing. The algorithm needs to have a prior knowledge of the objects likely to be repetitively present in the scene, and performs real-time 3D recognition and the creation of a simple pose graph map of relative object locations. The models of the objects can be easily built up by using the Kinect Fusion algorithm previously described and matches are found according to the ICP method as in Kinect Fusion. The map is continuously optimized as new measurements arrive, that provide precise prediction of the next camera measurement. These predictions are used for robust camera tracking and the generation of active search regions for further objects detections. To meet real time constraints, the algorithm is implemented in GPU as all the real time algorithms based on the Kinect or similar depth cameras.

1.2.4 Some Remarks about Visual Navigation

In this Section an overview of the most important approaches regarding visual navigation is presented. As shown previously, in literature several approaches and solutions to perform visual navigation have been proposed, each of whom presents limits and advantages. For example, dense approaches are very good on providing meaningful maps and have a good grade of robustness, but their implementation requires a very huge amount of computational resources. On the other side, sparse approaches require less computational power but are much less robust than the others. Semantic approaches try to merge the benefit of the other two classes using high level information, but they are still at an embryonal phase of development and currently they do not provide very important results.

Since the goal of this Master Thesis is the development of a real time visual navigation system based on a monocular camera and able to be executed by a system with few computational resources available, the natural choice is to implement the Mono-SLAM algorithm which seems to be the sparse solution providing better results. Moreover, the Mono-SLAM approach has been

already extended to the semantic navigation field, so that its implementation could be extended in future works and be adapted to most powerful approaches using semantic techniques. Finally, the knowledge required by Mono-SLAM to be implemented could actually bring new skills to the author of this Master Thesis about computer vision, programming and robotics navigation in general.

1.3 This Master Thesis

This work has been carried out in the RRG (Robotics Research Group) of Politecnico di Torino under the supervision of Basilio Bona and Luca Carlone, in collaboration with Telecom Italia, under the supervision of Gaspardone Marco and Roberto Antonini, and with the technical support of Matteo Matteucci from Politecnico di Milano.

Aim of this Master Thesis is to exploit and implement a particular solution to the problem of visual navigation using only a monocular vision system as source of information. This chosen approach, known as Mono-SLAM algorithm, is one of the first developed solutions able to reconstruct the motion of the camera in 3D space in real time and nowadays is still very used in several applications. During this period of Master Thesis, the author has deeply explored from a theoretic point of view the original approach and its evolutions, then he has implemented the algorithm in C/C++ language using the *Robotics Operative System* (ROS) framework, which is a well known and open source framework for mobile robotics applications.

1.3.1 Outline

Chapter 1 The introduction to the Master Thesis, providing a brief overview on the problem of robotics navigation, including an excursion into classical approaches and visual solutions; an overview on the state of the art about visual navigation, which is the topic this Thesis is focus on; together with the outline of the Master Thesis itself.

Chapter 2 A deep description of the basic theoretical camera model commonly used in computer vision and its extensions to better fit real world cameras, together practical applications of camera calibration techniques, i.e., approaches to find the correct values of the model parameters which fit a real camera. One of the models discussed in this Chapter are used in the implementation of the algorithm here proposed.

Chapter 3 A description of some computer vision solutions and approaches used in the implementation of the algorithm presented in this Master Thesis. Including an excursus on the most used approaches to perform motion tracking and their benefits and limits.

Chapter 4 An intuitive and simple introduction to the reconstruction problem, i.e., the problem of reconstruct 3D information of the environment starting from 2D images. This Chapter introduces the main issues which Mono-SLAM and similar approaches have to deal with.

Chapter 5 A very deep analysis of the Mono-SLAM algorithm, including the the execution of all the calculations needed to implement the algorithm and a discussion of solutions used to improve real time performances in this approach.

Chapter 6 A description of the proposed solution, including a novel solution proposed in this Master Thesis and other practical techniques to improve the original algorithm, the tools used to implement the proposed solution, together with a presentation and discussion of the obtained results.

Chapter 7 The conclusion of the Thesis, presenting open topics, future works and final remarks.

Chapter 2

Camera Models and Calibration

Where I think the most work needs to be done is behind the camera, not in front of it

Denzel Washington

Camera plays a very important role in monocular navigation. For this reason, to build a system effectively working, it is of relevance to know how a camera system works.

All the models used to describe how a camera works are composed by a linear part and a nonlinear correction:

- the *linear camera model* describes how the 3D world is projected on the 2D image,
- the *non linear distortion correction* aims at modeling and correcting the non linear geometrical distortion introduced by the lens of the camera glasses.

While the linear or pinhole camera model is a standard model, in literature there exist several distortion correction models which should be chosen accordingly to the application and the types of lenses used in the camera.

This Chapter aims at describing in details the linear camera model and the most used distortion models and explain the techniques of calibration to generate the model of a specific camera. The Chapter is organized as follows: Section 2.1 presents all the notations and conventions used in the above discussion; Section 2.2 illustrates a very simple model of the camera known as the *pinhole camera model*, which is the starting point to build the two models presented later; in Sections 2.3 the linear model of the camera is discussed, enhancing in particular its limitation when facing with real camera systems; after that, Section 2.4 provides an overview on the nonlinear effects affecting real camera systems, introduces the main nonlinear components, labeled as the *geometrical distortions*, and presents, from a theoretical point of view, some useful models able to handle distortions; moreover, Section 2.5 presents two interesting complete camera models where both linear components and nonlinear distortions are considered. The models are analyzed from a theoretical point of view and presented in order to be easily adapted to the Mono-SLAM algorithm; finally, Section 2.6 presents two main camera calibration procedures which aim at estimating the parameters of the model of a given camera, applies that procedures to a real camera and presents and discusses the results obtained.

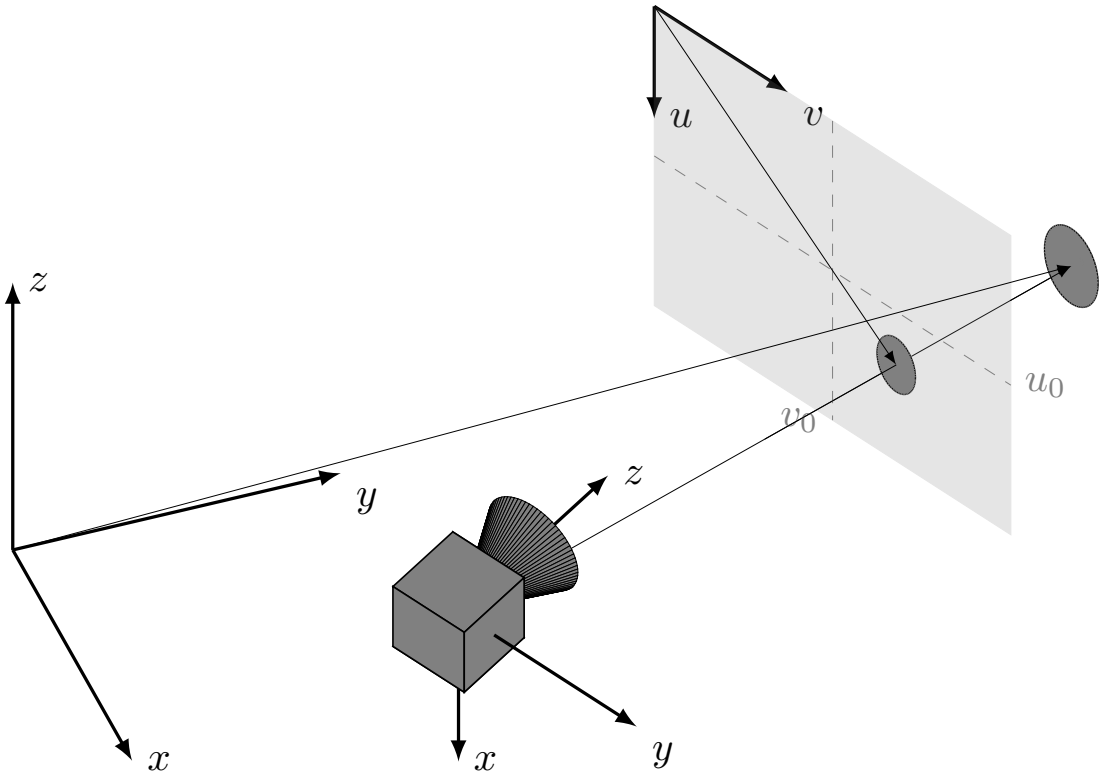


Figure 2.1: Conventions used to define world, camera and image reference frame. The world reference frame $\mathcal{W}$ is located somewhere in the environment. The camera reference frame $\mathcal{C}$ is centered in the optical center of the camera and has the axis as shown in figure. The image reference frame $\mathcal{I}$ is centered in the top left corner of the image.

2.1 Conventions and Notations

In this Section, conventions and notations used in the remain part of this Chapter and also in other Chapters of this Master Thesis are presented. Note that the following considerations refer to Figure 2.1.

- A vector in a 3D space is indicated with bold notation ($\boldsymbol{v}$) while a vector in a 2D space is indicate with underline notation ($\underline{v}$).
- The world reference frame $\mathcal{W}$ is a 3D reference frame positioned somewhere in the scene. It is fixed, i.e., it can not be moved, and its position and orientation are usually chosen when the first frame is captured. To enhance the fact that $\boldsymbol{v}$ is expressed in $\mathcal{W}$-coordinates, the notation $\boldsymbol{v}^{\mathcal{W}}$ is used.
- The camera reference frame $\mathcal{C}$ is a 3D reference frame attached to the moving camera using the following convention:

  - the origin O of $\mathcal{C}$ is located in the optical center of the camera,
  - the z-axis coincides with the focal length of the camera, oriented toward the scene,
  - the x-axis pointing downwards,
  - the y-axis pointing to the right, according to the right hand rule.

  To enhance the fact that $\boldsymbol{v}$ is expressed in $\mathcal{C}$-coordinates, notation $\boldsymbol{v}^{\mathcal{C}}$ is used.

- The image reference frame $\mathcal{I}$ is a 2D reference frame representing the coordinates in the image plane. The reference frame is attached to the image plane using the following convention:
  - the origin O of $\mathcal{I}$ is positioned in the upper left corner of the image,
  - the u-axis is horizontal pointing right, parallel to the x-axis of $\mathcal{C}$,
  - the v-axis is vertical pointing down, parallel to the y-axis of $\mathcal{C}$,
  - the point $\underline{p}_0 = \begin{pmatrix} u_0 & v_0 \end{pmatrix}^T$ indicates the principal point of the image, that is the point where the optical axis of the camera crosses the image plane. Note that the principal point does not necessarily coincide with the center of the camera, since to obtain this result, the alignment of the lenses should be precise at micrometers.

  When a digital camera is used, the image is discretized and the $\mathcal{I}$-coordinates are scaled according to the size of the single pixel. Since in all the application here presented, the image is normally measured in pixels. When it is necessary distinguish if the image is represented in pixels or in real measurement units, the following notation is used: with $\mathcal{I}$ the pixel-sized reference frame is indicated and with $\mathcal{I}^N$ the normalized image reference frame, i.e., the image reference frame where the coordinates are expressed in real size (e.g. μm), which is centered in the principal point of the camera. To convert $\mathcal{I}$ to $\mathcal{I}^N$, d_u and d_v are defined as the physical dimensions of each pixel along the u and v axes respectively. The correspondent point in normalized coordinates of a point $\underline{p}$ is given by

$$\underline{p}^N = \mathbf{F} \cdot \left(\underline{p} - \underline{p}_0\right) = \begin{pmatrix} d_u(u - u_0) \\ d_u(v^N - v_0^N) \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} d_u & 0 \\ 0 & d_v \end{pmatrix}. \tag{2.1}$$

- Due to the projective nature of the camera, it is very common to use the projective geometry and the homogeneous coordinates. Subsequently, the *tilde* hat notation ($\tilde{\boldsymbol{v}}$) indicates a vector expressed in homogeneous coordinate. For more information about projective geometry and the homogeneous coordinates please refer to [14]. Please note that when an homogeneous transformation (expressed by a matrix $\mathbf{A}$) is applied to a normal vector, the notation $\boldsymbol{b} = \mathbf{A}\left(\boldsymbol{a}\right)$ is used, indicating that $\boldsymbol{b}$ is the vector associated to the homogeneous vector $\tilde{\boldsymbol{b}} = \mathbf{A}\tilde{\boldsymbol{a}}$.

- The camera lens system introduces a geometrical distortion which the model must handle, to underline the fact that a point is expressed as a distorted pixel, the subscript $_d$ is used. For instance, the $\underline{p}_d$ is the distorted point corresponding to $\underline{p}$ in the undistorted image, similarly, $\underline{p}_d^N$ refers to a point in a distorted normalized image.

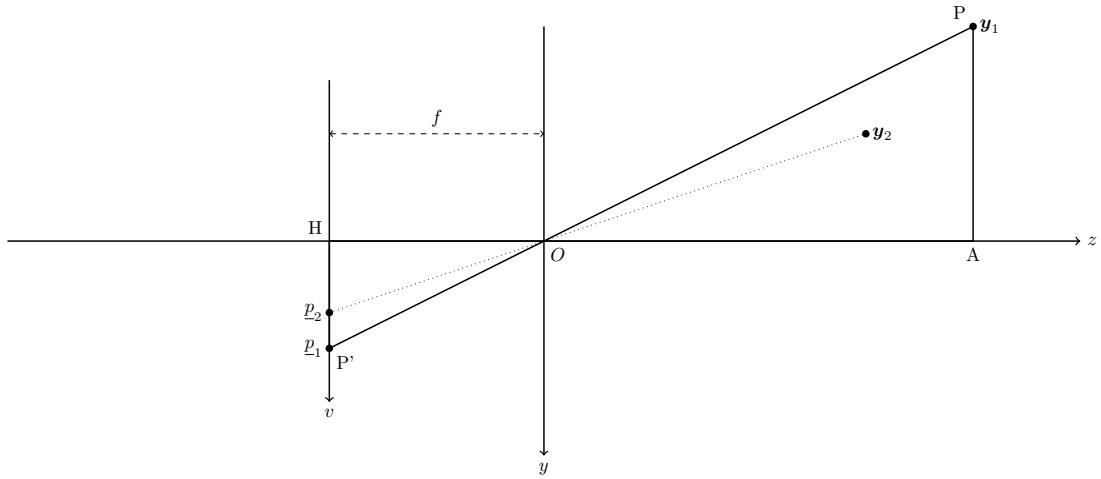


Figure 2.2: Projection of a point in 3D space onto the image plane.

2.2 Pinhole Camera Model

A camera is a system able to project point in a 3D space, i.e., the scene, in a 2D plane according to a specific relation. For an ideal pinhole camera, that is a camera having a single point (the optical center) as aperture and no lenses. This relation is simply described by a parameter f known as *focus length* of the camera, that indicates the distance between the image plane and the origin of the camera reference frame $\mathcal{C}$, i.e., the optical center of the camera.

For simplicity, in the following discussion a 2D view of the camera system is considered, so that the u-axis of $\mathcal{I}$ and the x-axis of $\mathcal{C}$ are hidden. Obviously, all the considerations and results hold true also for the hidden dimension.

As seen in fig. 2.2, the points $\boldsymbol{\psi}_i = \begin{pmatrix} x & y & z \end{pmatrix} \in \mathcal{C}$ are projected on the image plane in the points $\underline{p}_i$. The two triangles $\widehat{OAP}$ and $\widehat{OHP'}$ are similar, so the following relation holds:

$$\frac{OA}{f} = \frac{AP}{HP'} \rightarrow HP' = f\frac{AO}{OA}.$$

By considering that the $\underline{p}$ coordinate change its sign, the coordinate of $\underline{p}$ in $\mathcal{I}^N$ are

$$v^N = -f\frac{y}{z}.$$

The same considerations hold for the u-axis, so that

$$\begin{cases} u^N = -f\frac{x}{z} \\ v^N = -f\frac{y}{z}. \end{cases}$$

The last equation can also be written in homogeneous coordinates as follows

$$\tilde{\underline{p}}^N = \begin{pmatrix} u^N \\ v^N \\ 1 \end{pmatrix} = \begin{pmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \tag{2.2}$$

The last equation represents the transformation function from a point in the scene to a point on the image plane when a pinhole camera is used to capture the picture.

2.3 Linear Camera Model

The linear camera model aims at modelling how a point in the world reference frame $\mathcal{W}$ is projected onto the image plane $\mathcal{I}$. The model provides two matrices called *extrinsic* and *intrinsic* matrices which model respectively how a point in $\mathcal{W}$ coordinates is transformed in $\mathcal{C}$ coordinate and how a point in $\mathcal{C}$ is projected in $\mathcal{I}$ coordinates.

Extrinsic Matrix

The extrinsic matrix of a camera represents roto-transaction from $\mathcal{W}$ to $\mathcal{C}$, as follows

$$\tilde{\boldsymbol{p}}^{\mathcal{C}} = \begin{pmatrix} \mathbf{R} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \tilde{\boldsymbol{p}}^{\mathcal{W}}, \tag{2.3}$$

where $\mathbf{R}$ and $\mathbf{t}$ are respectively the rotation matrix and the translation vector of the roto-translation main transformation. The last relation can also be written in the following form

$$\boldsymbol{p}^{\mathcal{C}} = \begin{pmatrix} \mathbf{R} & \boldsymbol{t} \end{pmatrix} \tilde{\boldsymbol{p}}^{\mathcal{W}}, \tag{2.4}$$

where the point in $\mathcal{C}$ in not expressed in homogeneous coordinate.

Please note that, as long as the camera is assumed as freely moving in the space, it is impossible to calibrate the extrinsic matrix (because it varies upon time). The required information, that cannot be deduced elsewhere, must be computed by the algorithm runtime.

Intrinsic Matrix

The intrinsic matrix of a camera represents the transformation from a point in $\mathcal{C}$ to its projection on the image plane $\mathcal{I}$, considering ideal the lens system of the camera. To build up the matrix, parameters introduced by the discretization are added to the relation described in equation (2.2). Note that the this additional parameters depend both on the conversion between normalized coordinates in pixel coordinates and on errors introduced by the non perfect way used to build the sensors itself.

To convert relation in equation (2.2) in pixel coordinates, the equation (2.1) is used, obtaining the relation

$$\underline{\tilde{p}} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} -f_u & 0 & u_0 \\ 0 & -f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{2.5}$$

where $f_u = f/d_u$ and $f_v = f/d_v$ represent the focal distance (a unique value in mm) expressed in units of horizontal and vertical pixels. Both components are usually very similar but they are not coincident since d_u and d_v are not guarantee to be equal, since the pixel could be rectangular. Therefore, the camera model naturally handles non-square pixels. The ratio $f_v/f_u = d_u/d_v$ is usually very near to 1.

Often, the physical size of the camera pixels is not available, for this reason the majority of models use only the parameters f_u and f_v but can not deduce the real size of the focal length f.

Note that the algorithm works well both in presence or absence of the *minus* sign affecting the focal length parameters. Moreover, the majority of camera system automatically corrects the

inversion of the image in the image place, for this reason usually camera models omit the minus sign affecting the focal lengths parameters. The last equation is so converted in

$$\tilde{\underline{p}} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{2.6}$$

Finally, a parameter is added to model the fact that the u and v axes are not exactly perpendicular each other. Obtaining the final form of the camera intrinsic matrix

$$\mathbf{K} = \begin{pmatrix} f_u & \gamma & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.7}$$

The parameters γ is known as skew coefficient of the camera and depends on f_u and the angle between the u and v sensor axes. However, in all the recent camera sensors, the relation $\gamma = 0$ holds practically true, so all the models here presented will use the following version of the camera intrinsic matrix

$$\mathbf{K} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.8}$$

A last consideration is necessary when considering the relation between the $\mathcal{I}^N$-coordinates and the matrix $\mathbf{K}$, from equations (2.1) and (2.2) follows directly that the intrinsic matrix can be decomposed as follows

$$\mathbf{K} = \begin{pmatrix} \frac{1}{d_u} & 0 & u_0 \\ 0 & \frac{1}{d_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{K}_x \mathbf{K}^N. \tag{2.9}$$

This decomposition is very useful when the camera distortion model is introduced.

Camera Principal Matrix

The two matrices previously presented can be encapsulated in a single matrix, known as *principal matrix* of the camera, which represents the transformation between a point expressed in the world reference frame $\mathcal{W}$ in the image reference frame $\mathcal{I}$ as follows

$$\tilde{\underline{p}} = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix} \tilde{\boldsymbol{p}}^{\mathcal{W}}. \tag{2.10}$$

The reconstruction problem

Consider equation (2.10) and let the non linear aberration effects be negligible: the reconstruction problem aims at computing $\boldsymbol{p}^{\mathcal{W}}$ – or equivalently $\boldsymbol{p}^{\mathcal{C}}$ – from the point $\underline{p}$ on the image reference frame. In other words, it aims to invert equation 2.10.

From a simple image this problem has not solution, simply because the matrix $\mathbf{K}(\mathbf{Rt})$ is not a square matrix. It has also rank $\rho(\mathbf{K}(\mathbf{R}|\mathbf{t})) \leq 3$, so it is impossible to estimate $\tilde{\mathbf{P}}_{\mathbf{W}} \in \mathbb{R}^4$ with Ordinary Least Squares or similar approaches. From an intuitive point of view, computing the position of a single point in the 3D – we need 3 piece of information – space from only one photo because we have only two piece of information is not possible. It is only possible to compute the epipolar line of the point, that is the imaginary line on which the point in the scene should be located.

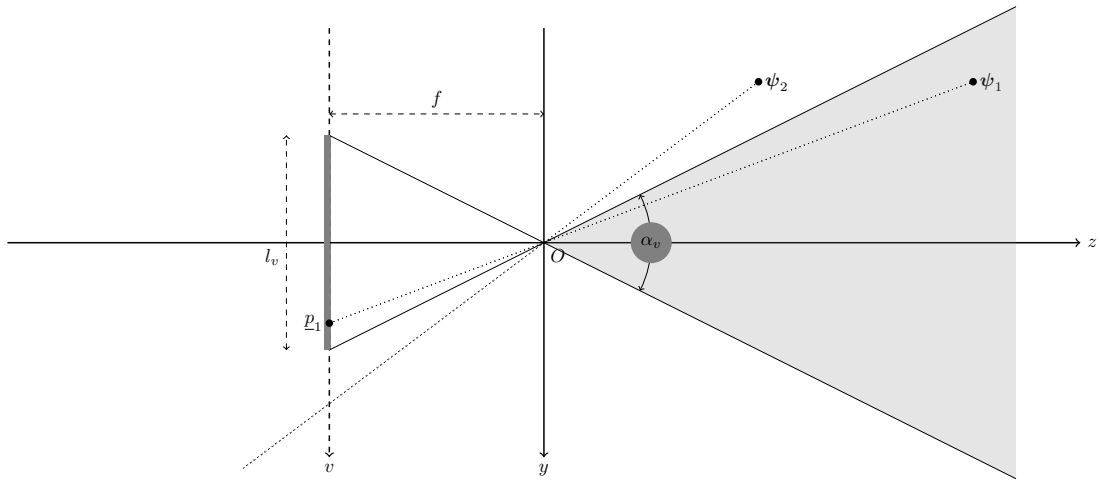


Figure 2.3: The field of vier of the camera is limited by the physical size of the image sensor, as shown in the Figure. The point $\boldsymbol{\psi}_1$ is correctly projected and captured on image while the projection of the point $\boldsymbol{\psi}_2$ gets out of the image sensor and is not captured in the image.

EKF, Photogrammetry and similar approaches aim to solve this problem merging information from two or more frames. All these methods have the problem of matching corresponding points from different images. The reconstruction problem is deepened in Chapter 4.

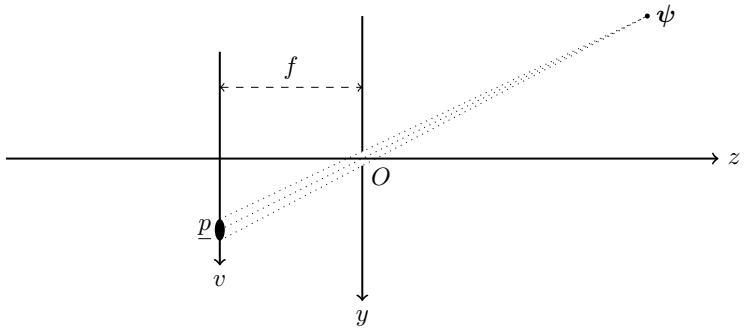
2.3.1 Adaption of the Camera Model

The difference from a pinhole camera to a real digital camera is the introduction of a digital sensor and the use of an optical system. A lens optical system is necessary, since using a pinhole camera in realty is not beneficial. If the hole is too big, the image will appeared blurred, while shrinking the hole beyond a certain point makes the captured photo to be very dark, since the light received by the sensors is not enough to be measured by the digital circuits. Moreover a small hole might cause diffraction and produce an image blurred too. All these phenomena are represented in Figure 2.4.

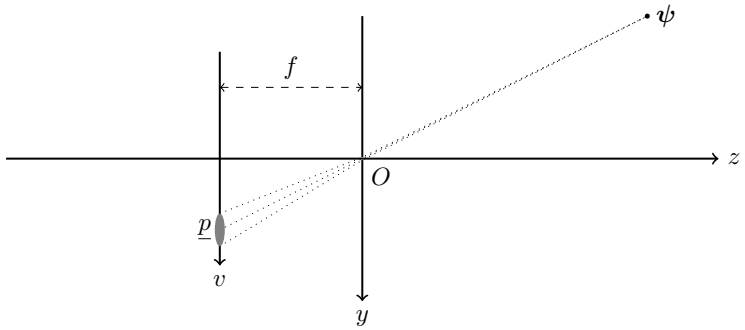
For this reason, real cameras work using a system of lenses used to capture and converge the light on the image plane. However, the optical system introduces some departure in performance, known as aberration, which will be discussed later in this Chapter. Moreover, in the real cameras, the world is not projected entirely on the image plane as supposed by the linear model, but only the half-space in front of the camera are projected on the image plane (in a pinhole camera) and only the light that converges on the sensitive material, i.e., the image sensor, are captured. The portion of the real world that the camera is able to capture is known as *field of view* of the camera, and its dimension are related to the dimension of the image sensor, as shown in Figure 2.3. In particular, the angles α_u and α_v limiting the field of view are given by

$$\alpha_u = 2 \arctan \frac{l_u}{2}, \qquad \alpha_v = 2 \arctan \frac{l_v}{2},$$

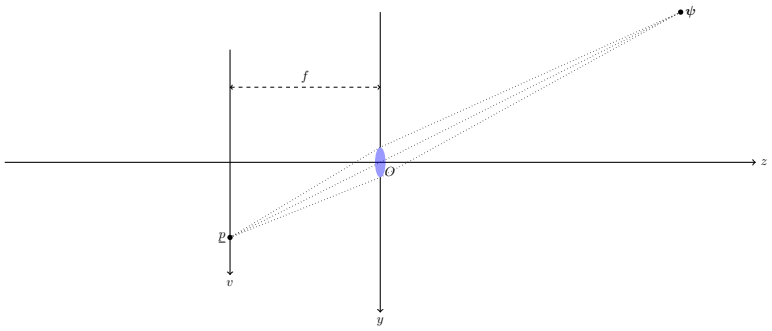
where l_u and l_v represents the physical size of the sensor.



(a) Big hole



(b) Small hole



(c) Lens system

Figure 2.4: Problems affecting pinhole cameras. If the pinhole of the camera is too big (a), the image will appear blurred, if it is too small (b), not enough light will reach the sensor diffraction might produce blurred image. A lens system (c) solves both the problems.

2.4 Non-linear components

Lens systems introduce some non-linear effects when projecting the scene on the image, labeled as aberrations. Aberrations are modeled by the liner model previously presented. In an imaging system, aberrations occur when light from one point of an object does not converge into a single point after transmission through the system. In the following subsection, a brief overview on aberration is provided.

2.4.1 Aberration: an overview

Aberrations fall into two classes: chromatic and monochromatic. Chromatic aberrations occurs because lenses have a different refractive index for different wavelengths of light (the dispersion of the lens). The refractive index decreases with increasing wavelength. They do not appear when monochromatic light is used. Chromatic aberrations cause different wavelengths of light to have differing focal lengths.

Monochromatic aberrations are caused by the geometry of the lens and occur both when light is reflected and when it is refracted. They appear even when using monochromatic light, hence the name. The monochromatic aberrations fall into several classes, that are presented below.

- **Defocus** is the aberration in which an image is simply out of focus. This aberration is familiar to anyone who has used a camera, videocamera, microscope, telescope, or binoculars. Optically, defocus refers to a translation along the optical axis away from the plane or surface of best focus. In general, defocus reduces the sharpness and contrast of the image. What should be sharp, high-contrast edges in a scene become gradual transitions. Fine detail in the scene is blurred or even becomes invisible. Nearly all image-forming optical devices incorporate some form of focus adjustment to minimize defocus and maximize image quality. Due to defocus aberration, only objects within a limited range of distances from the camera will be reproduced clearly.
- **Spherical aberration** occurs due to the increased refraction of light rays when they strike a lens or a reflection of light rays when they strike a mirror near its edge, in comparison with those that strike nearer the center. It signifies a deviation of the device from the norm, i.e., it results in an imperfection of the produced image.
- **Coma** appears when off-axis point sources are presented. This point appears distorted, such as a star appearing to have a tail (coma) like a comet.
- **Astigmatism** occurs when rays that propagate in two perpendicular planes have different foci. If an optical system with astigmatism is used to form an image of a cross, the vertical and horizontal lines will be in sharp focus at two different distances.
- **Geometrical distortion** is a deviation from rectilinear projection, it causes that straight lines are not projected in straight lines on the image plane.

Note that the majority of the aberrations here presented are actually impossible to correct, this because they cause an image, or better some parts of an image, to be not sharp. Theoretically, aberrations such as coma, chromatic aberration or astigmatism can be corrected if the point spread function (SPF) of the lens system is available. However, this specific information are very difficult to obtain, also because often the PSF depends on the distance of the projected point from the image plane, which is completely unknown and can not be measurable from the image only. However, their effects are often very small and can be negligible. To deepen the art of restoring images please refers to [15], while if the reader is interested in theoretics on optical aberration [16–18] should provide much information.

The only aberrations which should be correct to make all the photogrammetric algorithm work correctly is the geometrical distortion. If geometrical distortion is not taken into account by the approach, it cause errors in measurements of key points, introducing big errors in computing their 3D position. Luckily, this kind of aberrations do not cause the image to be blurred. For this reason it is simple to model and correct.

2.4.2 Geometrical Distortion

The origin of geometrical distortion lies in a difference between the transverse magnification of a lens and the off-axis image distance. When this distance deviates from that predicted by paraxial theory for constant transverse magnification, distortion can arise due to differences in focal lengths and magnifications through various parts of the lens. In the absence of other aberrations, geometric distortion is manifested by a mis-shaped image.

Fortunately, the geometrical distortion can be measured and correct before processing the image, but cannot be included in the camera model due to its unlinearirty. The distortion effects are modeled in the so called distortion coefficients: k_i and p_i, which are known respectively radial and tangential distortion coefficients.

Distortion is labeled into two different main classes: *radial distortion* and *tangential distortion*. Radial distortion is due to the geometrical shapes of lenses, since it is much easier, from a manufactory point of views, to make a "spherical" lens then to produce a more mathematically ideal "parabolic" lens. According to its name, it causes a displacement of a point from the ideal position along the radial direction, considering the principal point of the image as the center, the effect of radial distortion in represented in Figure 2.5. The second main contribution of distortion, i.e., the tangential distortion, is instead caused by the not perfect alignment of lens and image sensors. It happens especially in cheap systems where sensor is fixed on the camera body simply using glue, as shown in Figure 2.6. Please note that several others geometrical distortion class can be defined, however their effects is negligible with respect to the main classes here presented. For this reason, they usually are not taken into account by models.

The distortion models aim at correcting or predicting the distorted pixels presented in each photos taken by a normal camera. They provide the implementation of a function, described by the distortion coefficients, which is able to compute the undistorted pixel coordinate corresponding to a given distorted pixel (distortion correction) or the other way around (distortion prediction). Note that, due to the complexity of the model, often only one of the two functions (distortion correction and prediction) are provided in a closed form, while the other one can be computed only thanks to recursive algorithm which numerically invert the first function. A good overview of the various camera distortion models is presented in [19], while a standard reference presenting a complete model of distortion is [20], where the so called *plumb bob* model is proposed.Since all the models are defined on the normalized image reference frame, the superscript "N" will be omitted to simplify the notation. According to the plumb bob model, the undistorted coordinates of a point $\underline{p}_d = \begin{pmatrix} u & v \end{pmatrix}$ in a normalized distorted image are given by

$$\underline{p} = \underline{p}_d l(r_d) + \delta_t, \tag{2.11}$$

where

$$\underline{p} l(r_d) = 1 + k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6 + \dots, \qquad r_d = \sqrt{u^2 + v^2} = \|\underline{p}_d\|_2,$$

and

$$\delta_t = \begin{pmatrix} \left(p_1 \left(r_d^2 + 2u^2\right) + 2p_2 uv\right) \left(1 + p_3 r_d^2 + \dots\right) \\ \left(p_2 \left(r_d^2 + 2v^2\right) + 2p_1 vu\right) \left(1 + p_3 r_d^2 + \dots\right) \end{pmatrix},$$

however the model is usually approximated considering only the first 2 (or sometimes 3) components of the radial distortion and, if present, the first 2 components of the tangential distortion.

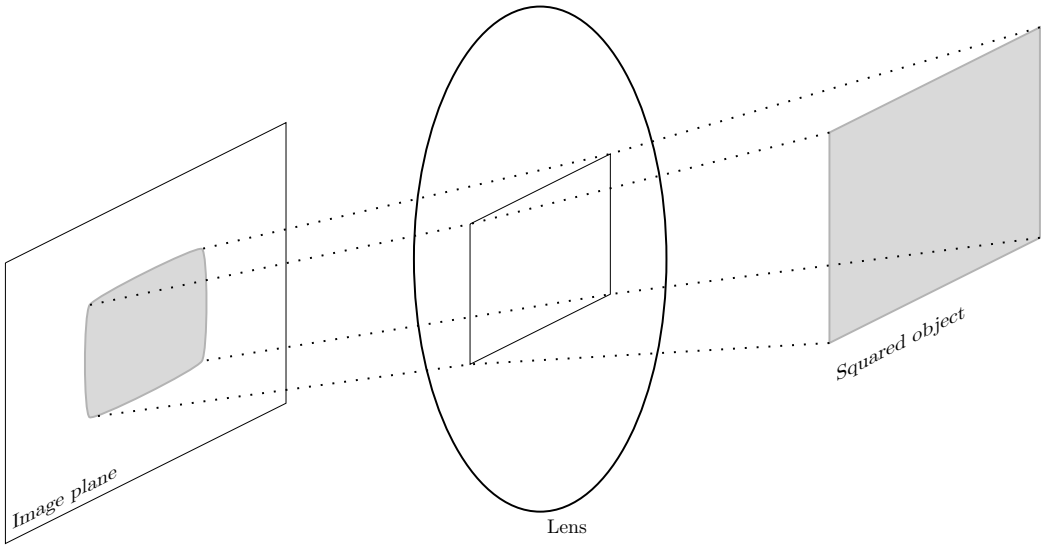
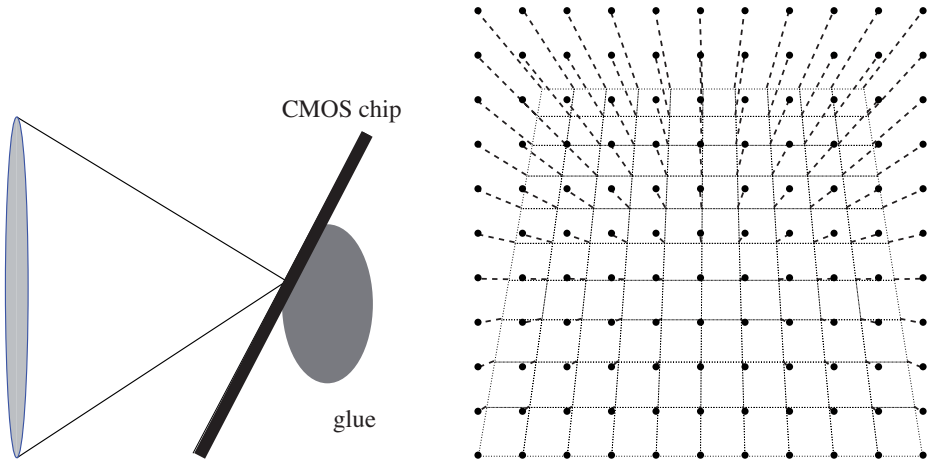


Figure 2.5: Due to radial distortion, the sides of a square paper to bow out on the image plane.



(a) Schematic of a cheap camera (b) Tangential distortion effect

Figure 2.6: Tangential distortion of a camera: when sensor and lens in a camera system are not perfectly aligned (a) image are affecting by tangential distortion, which causes an asymmetric displacement of each point of the image from their idea position. In (b), the ideal black dot are positioned on the distorted dotted grid due to tangential distortion.

2.4.3 Davison et al. Distortion Model

The distortion model implemented by Davison et al. in [1, 9, 21, 22] is, in fact, a simplified version of the plumb bob model where only the first 2 radial distortion parameters are considered, leading to the distortion correction function

$$\underline{h}_u(\underline{p}_d) = \underline{p} l(r_d), \qquad l(r_d) = 1 + k_1 r_d^2 + k_2 r_d^4. \tag{2.12}$$

Note that, while the function is very simplified with respect to the original one, it is still too complex to be inverted in a closed form, since the inversion of this function can be written as

$$\underline{p}_d = \underline{h}_d\left(\underline{p}\right) = \frac{1}{l(r_d)} \cdot \underline{p}_d, \tag{2.13}$$

with

$$\begin{aligned} r_u &= r_d l(r_d), \\ r_u &= \|\underline{p}\|_2, \end{aligned}$$

from which it is impossible to obtain the r_d expression in closed form. Davison et al. proposes to solve the last equation system using some numerical methods such as the Newton-Raphson method [23] to compute r_d and so $\underline{p}_d$. Starting from an initial approximation x_0 of a function $g(\cdot)$, Newton-Raphson method is able to find successively better approximations to the root according to the following equation

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}, \tag{2.14}$$

where $g'(\cdot)$ denotes the derivative of $g(\cdot)$ over x. In this specific case the function $g(\cdot)$ is chosen as

$$g(r_d) = r_d l\left(r_d\right) - r_u,$$

leading to the iteration equation

$$r_{d,i+1} = r_{d,i} - \frac{r_d l\left(r_d\right) - r_u}{r_{d,i} l'\left(r_{d,i}\right) + l\left(r_{d,i}\right)},$$

where

$$l'\left(r_d\right) = \frac{\partial l\left(r_d\right)}{\partial r_d} = 2 r_d f(r_d), \qquad f(r_d) = \left(k_1 + 2 k_2 r_d^2\right).$$

The initial value $r_{d,0}$ is set as $r_{d,0} = r_u$ and the iteration is performed until $i > I$, for a certain $I \in \mathbb{N}$. The final value $r_{d,I}$ can so be used to perform equation (2.13).

The Mono-SLAM algorithm, based on the Extended Kalman Filter (EKF) algorithm, requires to linearize equations to perform. For this reason, it is useful to compute jacobians of the two function here described. In particular from equations (2.12) if follows directly

$$\frac{\partial \underline{h}_u}{\partial \underline{p}_d} = l(r_d)\mathbb{I}_2 + \frac{\partial l\left(r_d\right)}{\partial r_d} \underline{p}_d \frac{\partial r_d}{\partial \underline{p}_d}, \tag{2.15}$$

where

$$\frac{\partial r_d}{\partial \underline{p}_d} = \frac{\|\underline{p}_d\|}{\partial \underline{p}_d} = \frac{1}{\|\underline{p}_d\|} \underline{p}_d^T = \frac{1}{r_d} \underline{p}_d^T, \tag{2.16}$$

and so

$$\frac{\partial \underline{h}_u}{\partial \underline{p}_d} = l(r_d)\mathbb{I}_2 + 2f(r_d)\underline{p}_d\underline{p}_d^T. \tag{2.17}$$

On the other hand, the jacobian related to equation (2.13) can not be directly computed since the function $\underline{h}_d(\cdot)$ is not expressed in closed form, however it is possible to numerically compute it by inverting $^{\partial \underline{h}_u}/_{\partial \underline{p}_u}$ as follows

$$\left.\frac{\partial \underline{h}_d}{\partial \underline{p}}\right|_{\underline{p}} = \left.\frac{\partial \underline{h}_u}{\partial \underline{p}_d}\right|^{-1}_{\underline{h}_d(\underline{p})}. \tag{2.18}$$

2.4.4 An Useful Distortion Model

A different distortion model proposed by [24] and used in the calibration tools proposed in Section 2.6 consists in modeling the distortion prediction function (instead of the correction function) in a very similar way as the plumb bob model (2.11) does, considering the first three parameters of the radial distortion and the first two of the tangential distortion. The function is defined as

$$\underline{h}_d(\underline{p}) = \underline{p}l(r_u) + \delta_t, \tag{2.19}$$

where

$$\underline{p}l(r_u) = 1 + k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6, \qquad r_u = \sqrt{u^2 + v^2} = \|\underline{p}\|_2,$$

and

$$\delta_t = \begin{pmatrix} \left(p_1\left(r_u^2 + 2u^2\right) + 2p_2uv\right) \\ \left(p_2\left(r_u^2 + 2v^2\right) + 2p_1vu\right) \end{pmatrix}.$$

Differently from the previous model presented, the possibility of writing the distortion prediction function in a closed form is very advantageous in the application of Mono-SLAM since this specific algorithm requires the use of the distortion prediction function much more times than the correction function. Computational time is saved since solving numerically an equation takes more time then computing directly the results from a closed function.

Obviously, the distortion correction function must be however defined and can not be written in a closed form. The iterative algorithm used to implement the $\underline{h}_u(\cdot)$ function is the following

$$\underline{p}_{i+1} = \frac{\underline{p}_d - \underline{\delta}_{t,i}}{l(r_{n,i})},$$

which is initialized with $\underline{p}_0 = \underline{p}_d$ and performed until $i > I$, for a certain $I \in \mathbb{N}$. Note that this algorithm is less efficient of the Newton-Raphson previously proposed, which is not used due to the complexity of the function which have to be invert. As done previously, the jacobians related to $\underline{h}_d(\cdot)$ and $\underline{h}_u(\cdot)$ must be defined. From equation (2.19) follows directly

$$\frac{\partial \underline{h}_d}{\partial \underline{p}} = l(r_n)\mathbb{I}_2 + \underline{p}\frac{\partial l(r_n)}{\partial r_n}\frac{\partial r_n}{\partial \underline{p}} + 2\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}\begin{pmatrix} v & u \end{pmatrix} + 2\begin{pmatrix} p_2 \\ p_1 \end{pmatrix} r\frac{\partial r}{\partial \underline{h}_n} + 2\begin{pmatrix} 2p_2u & 0 \\ 0 & 2p_1v \end{pmatrix}, \tag{2.20}$$

where

$$\frac{\partial l(r_n)}{\partial r_n} = 2r_n f(r_n), \qquad f(r_n) = k_1 + 2k_2 r_n^2 + 3k_3 r_n^4,$$

$$\frac{\partial r_n}{\partial \underline{p}} = \frac{1}{r_n}\underline{p}^T,$$

so that

$$\frac{\partial \underline{h}_d}{\partial \underline{p}} = l(r_n)\mathbb{I}_2 + 2f(r_n)\underline{p}\underline{p}^T + 2\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}\underline{h}_n^T\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + 2\begin{pmatrix} p_2 \\ p_1 \end{pmatrix}\underline{p}^T + 4\begin{pmatrix} p_2 u & 0 \\ 0 & p_1 v \end{pmatrix}. \tag{2.21}$$

While jacobian related to the distortion correction function in equation (2.4.4) is numerically computed by inverting the jacobian just defined, as

$$\left.\frac{\partial \underline{h}_u}{\partial \underline{p}_d}\right|_{\underline{p}_d} = \left.\frac{\partial \underline{h}_d}{\partial \underline{p}}\right|^{-1}_{\underline{h}_d(\underline{p})}. \tag{2.22}$$

A Simplified Distortion Model

The model just presented is often too complete to be used in normal application and can be simplified without loss in precision considering only the first two parameters of the radial distortion and no tangential distortion. With this simplification, the simplified distortion prediction function is given by

$$\underline{h}_d(\underline{p}) = \underline{p}l(r_u), \tag{2.23}$$

where

$$\underline{p}l(r_u) = 1 + k_1 r_u^2 + k_2 r_u^4, \qquad r_u = \sqrt{u^2 + v^2} = \|\underline{p}\|_2.$$

And now the distortion correction function is given by

$$\underline{p} = \underline{h}_u\left(\underline{p}_d\right) = \frac{1}{l(r_u)} \cdot \underline{p}_d, \tag{2.24}$$

with

$$r_d = r_u l(r_u),$$
$$r_d = \|\underline{p}_d\|_2,$$

and can be evaluated by mean of the Newton-Raphson algorithm as

$$r_{u,i+1} = r_{u,i} - \frac{r_u l\left(r_u\right) - r_u}{r_{u,i} l'\left(r_{u,i}\right) + l\left(r_{u,i}\right)},$$

where

$$l'\left(r_u\right) = \frac{\partial l\left(r_u\right)}{\partial r_u} = 2r_u f(r_u), \qquad f(r_u) = \left(k_1 + 2k_2 r_u^2\right).$$

The jacobians related to $\underline{h}_d(\cdot)$ and $\underline{h}_u(\cdot)$ must now be defined. From equation (2.19), it follows directly

$$\frac{\partial \underline{h}_d}{\partial \underline{p}} = l(r_n)\mathbb{I}_2 + \underline{p}\frac{\partial l(r_n)}{\partial r_n}\frac{\partial r_n}{\partial \underline{p}}, \tag{2.25}$$

where

$$\frac{\partial r_n}{\partial \underline{p}} = \frac{1}{r_n}\underline{p}^T,$$

so that

$$\frac{\partial \underline{h}_d}{\partial \underline{p}} = l(r_n)\mathbb{I}_2 + 2f(r_n)\underline{p}\underline{p}^T. \tag{2.26}$$

While the last jacobian related to the distortion correction function in equation (2.4.4) is numerically computed by inverting the jacobian just defined, as

$$\left.\frac{\partial \underline{h}_u}{\partial \underline{p}_d}\right|_{\underline{p}_d} = \left.\frac{\partial \underline{h}_d}{\partial \underline{p}}\right|^{-1}_{\underline{h}_d(\underline{p})}. \tag{2.27}$$

2.5 Camera Models

In this Section two different complete camera models are presented, the model used in the implementations of the Mono-SLAM algorithm proposed by Davison et al., and the model implemented in the well known Open Source Matlab Camera Calibration Toolbox used also in this work. Each model are constructed upon the theoretical consideration presented previously in this Chapter. This Section, together with the following one, is also devote to presents the benefits and limits of each model in the case of the specific task of Mono-SLAM.

2.5.1 Camera models applied to Mono-SLAM

Before deeply discussing the two camera models previously mentioned, a little excursus is necessary to understand how this models are used in the Mono-SLAM algorithm, which is the main subject of this Master Thesis. In particular, the algorithm needs to do two main tasks with a general camera model, independently from its implementation. It needs two functions to be defined, that are the *projection function* $\underline{h}(\cdot)$ and the *unprojection function* $\boldsymbol{h}(\cdot)$.

Projection function requires in input the 3D position of a point in the scene and returns the projected point in the distorted image plane, as follows

$$\underline{p}_d = \underline{h}(\boldsymbol{p}).$$

On the other hand, the *unprojection function* computes the epipolar line of a given point on the real image, i.e., the locus of all the possible points which project on the input argument of that function itself, as

$$\boldsymbol{h}^{\mathcal{C}} = \boldsymbol{h}(\underline{p}_d).$$

This parameters will be provided by the camera calibration phase, as discussed in the following Section.

The following models will be discuss in order to define these two main functions, and the corresponded jacobians required by the Mono-SLAM algorithm.

2.5.2 Davison et al. Camera Model

The camera model used by Davison et al. requires to know the values of the following parameters:

- the camera focal length f expressed in mm,
- the pixels dimension $d_u \times d_v$,
- the coordinates of the principal point of the camera $p_0^N = \begin{pmatrix} u_0^N & v_0^N \end{pmatrix}^T$ or $p_0 = \begin{pmatrix} u_0 & v_0 \end{pmatrix}^T$, with $u_0^N = d_u u_0$ and $v_0^N = d_v v_0$,
- the first two order radial distortion parameters k_1 and k_2.

In this model, the camera intrinsic matrix form is the following

$$\mathbf{K} = \begin{pmatrix} -\frac{f}{d_u} & 0 & u_0 \\ 0 & -\frac{f}{d_u} & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.28}$$

which can be decomposed in the two matrix

$$\mathbf{K}_x = \begin{pmatrix} \frac{1}{d_u} & 0 & u_0 \\ 0 & \frac{f}{d_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{K}^N = \begin{pmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

such that $\mathbf{K} = \mathbf{K}_x \mathbf{K}^N$.

To obtain the distorted point in $\mathcal{I}^N$-coordinates from a point $\boldsymbol{y}^{\mathcal{C}} = \begin{pmatrix} x & y & z \end{pmatrix}^T \in \mathcal{C}$, the following operations are performed: first of all, $\underline{p}^N$ is computed using $\mathbf{K}^N$ matrix,

$$\underline{\tilde{p}}^N = \mathbf{K}^N \boldsymbol{y}^{\mathcal{C}}, \qquad \underline{p}^N = \mathbf{K}^N \left(\boldsymbol{y}^{\mathcal{C}} \right) = \begin{pmatrix} -f\frac{x}{z} \\ -f\frac{y}{z} \end{pmatrix}.$$

$\underline{p}^N$ is then distorted according to the model presented in Section 2.4.3 at page 19, using equation (2.13) as

$$\underline{p}_d^N = \underline{h}_d^N \left(\underline{p} \right),$$

and finally, $\underline{p}_d^N$ is expressed in $\mathcal{I}$-coordinates using $\mathbf{K}_x$

$$\underline{\tilde{p}}_d = \mathbf{K}_x \underline{\tilde{p}}_d^N, \qquad \underline{p}_d = \mathbf{K}_x \left(\underline{p}_d^N \right) = \begin{pmatrix} \frac{1}{d_u} u + u_0 \\ \frac{1}{d_v} v + v_0 \end{pmatrix}.$$

Finally, the projection function $\underline{h}(\cdot)$ can be expressed as

$$\underline{h}(\boldsymbol{\psi}) = \mathbf{K}_x \left(\underline{h}_d \left(\mathbf{K}^N \left(\boldsymbol{\psi} \right) \right) \right), \tag{2.29}$$

with related jacobian given by

$$\frac{\partial \underline{h}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}} = \frac{\partial \underline{p}_d}{\partial \underline{p}_d^N} \frac{\partial \underline{p}_d^N}{\partial \underline{p}^N} \frac{\partial \underline{p}^N}{\partial \psi}, \tag{2.30}$$

where

$$\frac{\partial \underline{p}_d}{\partial \underline{p}_d^N} = \begin{pmatrix} \frac{1}{d_u} & 0 \\ 0 & \frac{1}{d_v} \end{pmatrix}, \qquad \frac{\partial \underline{p}^N}{\partial \psi} = \begin{pmatrix} -f\frac{1}{z} & 0 & f\frac{x}{z^2} \\ 0 & -f\frac{1}{z} & f\frac{y}{z^2} \end{pmatrix},$$

and $\partial \underline{p}_d^N / \partial \underline{p}^N$ is defined in equation (2.18).

On the other hand, unprojection function $\boldsymbol{h}(\cdot)$ is given by inverting equation (2.29), leading to

$$\boldsymbol{h}(\underline{p}_d) = \mathbf{K}^{N^{-1}} \left(\underline{h}_u \left(\mathbf{K}_x^{-1} \left(\underline{p}_d \right) \right) \right), \tag{2.31}$$

where

$$\underline{p}_d^N = \mathbf{K}_x^{-1} \left(\underline{p}_d^N \right) = \begin{pmatrix} d_u(u - u_0) \\ d_v(v - v_0) \end{pmatrix},$$
$$\boldsymbol{h}^C = \mathbf{K}^{N^{-1}} \left(\underline{p}^N \right) = \begin{pmatrix} -\frac{1}{f} u \\ -\frac{1}{f} v \\ 1 \end{pmatrix},$$

and $\underline{h}_u(\cdot)$ is defined in equation (2.12). The jacaobian relative to the last defined function is given by

$$\frac{\partial \boldsymbol{h}(\underline{p}_d)}{\partial \underline{p}_d} = \frac{\partial \boldsymbol{h}^C}{\partial \underline{p}^N} \frac{\partial \underline{p}^N}{\partial \underline{p}_d^N} \frac{\partial \underline{p}_d^N}{\partial \underline{p}_d}, \tag{2.32}$$

where

$$\frac{\partial \boldsymbol{h}^C}{\partial \underline{p}^N} = \begin{pmatrix} -\frac{1}{f} & 0 \\ 0 & -\frac{1}{f} \\ 0 & 0 \end{pmatrix},$$
$$\frac{\partial \underline{p}_d^N}{\partial \underline{p}_d} = \begin{pmatrix} d_u & 0 \\ 0 & d_v \end{pmatrix},$$

and $\partial \underline{p}^N / \partial \underline{p}_d^N$ is defined in equation (2.15).

The model used by Davison et al. has an important disadvantage with respect to the model that are presented in the next Section, that is the fact that the parameters d_u and d_v and f can not be estimated from any calibration process. In particular, pixel dimensions may be obtained from the datasheet of the camera system, which often is not available, especially for low cost commercial cameras. For this reason, the majority of the models coming out from calibration are developed in order to do not need to know the parameters f, d_u and d_v, which are not measurable, but only their composition $f_u = f/d_u$ and $f_v = f/d_v$, which are measurable. In particular that normalized model is obtained by normalizing all the parameters in order to have $f = 1$ in an unknown units of measurement. Obviously, also the distortion coefficient k_1 and k_2 are involved by normalization. Their normalized values can be deduced considering that the $l(\cdot)$ function, which represents the relative distortion of a point with respect to its distance from the principal point, in invariant to normalization, so that

$$l(r_d) = l(\dot{r}_d) \rightarrow 1 + k_1 r_d^2 + k_2 r_d^4 = 1 + \dot{k}_1 \dot{r}_d^2 + \dot{k}_2 \dot{r}_d^4,$$

must hold true, where the "hat" $\cdot$ indicates a normalized parameter.

Since all parameters are normalized in order to have $\dot{f} = 1$, it comes out that

$$\frac{r_d}{\dot{r}_d} = f.$$

and subsequently

$$\dot{k}_1 = k_1 \left(\frac{r_d}{\dot{r}_d} \right)^2 = k_1 f^2, \qquad \dot{k}_2 = k_2 \left(\frac{r_d}{\dot{r}_d} \right)^4 = k_1 f^2.$$

Hence, it has been demonstrated that normalized (dimensionless) model is equivalent to the original one.

2.5.3 An Usefull Camera Model

The second model is the one used in the well-known camera calibration Toolbox of Matlab, the open source OpenCV library and the implementation of the Mono-SLAM algorithm proposed in this Master Thesis. It requires to know the following parameters:

- the camera focal length expressed in pixel coordinates f_u and f_v,
- the coordinates of the principal point of the camera $p_0 = \begin{pmatrix} u_0 & v_0 \end{pmatrix}^T$,
- the first three orders radial distortion parameters k_1, k_2 and k_3.
- the first order tangential distortion parameters p_u and p_v.

Note that in this model knowing real dimension of the pixels and the focal length are not required. The liner part of the model here presented is expressed by the camera intrinsic matrix

$$\mathbf{K} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.33}$$

which can be decomposed in the two matrices $\mathbf{K}_x = \mathbf{K}$ and $\mathbf{K}^N = \mathbb{I}_3$, since, as mentioned previously, the model supposes that $f = 1$ in an unknown units.

As in model of Davison et al. , the projection and unprojection functions are given by

$$\underline{h}(\boldsymbol{\psi}) = \mathbf{K}_x \left(\underline{h}_d \left(\mathbf{K}^N \left(\boldsymbol{\psi} \right) \right) \right), \tag{2.34}$$

$$\boldsymbol{h}(\underline{p}_d) = \mathbf{K}^{N^{-1}} \left(\underline{h}_u \left(\mathbf{K}_x^{-1} \left(\underline{p}_d \right) \right) \right), \tag{2.35}$$

where

$$\begin{aligned} \underline{p}_d &= \mathbf{K}_x \left(\underline{p}_d^N \right) = \begin{pmatrix} f_u u + u_0 \\ f_v v + v_0 \end{pmatrix}, \\ \underline{p}_d^N &= \mathbf{K}_x^{-1} \left(\underline{p}_d \right) = \begin{pmatrix} \frac{1}{f_u}(u - u_0) \\ \frac{1}{f_v}(v - v_0) \end{pmatrix}, \\ \underline{p}^N &= \mathbf{K}^N \left(\boldsymbol{\psi}^{\mathcal{C}} \right) = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \\ z \end{pmatrix}, \\ \boldsymbol{h}^{\mathcal{C}} &= \mathbf{K}^{N^{-1}} \left(\underline{p}^N \right) = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}, \end{aligned}$$

and the distortion prediction and correction functions are defined by the model presented in Section 2.4.4 at page 20 in their complete forms in equations (2.4.4) and (2.4.4) or in their simplified version in equations (2.23) and (2.24). From now on, with the term simplified model it is intended the camera model here presented when using the simplified version of the distortion model.

Finally, the two jacobians of the two main function are given by

$$\frac{\partial \underline{h}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}} = \frac{\partial \underline{p}_d}{\partial \underline{p}_d^N} \frac{\partial \underline{p}_d^N}{\partial \underline{p}^N} \frac{\partial \underline{p}^N}{\partial \psi}, \tag{2.36}$$

$$\frac{\partial \boldsymbol{h}(\underline{p}_d)}{\partial \underline{p}_d} = \frac{\partial \boldsymbol{h}^C}{\partial \underline{p}^N} \frac{\partial \underline{p}^N}{\partial \underline{p}_d^N} \frac{\partial \underline{p}_d^N}{\partial \underline{p}_d}, \tag{2.37}$$

where

$$\begin{aligned} \frac{\partial \underline{p}_d}{\partial \underline{p}_d^N} &= \begin{pmatrix} f_u & 0 \\ 0 & f_v \end{pmatrix}, \\ \frac{\partial \underline{p}^N}{\partial \boldsymbol{\psi}} &= \begin{pmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{pmatrix}, \\ \frac{\partial \boldsymbol{h}^C}{\partial \underline{p}^N} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \\ \frac{\partial \underline{p}_d^N}{\partial \underline{p}_d} &= \begin{pmatrix} \frac{1}{f_u} & 0 \\ 0 & \frac{1}{f_v}, \end{pmatrix} \end{aligned}$$

$\partial \underline{p}_d^N / \partial \underline{p}^N$ and $\partial \underline{p}^N / \partial \underline{p}_d^N$ are defined in equations (2.21) and (2.22) or (2.26) and (2.27) for the simplified version of the model.

Differently from the camera model used by Davison et al. , the model here just presented is ready to the phase of camera calibration, which aims to estimate the parameters of this model best suiting for the camera system used. In the next Section, two different calibration procedure are presented and applied to the model just discussed.

2.6 Camera Calibration

Under the label camera calibration are grouped all the methods that can estimate the camera parameters (both intrinsic, extrinsic and distortion) processing some photos, taken by the system to be calibrated, of an object with known geometric. Most of calibration procedures are based on a chessboard pattern with known dimensions. This pattern can be used for intrinsic and distortion measurement – taking some photos from different position – and for extrinsic matrix estimation – in this case the pattern defines the world RF. This approach is used both in the OpenCV library and the camera calibration toolbox for Matlab.

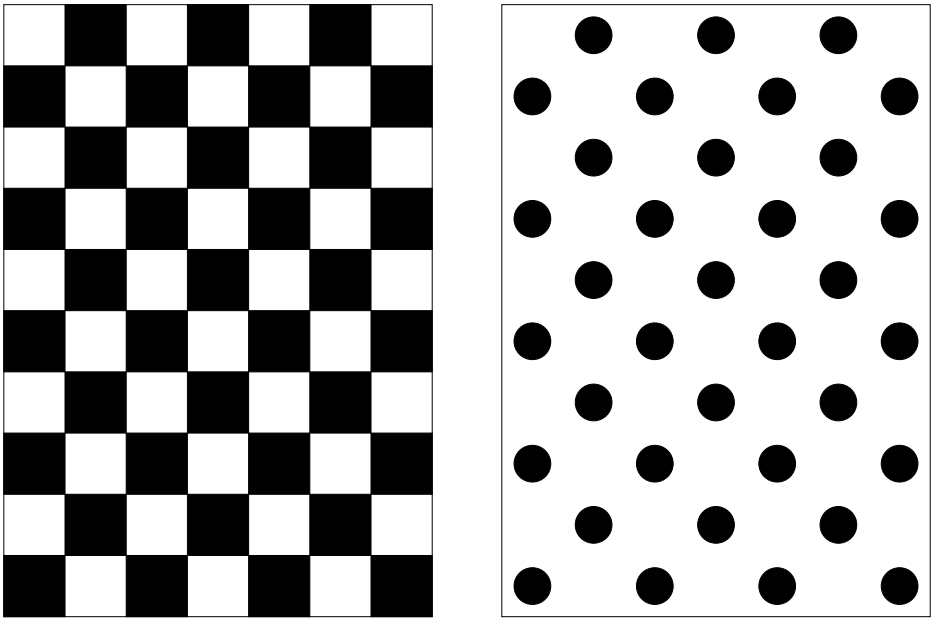
This Section is devoted to present the calibration phases required to obtain the parameters of the camera model described in Section 2.5.3 at page 24, both the complete model and the simplified one, reporting also the results obtained in the calibration of the camera used in the implementation of the Mono-SLAM algorithm. At the end of this Section, a comparison of the two models and related calibrations is presented, explaining which model best suits the Mono-SLAM algorithm and why.

2.6.1 Camera Calibration: an overview

The main idea of any camera calibration procedure is to target the camera on a structure with known geometry having many individual and identifiable points whom position is well known in a predefined reference frame. By viewing this structure from different point of view and so comparing the various location of the identifiable points of the structure with their known position, it is possible to compute the relative poses of the camera with respect to the structure and as well as the parameters of the camera model.

Ideally every structure can be used to perform the calibration operations, however 2D patterns, as in Figure 2.7, are very common for their simplicity. Some calibration methods proposed in literature rely on three-dimensional objects (such as a box covered with markers), but it is definitely much easier deal with flat patterns since building precise 3D calibration patterns are very difficult. Moreover 2D patterns can be easily built up in order to be compatible with automatic pattern recognition algorithms. The patterns used are also asymmetric, in order to allow the algorithm to measure their orientations without ambiguity. The pattern should be built up taking care of preserving it to be exactly planar, in order to avoid measurement errors, for this reason it is convenient lay down the pattern on a rigid planar structure, such as a piece of glass. Furthermore the size of the pattern needs to be known precisely and should be measured by hand since printer can rescale its input, however, if the extrinsic parameters of the camera are not important, the size of the chessboard can be arbitrarily set, since the intrinsic calibration parameters are all scaled in pixels.

After that the pattern is ready, the user have to provide to the tools several different images of the pattern taken from different points of view. To obtain good results from the calibration, it is very important to capture the pattern from different distances and different locations in the camera (i.e., top, center, left, right, etc.), and also from different inclinations with respect to the



(a) Chessboard pattern (b) Circles grid

Figure 2.7: Two example of the most used patterns to perform camera calibration operation. Note that both present an asymmetry which allows to define uniquely the board orientation. Automatic algorithm are able to extract the grid corners of the centers of the circles.

camera optical axis. In Figure 2.8 it is presented a short calibration sequence taken from real data, the calibration results provided afterwards are related a bigger sequence containing these images. Then, a preprocessing phase is in charge to extract the key points defining the structure geometry from each photos. Finally, the real calibration algorithm will compute the several model parameters by comparing the position of the several key points with their absolute location in a standard reference frame. Usually, this reference frame is chosen in order to have x and y axes lying on the plane defined by the pattern and aligned with it, and the z axis perpendicular to that plane, as in Figure 2.13. Often calibration methods provide also the camera extrinsic matrix for each calibrating photo where the word reference frame $\mathcal{W}$ coincides with the pattern reference frame.

An interesting overview on various camera calibration techniques are presented in [19] while a state of the art algorithm for camera calibration is preseted in [25]. The two methods here presented are inspired by [25] and [24].

2.6.2 OpenCV Camera Calibration

OpenCV does not provide an implemented toolbox able to automatically compute the camera intrinsic parameters of its models, however it provides several routines which are able to compute those parameters. Moreover, surfing over the net several calibration toolbox implemented using OpenCV can be easily found.

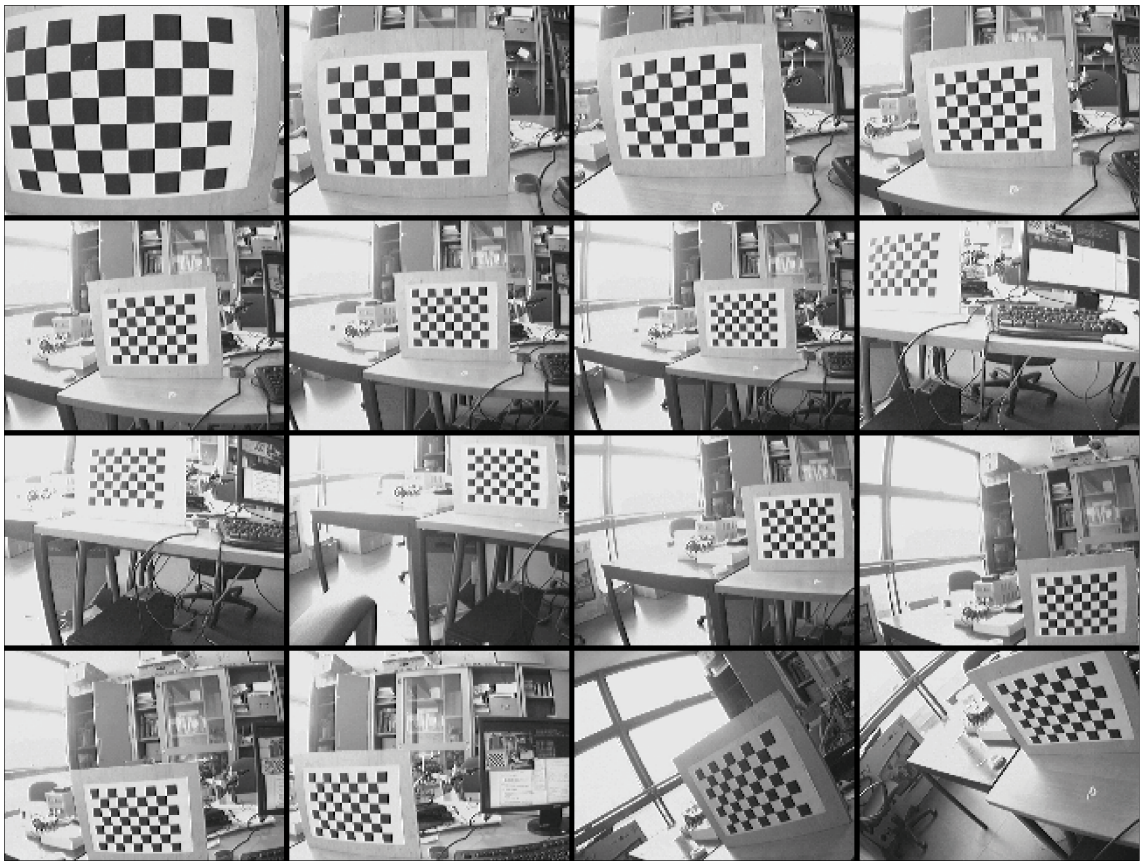
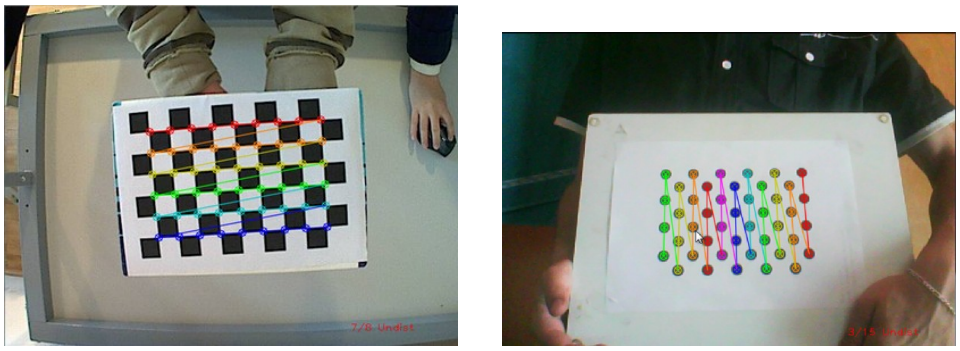


Figure 2.8: Short images calibration sequence. Calibration results reported in this Section are related a bigger sequence containing these images.

The implementations used during this work of thesis is the one proposed in the OpenCV documentation page[1]. The key point extraction algorithm are performed using the one of two functions `findChessboardCorners` and `findCirclesGrid`, which are able to extract respectively the chessboard corners and the circles center of the grid. An example of their output is given in Figure 2.9, where the found key points are drawn on the image using function `drawChessboardCorners`.

After that, the function `calibrateCamera` will compute the camera parameters according to the OpenCV model presented previously in this Chapter. The function returns the mean value of the re-projection error affecting the calibrated model. Using this calibration process, OpenCV is able, thanks to the function `undistort`, to correct the distortion presented on a given image, moreover the library provides several interesting functions very useful to work with the calibrated camera.

[1] `http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html`



(a) Chessboard pattern (b) Circles grid

Figure 2.9: Key points extracted by mean of the OpenCV functions **findChessboardCorners** (a) and **findCirclesGrid** (b) and drawn on the image using the function **drawChessboardCorners**.

Note that this algorithms are implemented in the ROS node Camera Calibration[2], the software is very useful because it presents a simple and user friendly interface and allows user to perform calibration only when enough images have been captured.

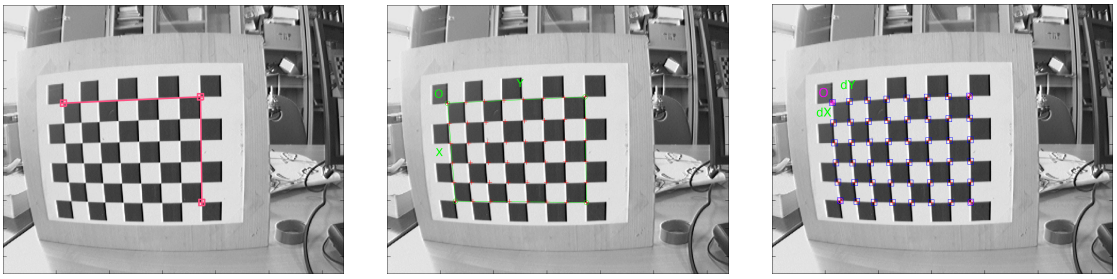
2.6.3 Matlab Camera Calibration Toolbox

The Matlab calibration toolbox is a very powerful toolbox allowing to calibrate a camera, according to the Matlab camera model presented in Section 2.5.3 at page 24. It provides a very simple and intuitive user interface which allows to perform the calibration basic operations, i.e., read images, extract corners and calibrate the camera, but also provides advance features which allows user to visualize results, analyze errors and improve calibration adding new images.

The grid corner extraction phase is implemented as a semiautomatic algorithm which requires the user to manually select the 4 extern corners of the patterns (a chessboard) of each provided image, as shown in Figure 2.10, the program is then able to find the internal corners autonomously.

After that all corners are extracted, the calibration procedure can be launched. According to the tool reference page, calibration is done in two steps: first initialization, and then non-linear optimization. The initialization step computes a closed-form solution for the calibration parameters without considering any lens distortion. The non-linear optimization step minimizes the total projection error (in the least squares sense) over all the calibration parameters (both intrinsic and extrinsic), by considering also distortion effects.The toolbox provides also some tools to improve the calibration, such as the possibility of automatically re-extract the corners using the information given by the first calibration, that is useful because the user may have not done a very careful job at extracting the corners on some highly distorted images. Re-launching the calibration procedure after recomputing corners should improve the calibration. Moreover, it is possible to remove images with high noise or add new image to the calibration.

[2] `http://www.ros.org/wiki/camera_calibration`



(a) External corners selection (b) Ideal location of inner corners (c) Extraction of inner corners

Figure 2.10: In the Matlab calibration toolbox, the user is required to select manually the 4 external corners of the pattern for each image (a), after that the algorithm estimates the positions of the inner corners (b) and finally finds their real positions (c).

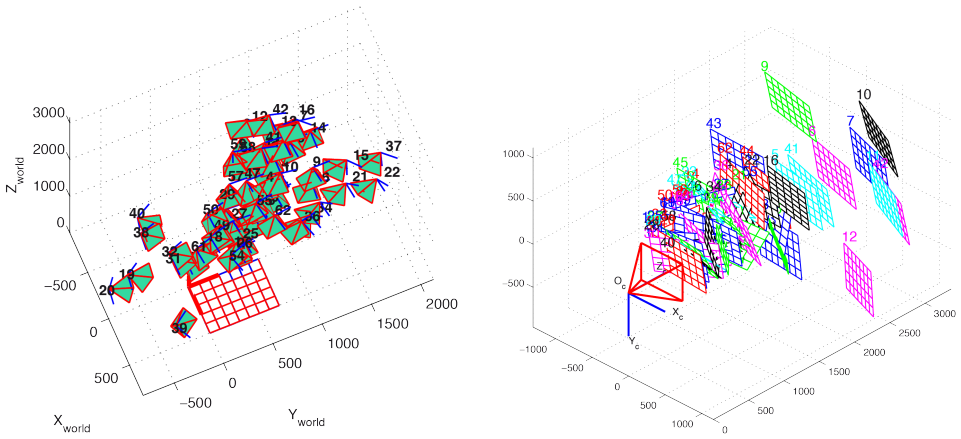
After the calibration phase, in addition to the calibration parameters, a several visual informations about the calibration goodness are available. First of all, images showing the relative camera position, i.e., the estrisinc calibration results, of each image with respect to the pattern and the other way around are available, as shown in Figure 2.11. Moreover, graphics related to the distortion effects introduced by the camera are available, as reported in Figure 2.14, presenting the two distortion contributions (radial and tangential) individually and together. In addition, a plot of the projection errors of all the extracted corners in both directions are reported in Figure 2.15; this last image is very useful to understand the goodness of the calibration obtained and also to find out some specific images which introduce big errors in the calibration (outliers), which can be easily deleted from the calibration procedure, note that the projection error plot are available both for the entire set of images or for a restricted set of images selected by the user.

Using the calibration obtained, the toolbox is also able to compute the extrinsic parameters relative to a given image, as shown in Figure 2.16, where the pattern reference frame is projected on the image, and to correct the distortion of a given image, as shown in figure Figure 2.12

2.6.4 Comparison on the calibration methods

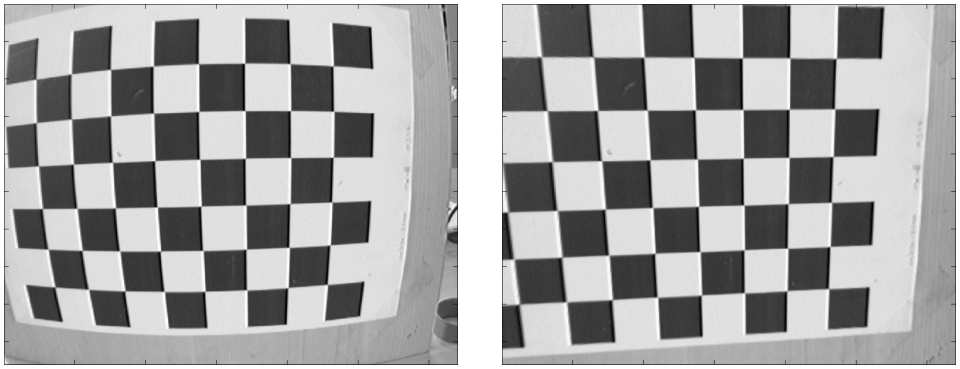
The real digital camera used to implement the Mono-SLAM algorithm was calibrated according to both calibration methods here presented and according to the complete and simplified models prosed in Section 2.5.3 at page 24. The calibration results of both methods and models are reported in Table 2.1. From the results it comes out that the two methods are actually equivalent. In fact, the values estimations are practically the same, despite the values of the tangential distortion components, which are however very small. Also the overall projection error of both methods are quite similar, note that the Matlab toolbox are a little bit more accurate in calibration, however both errors are strongly below 1 pixel, i.e., are negligible with respect to the discretization error introduced by the digital sensor. In addition, the simplified model does not lose more precision with respect to the complete one while the simplification introduced are remarkable.

While the two calibration tools are equivalent, the user have to choose which is most suitable for the application. If a very powerful tools for error analyzing is required, obviously the Matlab toolbox is the more useful, on the other side, if a fully automatic toolbox is preferred, an OpenCV of camera calibration is a very good choice. Please note that all analysis available on the Matlab



(a) Camera positions with respect to the pattern (b) Pattern positions with respect to the camera

Figure 2.11: Visualization of the Matlab toolbox extrinsic calibration. The relative positions of the camera with respect to the pattern reference frame (a) and the relative positions of the pattern with respect to the camera reference frame (b) for each calibration image.



(a) Original image (b) Undistorted image

Figure 2.12: The Matlab toolbox is able to correct distortion of a given image (a) providing the undistorted image (b).

Toolbox can also be implemented using OpenCV, which is however more elastic then the other tool. For example, OpenCV allows to define a personalized calibration pattern.

Regarding the two different models, since the errors of the simplified model is quite similar to the complete one, using the simplified model is better especially because, as seen previously, this model allows to perform some operation, in particular the numerical solution of the distortion correction function, faster. Obviously, which is the best model is strongly from the camera system. For instance, in a wide angle camera, the complete model is certainly most suitable

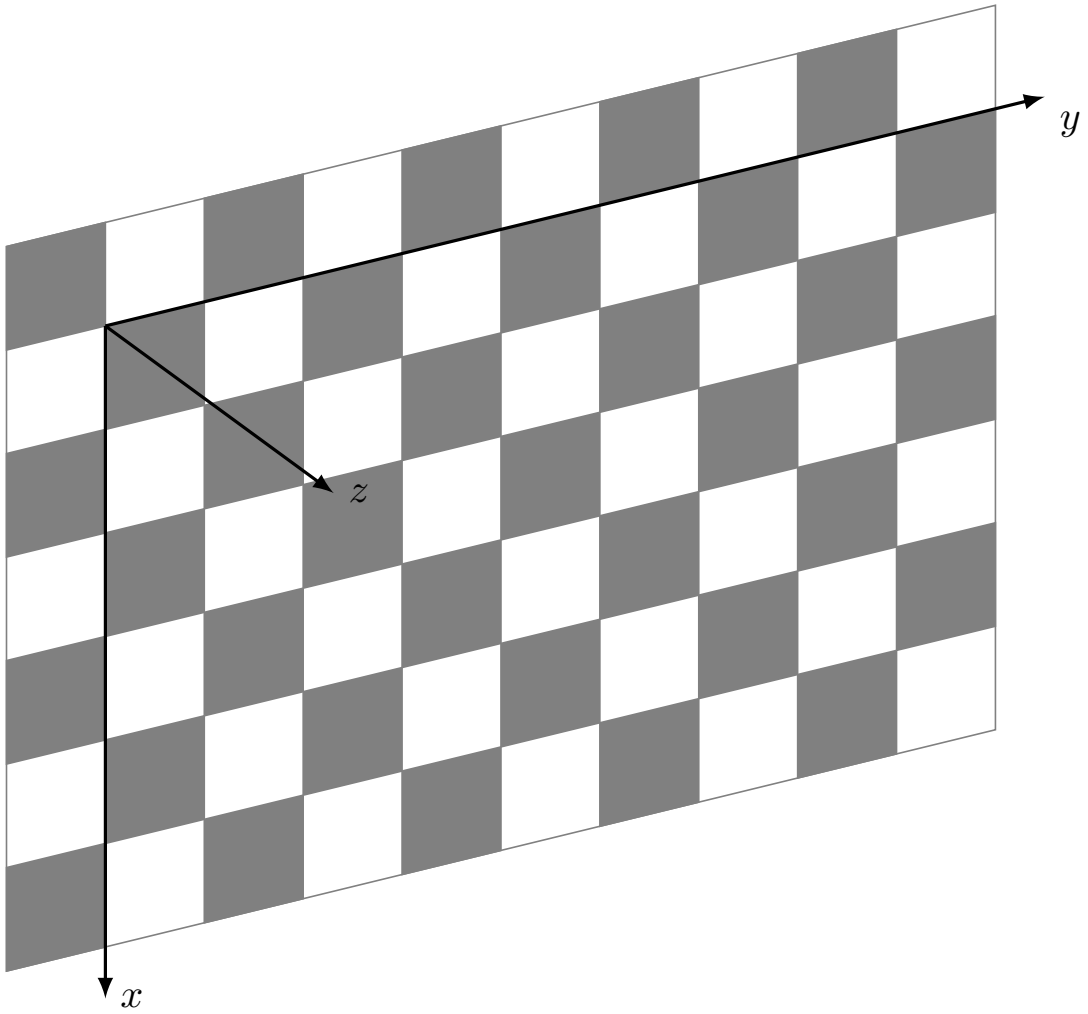
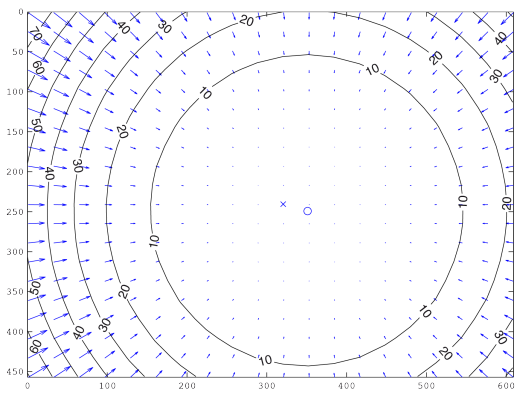
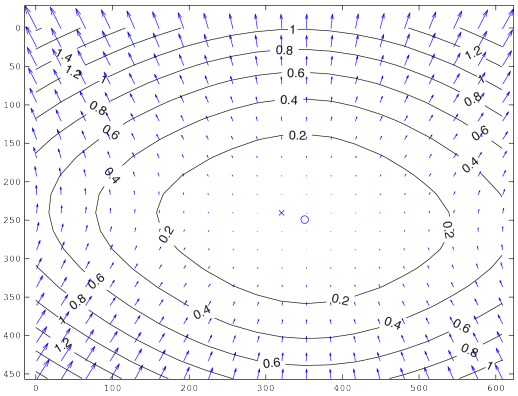


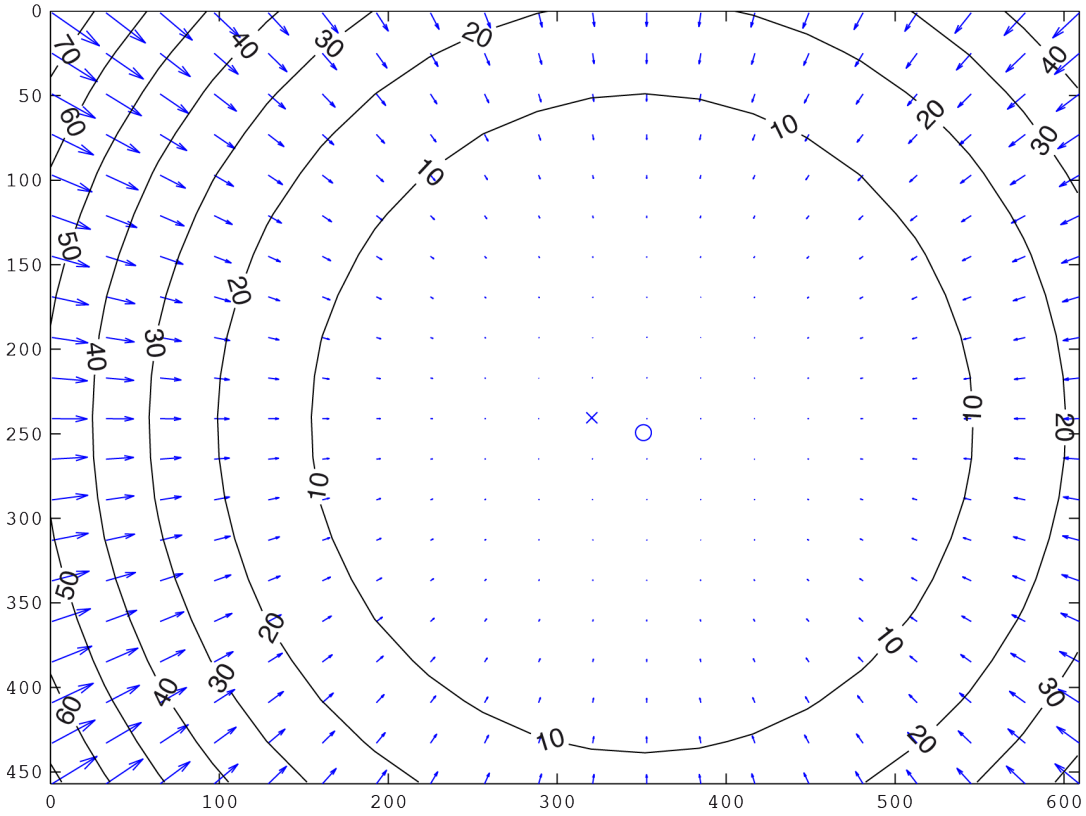
Figure 2.13: Reference frame defined by a calibration pattern. The x and y axes lie on the plane defined by the pattern and aligned with it, while the z axis is perpendicular to that plane according to the right hand rule.



(a) Radial component of distortion model

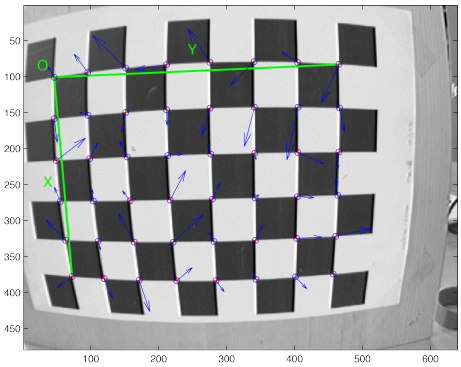


(b) Tangential component of distortion model

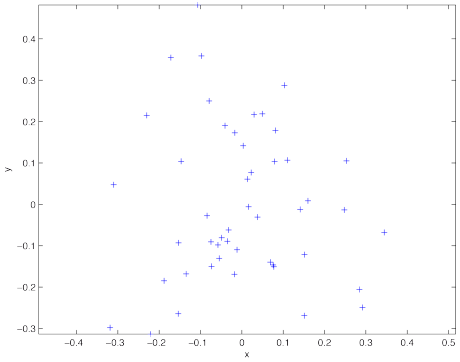


(c) Complete model of distortion

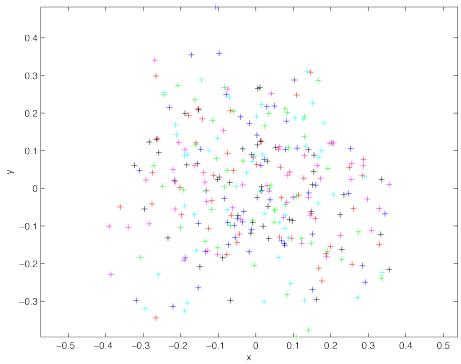
Figure 2.14: The calibration of the lens distortion visualized in the radial (a) and tangential (b) components, together with the complete model (c). Note that the camera principal point, labeled with o, is not coincident with the image center, labeled with x.



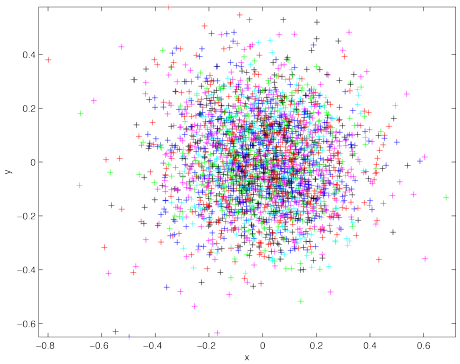
(a) Projection errors of a specific image



(b) Projection errors magnitude of a specific image, the same as (a)



(c) Projection errors magnitude of a user defined set of 10 images



(d) Projection errors magnitude of the overall set of images

Figure 2.15: The Matlab toolbox provides several ways to visualize the projection errors, i.e., the difference between the corner positions computed by the model and their real position in the image. Figure (a) shows the errors, represented by arrows, on a specific image, while (b), (c) and (d) plot the error magnitude respectively of a specific image, a user defined set of images and the overall set of calibration images.

Table 2.1: Calibration results of Matlab Calibration Toolbox and OpenCV calibration module performed over the complete and simplified camera models presented in Section 2.5.3 at page 24.

	Complete Model				**Simplified Model**			
	Matlab		**Opencv**		**Matlab**		**Opencv**	
	Value	Error	Value	Error	Value	Error	Value	Error
f_u	563.73	1.054	562.04	–	259.72	0.910	259.42	–
f_v	558.60	0.965	557.42	–	555.57	0.898	555.32	–
u_0	349.92	1.489	348.05	–	350.01	1.040	348.70	–
v_0	248.29	1.332	245.45	–	241.61	0.780	241.14	–
k_1	-0.45528	0.006	-0.4517	–	-0.434	0.00316	-0.425	–
k_2	0.30653	0.029	0.2975	–	0.208	0.0067	0.183	–
k_3	-0.13807	0.041	-0.1305	–	–	–	–	–
p_u	-0.00306	0.0005	-0.00199	–	–	–	–	–
p_v	0.00015	0.0004	0.00055	–	–	–	–	–
error	0.2508		0.2700		0.2612		0.2818	

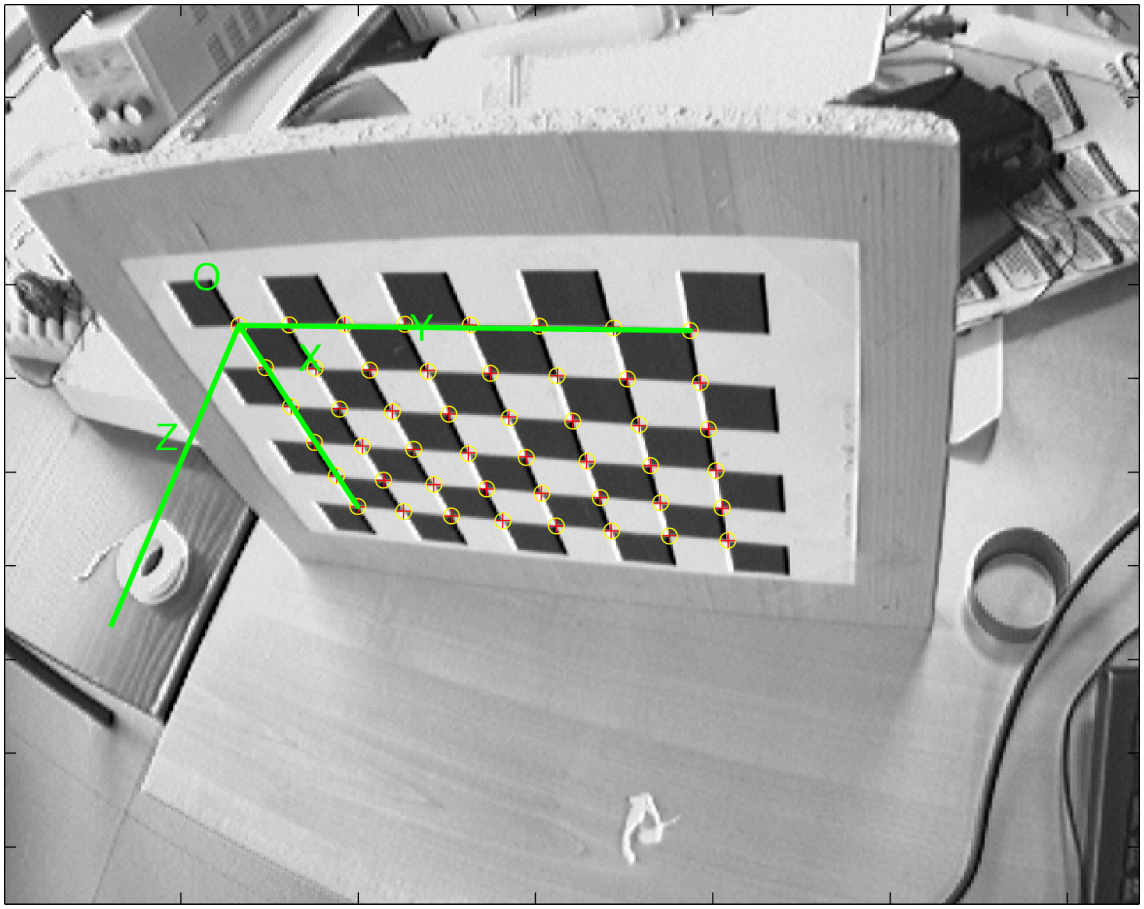


Figure 2.16: An example of extrinsic parameters computation in Matlab calibration toolbox, the parameters are visualized by projecting onto the image the pattern reference frame.

Chapter 3

Image Processing

Time is an illusion. Lunchtime doubly so.

Douglas Adams

The previous Chapter describes how it is possible to extract all the possible 3D information of a point detected in a single 2D image, explaining that the best a monocular camera can do is to estimate a line in the 3D environment where the a point must lies. For this reason, to estimate the position of this point it is necessary having more information, such as at list two images containing the point taken from two different point of view. Of course, the position of that point in the two images will be different, and it is necessary to establish a correspondence between the two projections of the same point in the two images to correctly perform depth estimation. Unfortunately, establishing this type of correspondence, i.e., performing correctly matching between two image, is not a simple and trivial job, especially because prospective and occlusions may strongly change the appearance of the projection. This Chapter deals with the still open problem of matching correspondence between two images by discussing the most important algorithms and solutions to perform this task. In addition, some basic image processing techniques are introduced. All of this after a brief overview on the terminology and data structures used in computer vision, provided in the next paragraph.

In computer vision, an image is represented by a matrix of pixels. A pixel is a data structure containing all information about the smaller distinguishable part of the image. For image in gray scale, it is usually represent by 8 bit values, providing so 256 different shades between black (0) and white (255). Color pixels usually contains three different 8 bit values, labeled as *channels*. Information are coded in the channels depend according to a specific *color model*. The most known and used color model is the *Red Green Blue* (RGB) model, where each channel represents the amount of intensity of the correspoing color: red, green and blue respectively. However, several different color models has been proposed in literature, such as *Hue Saturation Value* (HSV), *Luminance-Bandwidth-Chrominance* (YUV), *Cyan-Magenta-Yellow-Key (black)*CMY(K), etc. About the HSV, it is interesting while it was elaborate to better represent the way in which human eyes perceive colors. In HSV, the three channels represent respectively hue, saturation, the value, i.e., intensity or brightness, of the represented color. Differently from RGB, if two colors appears to human eye very similar, they are "near" in the HSV space. Two good reference to deepen about these arguments are [15, 16]. Regarding this work and the algorithm here presented, only images in gray-scale are considered, since gray-scale images are less noisy then color images and, at the same time, they do not lose information about the environment conformation.

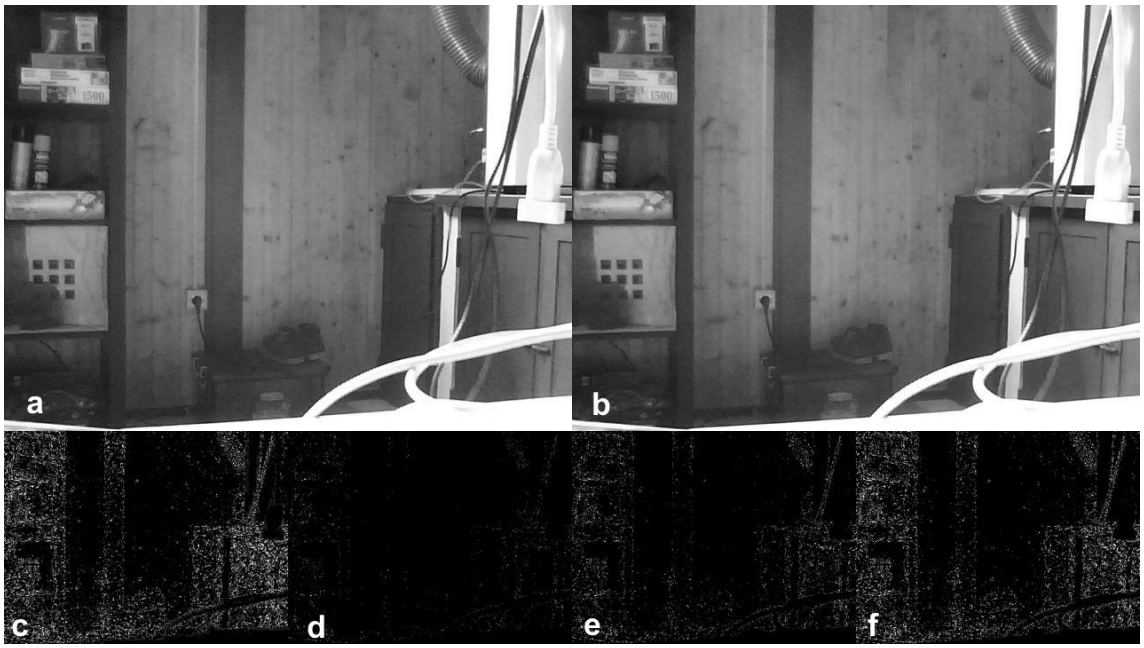


Figure 3.1: To consecutive frames form a video stream (a) and (b) of a not moving camera could present differences in pixels even if to human eyes they appear completely the same. Figure (c) shows the difference between the two frames (opportunely rescaled) while Figure (e), (f) and (b) enhanced pixels which difference are greater than 10, 5 and 3 respectively.

The first pixel of the matrix represents the top left pixel in the image, which is usually considered the origin of the image reference frame $\mathcal{I}$. The u-axis is horizontal pointing right and the v-axis is vertical pointing down. Please note that, since usually in language programming, the first entry of the address of a the matrix indicates the rows and the second one indicates the column, using this notation the element $I(x, y)$ is the element having $u = y$ and $v = y$.

In processing images, the complete raw data of an image is never used. First of all, raw data of an image tends to be much noisy so that it is almost infeasible for comparison purpose. E.g., in theory it is easy to detect motion from a frame stream coming from a camera which do not move, since it is sufficient to compare each pixel of the previous frame with the same pixel of the current one. If the pixel value is different, it can be assume that some movements in the area of the pixel has occurred and so that a different object is observed at those pixel. Unfortunately, this approach do not work in real environments, since cameras are very sensible to changes of illuminations. So two consecutive images which could be appear exactly the same to human eye, can be completely different if analyzed by a camera. As shown in Figure 3.1. Moreover, the amount of data coming from a video camera is too much to be practically processed meeting real time constraints. For this reason it is a common approach in computer vision algorithms to perform a preprocessing phase, which aims at improving quality by removing noise and compress the image, before processing the image itself. Finally note that the majority of commercial cameras automatically preprocess the images before sending them to the main calculator.

A quite obvious but important results of computer vision is that an aggregation of adjacent

pixels may contain much more information then information contained in all these pixel individually. Such aggregation is usually labeled as *feature* in computer vision, and it is probably the most important entity computer vision approaches have to deal with. Selecting, describing and matching features are, in fact, still nowadays open problems and several different approaches have been proposed to perform these tasks. Note that, usually, literature is a little bit confusing when using terms such as *interest point* and *feature*. In this Chapter, *interest point* will refer to a specific point in the image while feature indicate an interest point with the addition of some data that enable it to be compared with other features.

This Chapter is organized as follows: Section 3.1 presents an overview about important computer vision algorithms and solutions related to this work, and Section 3.2 introduces and discusses severals approaches to detect and match interest points

3.1 Image Processing Techniques

Computer vision is a very interesting and wide field of studying which has reached several important results in the last years, moreover every day new goals are reached and new targets are set. Hence, it is quite impossible to present a complete overview about the enormous amount of applications and solutions regarding computer vision. For this reason, in this Section only the important results directly related to this Master Thesis are provided. In particular, the concept of *convolution* signals are introduced and applied to images. Moreover, *convolution* will be used to present a very simple but effective approach regarding interest points matching, i.e., patches *cross correlation*. Finally, the *integral image* will be introduced, which is a solution which can help to speed up the cross correlation process.

3.1.1 Convolution and Image Filtering

Given two mono dimensional and real signals $x_1(\cdot)$ and $x_2(\cdot)$, the *convolution product* between this signals is given by

$$(x_1 \otimes x_2)(t) = \int_{-\infty}^{+\infty} x_1(\tau)x_2(\tau - t)d\tau,$$

or, in case of time discrete signal

$$(x_1 \otimes x_2)(k) = \sum_s x_1(s)x_2(s - k).$$

Of course the convolution can be extended to multidimensional signals, in particular for 2D signals it is defined as

$$(I_1 \otimes I_2)(u, v) = \sum_r \sum_s I_1(r, s)I_2(r - u, s - v), \tag{3.1}$$

where I_1 and I_2 are in general signals or, as in this case, images. The image convolution just described is at the base of the spatial filtering technique, which allows to modify the image according to a matrix known as *convolution kernel*. Since the mathematical formulation of convolution implies that the second signal is flipped, due to the minus sign, very often in computer vision the flipped kernel is labeled as *convolution mask*. However, in the majority of filters, the kernel is symmetric, so in computer vision fields this two entity are usually confused.

In practice, the image convolution with a mask consists in moving the mask on the image and, for each position of the mask: the value of the pixel in the output image corresponding to the center

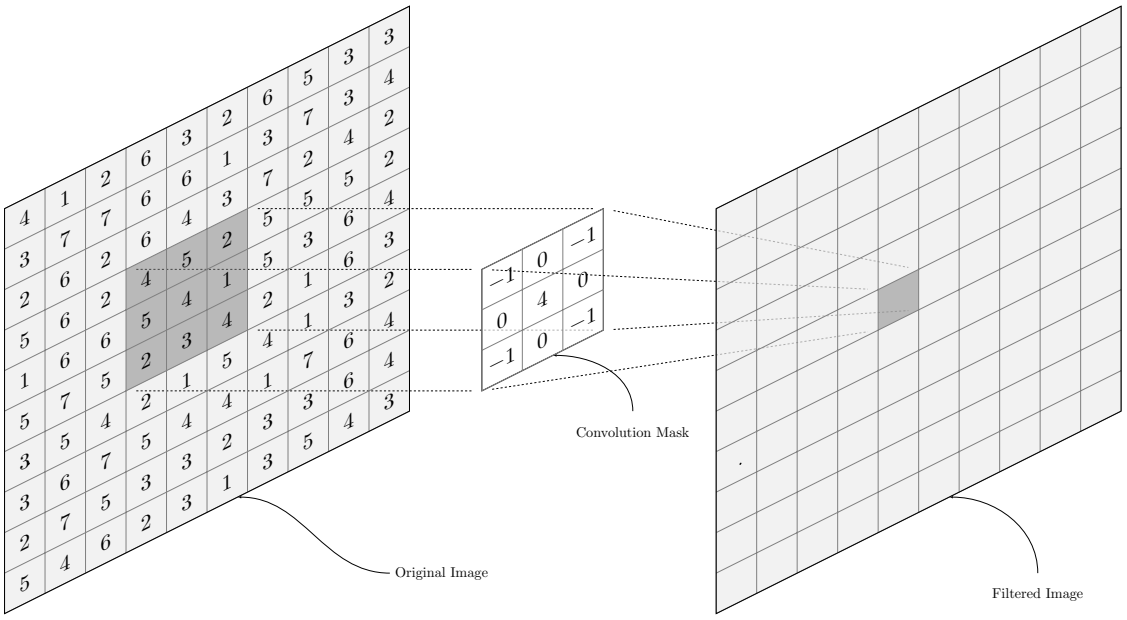


Figure 3.2: Image convolution. The convolution mask is moved on the image. The pixel on the filtered image corresponding to the center of the convolution mask is computed as the sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the mask.

of the kernel is calculated by multiplying each kernel value by the corresponding input image pixel values. This procedure is described in Figure 3.2.

Some very useful filters will be now described, their effect is visualized in Figure 3.3. Given an image I, the two derivatives of the image along horizontal end vertical axis are defined by

$$I_u = \frac{\partial I}{\partial u}, \qquad I_v = \frac{\partial I}{\partial v}, \tag{3.2}$$

however, they can be approximately computed by means of the two kernels k_u and k_v defined as

$$k_u = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, \qquad k_v = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^T, \tag{3.3}$$

so that

$$I_u \simeq I \otimes k_u, \qquad I_v \simeq I \otimes k_v. \tag{3.4}$$

Their effects are visualized in Figures 3.3b and 3.3c respectively. Moreover Figure 3.3e shows the original image filtered by a Gaussian low-pass kernel, where the mask is a discretized 2D gaussian signal of a given covariance matrix $\sigma^2 \mathbb{I}_2$, and Figure 3.3f shows the effect of a motion blur filter, where the mask is a straight line which direction defines the direction of the motion blur.

Since convolution is very useful to model undesirable effects affecting real images, such as motion blur (Figure 3.3f) or defocusing (Figure 3.3e), its inverse operator, known as *deconvolution*, is very useful to restore blurred images if the convolution kernel is known. Unfortunately, image

Figure 3.3: Some filters have been applied to image (a): (b) and (c) shown results on horizontal and vertical derivative approximation respectively. (e) is the results of a gaussian filter and (f) of a motion blur filter.

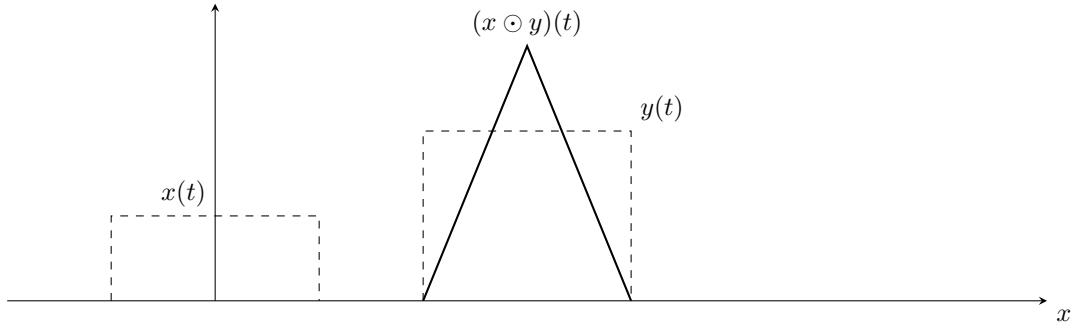


Figure 3.4: Example of cross-correlation applied to 1D signals. $x(t)$ and $y(t)$ are two time signals having the same shape. The cross-correlation $x \odot y$ has a maximum locate into the translation which makes x to fit y.

restoration is not a simple task to perform, since convolution is not an invertible operator, and, moreover, usually convolution kernel of real images are not uniform in the image, e.g., in real image affected by motion blurring, the magnitude of the blur depends on the depth of the point considered, so each point is affected by different motion blur. Image restoration is, however, possible in some conditions and several solutions have been proposed to perform this task. For more information about this topic, please refer to [15, 26].

All filters built upon convolutions are labeled as *linear filters*. Of course, there can be defined several non linear filters based both to convolution and non linear operators. Some interesting examples of this filters are at the base of well known algorithm of corner detection, as discussed in Seciton 3.2.

3.1.2 Patch Matching

An important operator strongly related to convolution is *correlation*. Correlation applied to two signal is simply defined as

$$(x_1 \odot x_2)(t) = x_1(t) \otimes x_2(-t) = \int_{-\infty}^{+\infty} x_1(\tau) x_2(\tau + t) d\tau, \tag{3.5}$$

and, of course, in case of images, it is given as follows

$$(I_1 \odot I_2)(u, v) = I_1(u, v) \otimes I_2(-u, -v) = \sum_r \sum_s I_1(r, s) I_2(r + u, s + v). \tag{3.6}$$

Note that perform convolution between an image and a kernel is exactly the same of perform correlation between an image and the correspondent mask, according to the definition of kernel and mask given previously.

Correlation is very useful since it is able to find the value of the strangulation to be applied to a signal in order to match another signal having similar shape. A 1D representation of this property of correlation is given in Figure 3.4. For this reason, correlation operator is a very useful and simple solution to find the position in an image of a part of that image. This operation is known as *patch matching*.

A *patch* is probably the most simple example of feature associated to an interest point, such as it is defined as a sub image of given dimension (usually small) extracted from an image around a specific pixel. The problem of patch matching requires to find the position of the center of the patch inside the original image, i.e., the interest point.

Since the linear *correlation* operator is dependent from scale factors, that is

$$(k_1 \cdot x_1) \odot (k_2 \cdot x_2) \neq x_1 \odot x_2, \tag{3.7}$$

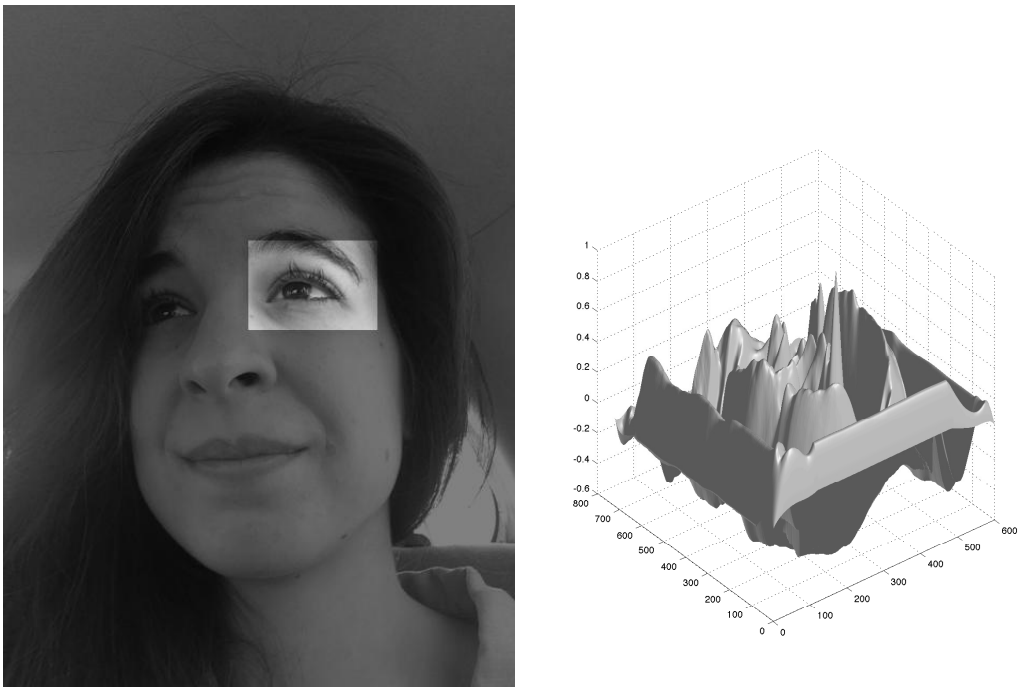
the *normalized cross correlation* operator is a better solution to perform patch matching. Given two patches P and P' of the same dimensions, the cross correlation score between the two patches is defined as

$$C_{NCC}(P, P') = \frac{1}{n} \sum_u \sum_v \frac{\big(P(u, v) - \bar{P}\big)\big(P'(u, v) - \bar{P'}\big)}{\sqrt{\sigma_P^2 \sigma_{P'}^2}}, \tag{3.8}$$

where n is the number of pixels contained in each patch and $\bar{P}$ and σ_P^2 are respectively mean and covariance of the patch, defined as

$$\bar{P} = \frac{1}{n} \sum_u \sum_v P(u, v),$$

$$\sigma_P^2 = \frac{1}{n} \sum_u \sum_v \big(P(u, v) - \bar{P}\big)^2 = \frac{1}{n} \sum_u \sum_v P^2(u, v) - \bar{P}^2.$$



(a) Original image with enhanced patch

(b) Normalized cross correlation results

Figure 3.5: The enhanced patch in figure (a) has been matched in the original image according to normalized cross correlation. Figure (b) provide the results of the cross correlation operation of the whole image.

The advantages of using normalize correlation is that the score $C_{NCC}(P, P')$ is limited in the range $[-1,1]$, where 1 indicates a perfect match while -1 a perfect mismatch, i.e., the patch is compared to its inverted patch.

To match a patch inside an image it is so necessary to compute the normalize cross correlation score of the patch itself and each patch extracted by the image and centered in all the possible positions. The position with the best cross correlation score is the center of the given patch inside the image. Note that this operation is an example of non linear filtering of the image. An example of normalized cross correlation is provided in Figure 3.5.

Integral image

An *integral image* is an image where each pixel contains the sum of the intensity of all pixel to the left and above this pixel of a given image. This concept was first introduced by Crow in [27], and it mathematical formulation is expressed as

$$\begin{aligned} int_I(u,v) &= \sum_{s \leqslant u} \sum_{t \leqslant v} I(s,t) \\ &= I(u,v) + int_I(u-1,v) + int_I(u,v-1) - int_I(u-1,v-1). \end{aligned} \tag{3.9}$$

Advantages in using integral images lie in the fact that, once constructed, the summation of all pixel values contained in a rectangle $\mathcal{R}$ in the images, defined by the four vertices upper left $\underline{p}_1$,

upper right $\underline{p}_2$, lower left $\underline{p}_3$ and lower right $\underline{p}_4$ is simply given by

$$\sum_{\underline{p}\in\mathcal{R}} I(\underline{p}) = int_I(\underline{p}_4) + int_I(\underline{p}_3) - int_I(\underline{p}_2) - int_I(\underline{p}_1). \tag{3.10}$$

Using integral image allows to speed up the matching process, which requires to computed summation of pixels values, and has find many application over the years. In particular regarding patch cross correlation, if the two integral matrices int_I and int_{I^2} are available, where I^2 of an image is the square of the image I, the computation of $\bar{P}$ and σ_P^2 can be speeded up.

3.2 Interest Points: Detectors and Descriptors

The previous Section presents some useful computer vision techniques and explains a simple algorithm to find a match between an image and a sub-image or patch. However, patches cross correlation is not a powerful techniques to match points between images, since it is actually few robust and can not be applied to general problems. In fact, to match correctly interest point between images, descriptors and matcher of that points must be robust to changes of illumination, rotation and general transformation. Unfortunately, patch matching are robust only to illumination changes, and also not very much.

Furthermore, a second issue regarding patch matching is the problem of how to choose the best points that are suitable to perform matching. For instance, also considering the best visual matching system available, it is impossible to track a point belonging to an object completely uniform, since no references can be found at all. The problem of choosing the interest point is not so trivial as it could seem, and several algorithm have been proposed in literature.

This Subsection aims at illustrating the most important algorithms performing interest points detection, or *corners detection*, which are early approaches to tackle this problem. Moreover, it aims to briefly illustrate the modern solution to perform interest points detection and matching based on *features*.

3.2.1 Corners Detectors

An interest point is a point on the image which can be detected with robustness and repeatability among different image, i.e., if an interest point is detect in an image, it must be detected also in other images containing that point taken from different points of view and with different illumination conditions. A common and early technique to achieve robustness is to select image regions with high gradient, that are regions in which quickly step from black (white) values to black (white) values, i.e., edges in the image. Several approaches to perform edge detection exists in literature, the laplacian operator, the Sobel operator and the Canny edge detector are the most popular among several solutions (Figure 3.6).

While edges have a good chances to be detected in different images, their are not suitable to perform point matching, since all the point of the same edge has actually the same appearance. Moreover, a single edge is not very suitable to perform motion tracking, since from the optical flow of an edge it is impossible to measure the speed components parallel to that edge, due to the *aperture problem*, as schematized in Figure 3.7. For this reason, detecting single points is usually preferable then detecting edges.

Hence, to perform matching corners detector have been proposed instead of edges. Note that a corner, in computer vision language, is not the intersection of two edge, but a general point in the image that can be robustly detected in images, and they could be intersection of edges, ends

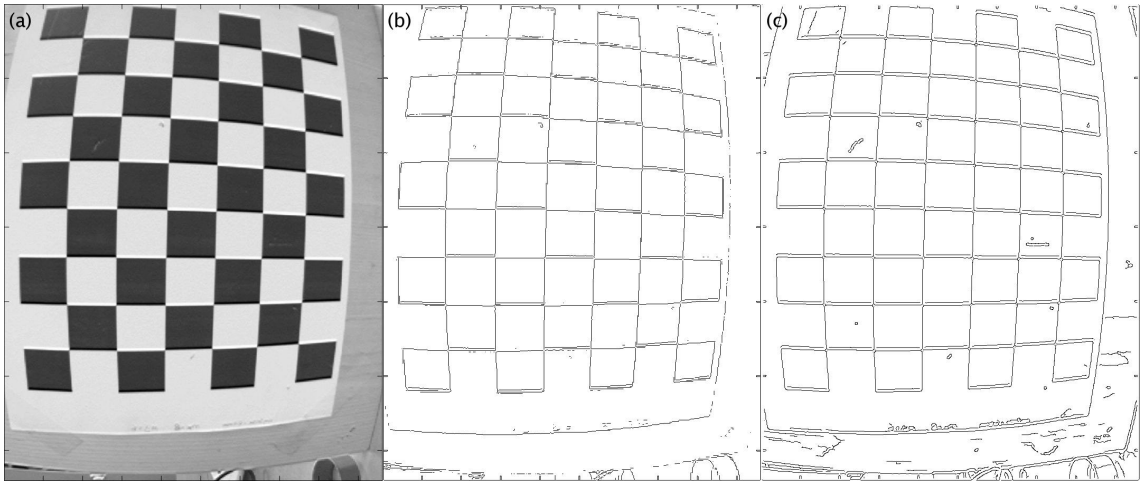


Figure 3.6: Two edge detectors operator. (a) is the original image, (b) and (c) are the results of Sobel and Canny edge detectors respectively.

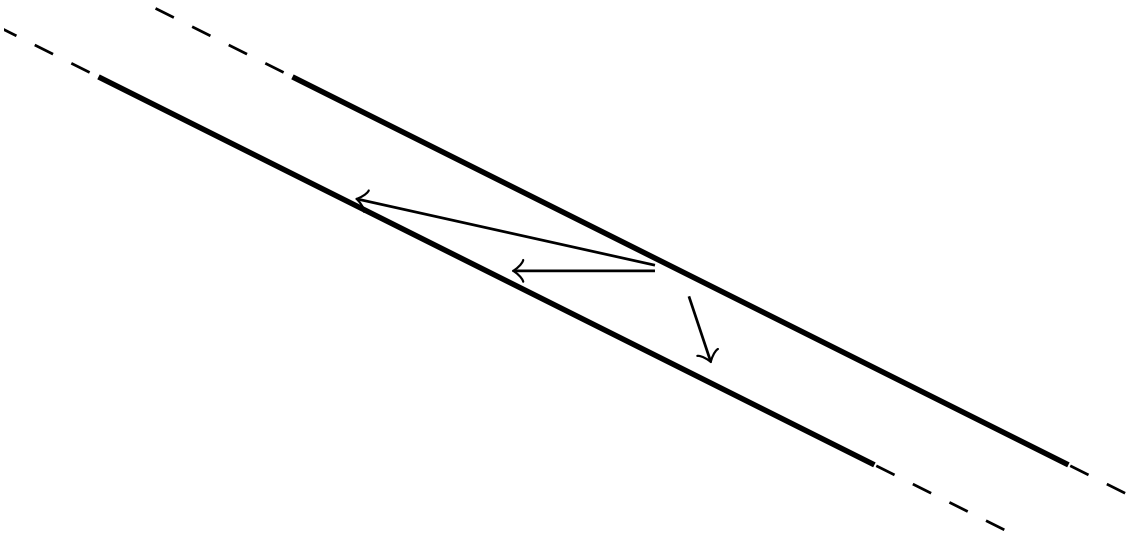


Figure 3.7: Detecting a single edge moving, the optical flow can not be estimated unambiguously.

of edges, local maxima or minimal points, etc. It is actually difficult to give a good definition of what a corner, or a interest point, is, since every different approach has and own definition. One of the most used definitions was the one proposed by Tomasi and Shi: "the right features are exactly those that make the tracker work best" [28].

One of the first approaches to detect corners was proposed by Moravec in [29]. The main idea of the approach is that a single pixel is distinguishable if it is different by the surrounding pixels. Moravec introduces a measurements of the self similarity of a point, defined as

$$S\left(\underline{p}\right) = \min_{\underline{p}' \in \mathcal{B}(\underline{p})/\underline{p}} SSD\left(P\left(\underline{p}\right), P\left(\underline{p}'\right)\right), \tag{3.11}$$

where $\mathcal{B}\left(\underline{p}\right)$ is the set of neighbors (horizontal, vertical and on the two diagonals) of $\underline{p}$ and the operator $SSD\left(\cdot,\cdot\right)$ is the the *sum of squared distances* of two patches, defined as

$$SSD\left(P_1, P_2\right) = \sum_u \sum_v P_1(u,v) P_2(u,v). \tag{3.12}$$

If $S\left(\underline{p}\right)$ is big, i.e., the patch centered in $\underline{p}$ is distinguishable from the surrounding patches, then the point is candidate to be a corner. The proposed algorithm computes the self similarity of each pixel in the image and return local maxima of the results as detected corners, corners are so defined as the points on the image having low self similarity. For uniform areas the interest measure will be close to zero, since the SSD should not differ significantly. If an edge is present in the direction perpendicular to the edge the SSD will be great, but in direction of the edge it should be small. This way edges should mostly be rejected, since the minimum of the calculated SSDs is used as interest measure. The main weakness of this detector is possible false classification of edges as interest points. The main limitation of this corner detectors is that it is not isotropic: if an edge is present that is not in the direction of the neighbours, then it will not be detected as an interest point.

The solution proposed by Moravec has been extended in an isotropic solution by Harris and Stephens in [30]. The solution computes the variation of intensity of a point $\underline{p} = \begin{pmatrix} u & v \end{pmatrix}^T$ defined as

$$E_{\underline{p}}(x,y) = \sum_s \sum_t w(s,t) \left(I\left(u+s+x, v+t+y\right) - I\left(u+s, v+t\right)\right). \tag{3.13}$$

Harris and Stephens note that $I\left(u+s+x, v+t+y\right)$ can be approximated using by the first order Taylor expansion

$$\begin{aligned} I\left(u+s+x, v+t+y\right) &\simeq I\left(u+s, v+t\right) + x \left.\frac{\partial I}{\partial u}\right|_{(u+s,v+t)} + y \left.\frac{\partial I}{\partial v}\right|_{(u+s,v+t)} \\ &\simeq I\left(u+s, v+t\right) + x I_u(u+s, v+t) + y I_v(u+s, v+t), \end{aligned}$$

where I_u and I_v are the approximations of the partial derivates of an image as defined in equation (3.4).

Given that approximation, the value of $E(\underline{p})$ can be computed as

$$\begin{aligned} E_{\underline{p}}(x,y) &\simeq \sum_s \sum_t w(s,t) \left(x I_u(u+s, v+t) + y I_v(u+s, v+t)\right)^2 \\ &\simeq \begin{pmatrix} x & y \end{pmatrix} \left[\sum_s \sum_t w(s,t) \begin{pmatrix} I_u^2(u+s, v+t) & (I_u I_v)(u+s, v+t) \\ (I_u I_v)(u+s, v+t) & I_v^2(u+s, v+t) \end{pmatrix} \right] \begin{pmatrix} x \\ y \end{pmatrix} \\ &\simeq \begin{pmatrix} x & y \end{pmatrix} \mathbf{H}_{\underline{p}} \begin{pmatrix} x \\ y \end{pmatrix}. \end{aligned} \tag{3.14}$$

Hence the operator $E_{\underline{p}}(x, y)$ measures the *weighted sum of squared distances* between a patches centered in $\underline{p}$ and a patched translated by (x, y). The weights $w(\cdot, \cdot)$ makes the function isotropic, in particular the isotropy is achieved if $w(\cdot, \cdot)$ is smooth and circular, such as a Gaussian. As in the previous approach, a corner is detected if $E_{\underline{p}}(x, y)$ as a local maximum in its center. Moreover, the evolution of that function along (x, y) can be analyzed by the study of the $\mathbf{H}_{\underline{p}}$ matrix, also called *Harris matrix*. In particular, studying the eigenvalues λ_1 and λ_2 of $\mathbf{H}_{\underline{p}}$ it is possible to distinguish between normal points, points belonging to edges and corners, as it follows

- if $\lambda_i \approx 0 \ \forall i = \{1,2\}$, then no interest point is detected,
- if $\lambda_i \gg 0 \ \forall i = \{1,2\}$, then a corner is detected,
- otherwise the points belongs to an edge.

Furthermore, Harris and Stephens propose a methods to speed up the algorithm by avoiding to compute eigenvalues of $\mathbf{H}_{\underline{p}}$. In particular, they define

$$R_{\underline{p}} = \det \mathbf{H}_{\underline{p}} - k \operatorname{trace} \mathbf{H}_{\underline{p}},$$

where k is a parameter to be empirically determined. $R_{\underline{p}} \gg 0$ corresponds to a corner while $R_{\underline{p}} \ll 0$ to an edge. Moreover, in practical application, to improve robustness, a corner is detected if $R_{\underline{p}} > R_{min}$ for a given R_{min}. This approach to corners and edged detector is known as *Harris operator*.

A very similar approach is the one proposed by Tomasi and Shi in [28]. In this work, Tomasi and Shi demonstrate that computing $\min(\lambda_1, \lambda_1)$ produces more robust results than using the operator $R_{\underline{p}}$ as in Harris corners detector, while it requires more computational efforts. Note also that the Harris operator results depend heavily on the given parameters of k, R_{min} and the size of the compared image patches, thus finding the right parameters for a specific application might take some fine tuning. On the other hand, Shi-Tomasi operator requires only two parameters, that are the response threshold λ_{min}, i.e., the minimum value of $\min \lambda_i$ admissible and patch size. Shi-Tomasi corners extractor is the algorithm used to find new features in the Mono-SLAM algorithm implemented by Davison et al. [1, 9] and also in the one implemented in this work.

3.2.2 Feature Descriptors

Once an interest point or corner has been detected on an image, the Mono-SLAM algorithm is able to track it along the video stream. Tracking is performed according to patch matching as described in the previous Section. However, the simple approach of patch matching has some important limitations such it is not invariant to scale and rotation. In fact, patch matching fails when a patch is compared with the same patch rotated of a sufficient angle and when the patch is compare with the same patch but zoomed. There exist several approaches to solve problem of scale and rotation in interest points matching, and this techniques are based on the computer vision concept of *feature*, that is an interest point with associated some information to perform comparison. In the following discussion, some overview on the most important features detectors and descriptors approaches is provided.

The algorithms introduced here detect interest points and compute for all suitable feature point candidates a descriptor. The descriptor is basically a vector which stores all the information needed to compare one feature with another one. A descriptor does not contain raw image data like an image patch, but higher level information calculated from the underlying image data and transformed in such a way that the contained information becomes scale and rotational invariant. The computations for a feature descriptors is, compared to the computations needed for corner

detectors is quite costly, so that the calculation of the features presented in the following will be slower compared to corner detectors. There are several computer vision applications that employ feature descriptors, especially in the domain of object recognition. Typically in this application scenario multiple descriptors are used to describe a single object. For real-time applications with high frame rates ($30 - 60ms$) where computational speed becomes crucial usually simpler features based on corner detectors are used.

Scale Inveriant Feature Transform (SIFT) proposed by Lowe in [31, 32] is probably the most well-known and widely used local descriptor. It is invariant both in rotation, scale, and is one of the most robust approaches. On the other hand, it requires a lot of computational resources so that often it can not be applied in real time application.

Baya et al. in [33] have proposed an approach, labeled as *Speeded Up Robust Features* (SURF) by adopting similar approaches for scale and rotation invariance of SIFT combined with efficient approximations to speed up the computation. SURF features are faster than SIFT but less robust in some case.

An alternative solution to SIFT has been proposed by Alcantarilla et al. in [34]. The KAZE features are build upon non linear filtering instead of Gaussian filters as in SIFT and SURF. The advantage of this features consists in the fact that non linear filters can make blurring locally adaptive to the image data, reducing noise but retaining object boundaries, obtaining superior localization accuracy and distinctiviness.

Finally, an interesting work able to perform similarly to SURF but faster and able to achieve real time application is the ORB approach [35].

Chapter 4

The Reconstruction Problem

It is a mistake to think you can solve any major problems just with potatoes

Douglas Adams

The two previous Chapters 2 at page 8 and 3 at page 37 explain respectively how a camera system works and how it is possible to match correspondent points in two different images. The notions they introduce are then used in the next Chapters 5 at page 58 to analyze and implement the Mono-SLAM algorithm, proposed by Davison et al. in the works [1, 9, 21, 22], which aims at reconstructing both scene and camera motion by a stream video given by the camera in real time.

The problem which Mono-SLAM tries to solve is a very big and complex problem labeled under the name of *reconstruction problem*, which, in its most simpler definition, aims at computing the 3D position of a point in the space using only 2D information coming from a camera system. For instance, in the Mono-SLAM formulation, the points to be reconstructed are some points of the scene plus the optical center positions of the camera at each frame. Please consider that from a single image is only possible to define a locus of points $\mathcal{L}(\underline{\mathrm{p}})$ in the environment in which the interesting point lies. This locus, known as *epipolar line*, can be is define as follows

$$\mathcal{L}(\underline{p}) = \left\{ \boldsymbol{p} \middle| \underline{h}(\boldsymbol{p}) = \underline{p} \right\} = \left\{ \boldsymbol{p} = \lambda \boldsymbol{h}(\underline{p}), \lambda > 1 \right\}, \tag{4.1}$$

where $\underline{h}(\boldsymbol{p})$ and $\boldsymbol{h}(\underline{p})$ are defined in Chapter 2.

While the problem definition is really simple, solving it is very complex, indeed several different and complex approaches have been proposed in literature. This Chapter is so devoted at introducing, from a very intuitive point of view, some of these approaches, i.e., *stereo triangolation*, *Structure from Motion* (SFM) and the *Mono-SLAM* algorithm, by explaining their benefits and limits. All the approaches will be exposed, without loss of generality, in a very simplified version and several images and examples will be provided to better understand the concepts. During the following discussion, moreover, some important issues affecting the reconstruction problem in general and the Mono-SLAM approach in particular will be introduced, that are the *scale loss* of the reconstruction and the *inverse resolution* of depth estimation. This concepts will then explained again, from a more theoretical point of view, in the next Chapter.

To simplifies formulas and notations, the following simplification are used: first of all, the camera system is consider ideal with no distortion and modeled as a pinhole camera; then, only a 2D projection of the environment is considered, in particular the plane yz, so that x-axis of $\mathcal{C}$ and u-axis of $\mathcal{I}$ are hidden. However, all the considerations hold true also for the hidden dimension.

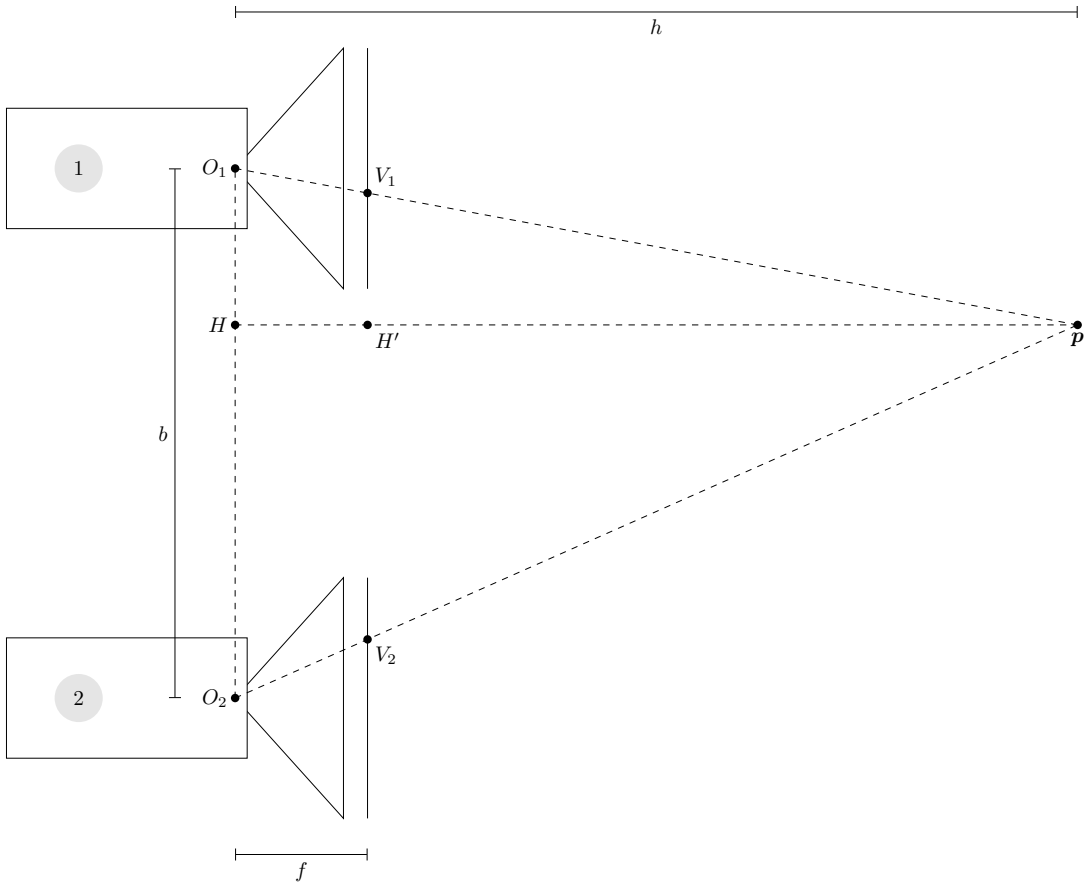


Figure 4.1: Example of a stereo system configuration, the point $\boldsymbol{p}$ is triangulate according to its projections v_1 and v_2 on the two cameras.

Moreover, no errors on measurement and matching is considered. Finally, only a point, labeled with $\boldsymbol{p}$, contained in the environment, is considered as key point, i.e, it is the only subject of the reconstruction.

4.1 Stereo Triangulation

Stereo triangulation is a well known technique able to reconstruct the 3D position of each interesting point in the scene by measuring its parallax, i.e., the displacement of the point viewed from two different points of view. A stereo system is composed by two identical cameras disposed in such a way that their image planes are coplanar and their optical axes are shifted by a known distance b, referred to as *baseline*, as shown in Figure 4.1.

The goal of stereo triangulation is to compute the value of h, also known as the *dept* of the point. With reference to Figure 4.1, note that the two triangles O_1O_2P and V_1V_2P are similar,

from which it comes out

$$\frac{\overline{PH'}}{\overline{PH}} = \frac{\overline{V_1V_2}}{\overline{O_1O_2}} \rightarrow \frac{h-f}{h} = \frac{b-\Delta v}{b} \rightarrow h = b\frac{f}{\Delta v} = b\frac{f}{\Delta v}, \tag{4.2}$$

since

$$\overline{V_1V_2} = |-v_1 + b + v_2| = b - (v_1 - v_2) = b - \Delta v,$$

where the Δv is called *disparity*. Since h is inversely proportional to the disparity $d = \Delta v$, then depth resolution obtained by triangulation decreases the farther away an object is from the optical center of a camera. Indeed, if the disparity is big, i.e., the point is near to the system, then a small change of disparity does not change the depth h much. On the other side, if the disparity is very small, a small change in Δv implies big change in h. From equation (4.2) is comes out that the error

$$\delta h = \left|\frac{\partial h}{\partial d}\right| \delta d = \frac{bf}{d^2}\delta d = \frac{h^2}{bf}\delta d.$$

Through depth resolution for far away objects can be increased by increasing baseline b this does only help to a certain degree, since depth resolution does not decrease linearly. Furthermore an increased baseline might inhibit depth estimation for close objects, since close objects might not be in the field of view of both cameras. This is an inherent problem of stereo vision but also of all systems which try to reconstruct the scene, leading to the incapacity of measuring a points with low parallax with a good precision. As it will explain in the next Chapter, Mono-SLAM has to face with a related problem due to the non linearly of depth resolution.

4.2 Structure from Motion

Structure from Motion (SfM) is a technique capable to reconstruct the camera motion and the scene by offline processing the stream video of a moving camera. For simplicity, in the following example the camera is free to move only on the y axis, and can not rotate. In Figure 4.2 it is shown a possible configuration of the same camera in three different allowed positions in the scene.

The problem can be mathematically formulated as finding the values of the the unknowns parameters for which the epipolar lines converges in $\boldsymbol{p}$, i.e., solving the equation system

$$\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} y_i \\ 0 \end{pmatrix} + \lambda_i \begin{pmatrix} v_i \\ f \end{pmatrix} \quad \forall i. \tag{4.3}$$

Note that the equations system presents $2i$ equations and $2i + 2$ unknown variables, so it presents 2 degrees of freedom which do not allow it to be solved. The first degree of freedom is related to the fact that the system can rigidly translate along the y axis, however it can be easily avoid inasmuch the absolute positioning of the camera is not really important. Usually the first position of the camera is set to zero, leading to the equation system

$$\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} y_i \\ 0 \end{pmatrix} + \lambda_i \begin{pmatrix} v_i \\ f \end{pmatrix} \quad \forall i, \qquad y_1 = 0, \tag{4.4}$$

affected by one degree of freedom, which is very hard to deal with. This remaining degree of freedom is, indeed, due to the fact that, when projecting a scene into an image, the scale of the scene is loss. This phenomenon in well explained by Figure 4.3a, where two different cameras configuration lead to the same measurements v_i.

Usually the problem is solved by arbitrarily fixing a second parameters, for instance $y_2 = 10$, still remembering that the scale of the solved problem is not true.

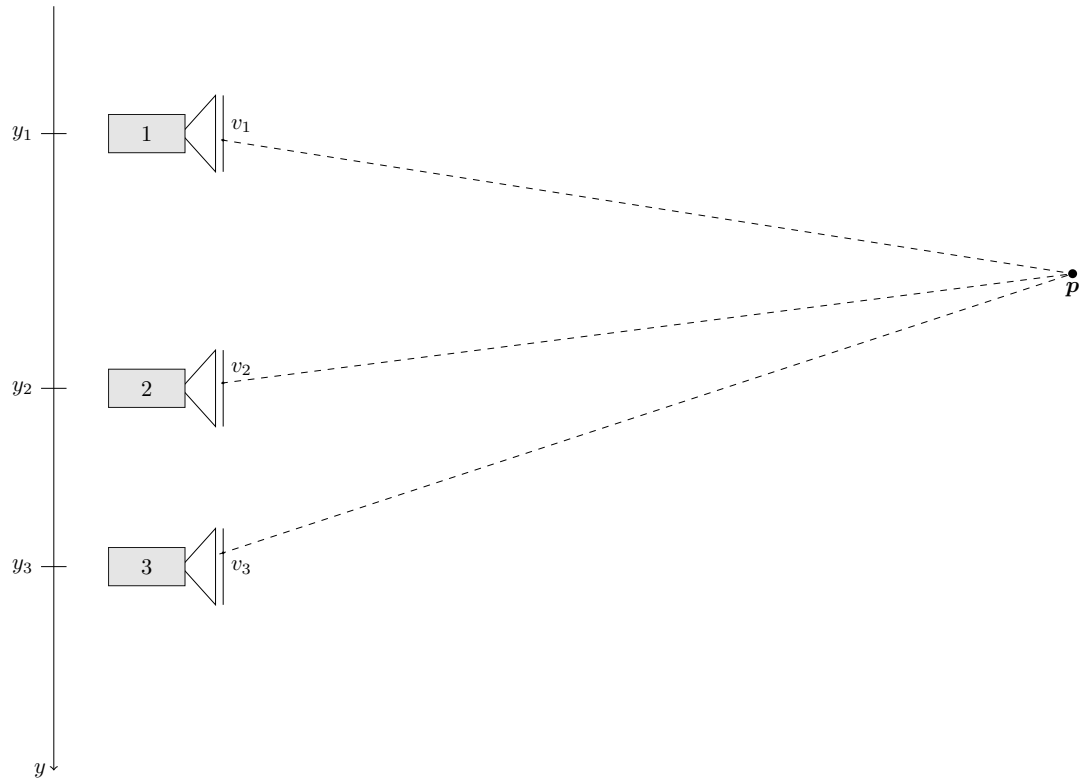


Figure 4.2: A camera is free to move on the y axis and is pointing a point $\boldsymbol{p}$. The SfM algorithm aims at reconstructing the camera motion, i.e., the values of y_1, y_2 and y_3 and the position of the point $\boldsymbol{p}$ simply using the informations stored in v_1, v_2 and v_3.

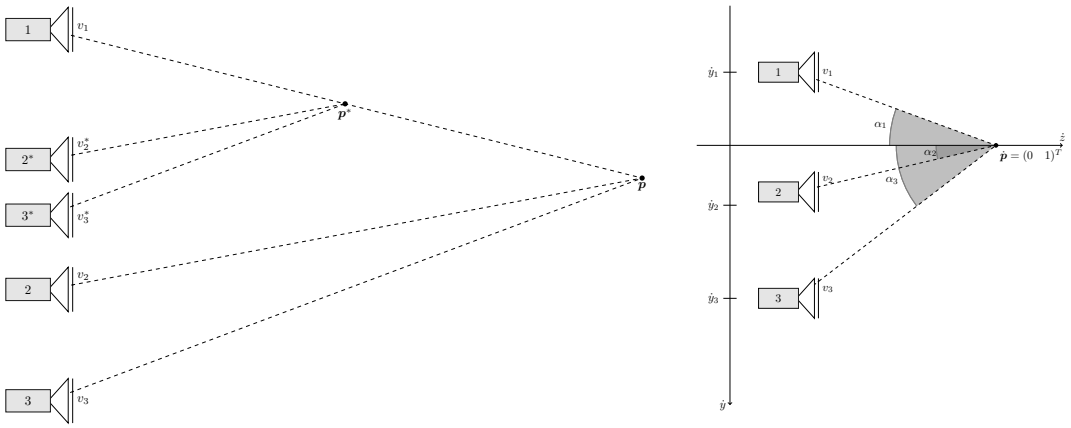
An alternative and smarter method to solve the problem consists in defining a complementary equations system by normalizing the problem in order to force $z = 1$ and setting $y = 0$. The problem so defined assumes the form

$$\begin{pmatrix} \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \dot{y}_i \\ 0 \end{pmatrix} + \dot{\lambda}_i \begin{pmatrix} \dot{v}_i \\ \dot{f} \end{pmatrix} \quad \forall i, \qquad \dot{y} = 0, \quad \dot{z} = 1, \tag{4.5}$$

Since the two methods are equivalent, the second one is preferred because it explicits the scale losing by defining a new problem which is *dimensionless*. In particular, the dimensionless solutions of the problem is able to find the measurements of the angles which define the system configurations instead of the classical euclidian coordinates. In particular each $\dot{y}_i$ will be related to the angles α_i which the various epipolar lines define with the horizontal axis, as in

$$\tan \alpha_i = \dot{y}_i \quad \forall i. \tag{4.6}$$

Due to its linearity, the last system can be easily solved using simple methods such as the Least Square Error (LSE) or similar approaches. Moreover, the dimensioned solution of the problem can



(a) Scale loss example

(b) Dimensionless formulation

Figure 4.3: The scale loss in Structure from Motion occurs because of the projection on the images. (b) shows two different configurations of camera positions and interesting point, i.e. $1-2-3-\boldsymbol{p}$ and $1^*-2^*-3^*-\boldsymbol{p}^*$ which produce the same measurements u_1, u_2 and u_3. Note that the angles defining the structure are invariant to the scale. (b) shows a dimensionless reformulation of the problem in which the equations are normalized in order to have $\dot{\boldsymbol{p}} = (0 \quad 1)$. In this reformulations, the $\dot{y}_i$ parameters are related only to angles α_i.

be easily computed considering the fact that the two solutions differ only by a scale factor d. To compute d something measurable in the scene is needed. If a measures l is available, d is simply given by the ratio between l and its dimensionless value $\dot{l}$. For instance, considering our example in Figure 4.3, if the measure of $l = y_1 - y_2$ is known, then d is given by

$$d = \frac{l}{\dot{y}_1 - \dot{y}_2}.$$

The solution of defining a dimensionless equivalent problem is also applicable to the Mono-SLAM algorithm, as shown in the next Chapter.

4.3 The Mono-SLAM algorithm

The Mono-SLAM approach is a method similar to SfM but implemented using a stochastic approach which allows it to be executed in real time, since it does not elaborate all the available data at the same time, as SfM does, but updates the informations every time a new frame is available. In simpler words, the algorithm does not expect to know exactly the position of the interesting point in the scene, but is able to find a locus of possible points in which the interested point should lies from the measurements received. By intersecting the previous locus of point with the locus coming from new measurements, the algorithm is able to reduce that locus until it becomes as small as a single point. A similar procedure is perform to obtain the camera positions at every frame. In this discussion, the same simplification done in the previous Section are used, i.e., the camera is allowed to translate only along the y-axis and no rotation are allowed. The Mono-SLAM algorithm intends to evaluate both camera trajectory and the position of the interesting point $\boldsymbol{p}$.

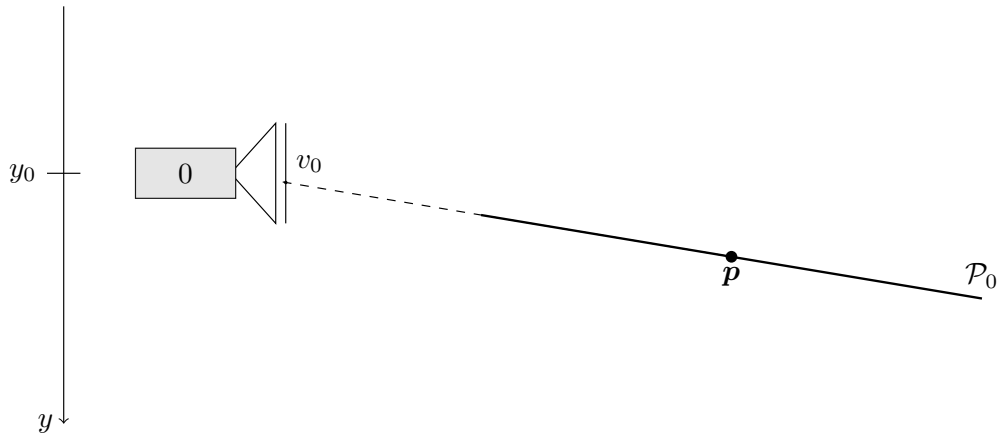


Figure 4.4: When the first frame is available, the algorithm sets the camera position y_0 at a predefined position in the space and computes the epipolar line related to the measurement v_0. A subset of the epipolar line (enhanced in black) is the region in which the point $\boldsymbol{p}$ should lies.

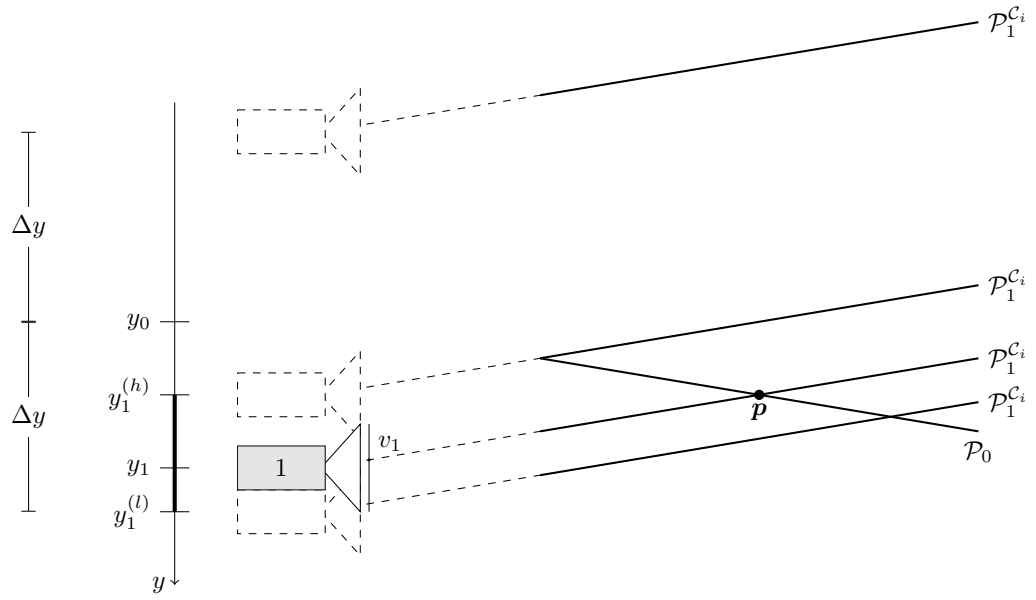
When the first frame is available, the algorithm sets the camera to a predefined initial position, that is usually the center origin of the reference frame, i.e., $y_0 = 0$, and computes a first probabililty region in which the point should be located starting from the information given by the first measurement v_0, as shown in Figure 4.4. In this simplified version of the problem, the probability region, labeled with $\mathcal{P}_0$, is a subset of the epipolar line of v_0.

When the second measurement v_1 occurs, the algorithm computes the new probability region $\mathcal{P}_1^{\mathcal{C}}$ defined in the current camera position, which is unknown. Since the algorithm has a notion of the maximum translation Δy for which the camera can move every frame, it intersects the locus of the $\mathcal{P}_1^{\mathcal{C}_{y_i}}$ generated by all the possible camera positions and the set $\mathcal{P}_0$ previously defined, as in

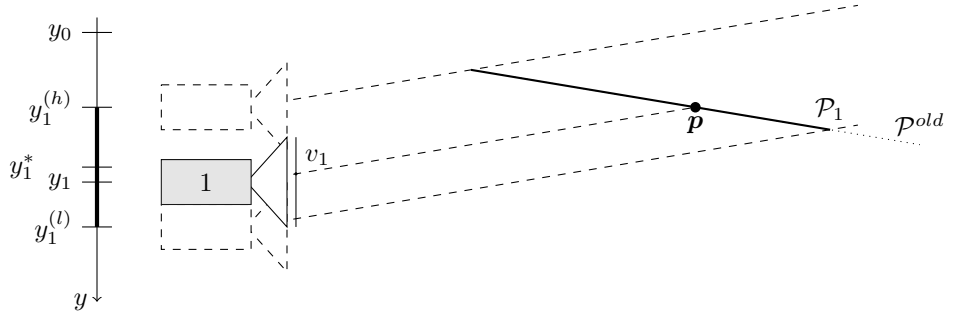
$$\mathcal{P}_1 = \mathcal{P}_0 \bigcap \left(\bigcup_{y_i \in \mathcal{Y}_0^+} \mathcal{P}_1^{\mathcal{C}_{y_i}} \right), \qquad \mathcal{Y}_0^+ = [y_0 - \Delta y, y_0 + \Delta y] \,.$$

The intersection $\mathcal{P}_1$ of the two sets reduces the possible location of the point $\boldsymbol{p}$ and establishes the set $\mathcal{Y}_1$ of possible location of the real position of the camera. The center y_1^* of $\mathcal{Y}_1$ is the best possible measurement of the real camera position y_1. What just discussed is schematized in Figure 4.5. Each step of the algorithm is so able to better define the position $\boldsymbol{p}$ in the environment, and so also the real position of the camera. For this reason, in this algorithm, the first positions of the camera trajectory are often less precise of the last ones, when more informations are available. Note that this problem does not afflict standard SfM solutions, since they analyze all information at the same time. On the other hand, this algorithm can be performed in real time and used with real robot applications.

Please note that this algorithm is subjected to scale loss too. The scale dimension is, in fact, forced by the ration between the parameter Δy and the depth of the point $\boldsymbol{p}$. Likewise seen in the previous Section, the problem can be normalized in order to obtain a dimensionless equivalent formulation.



(a) Prediction



(b) Update

Figure 4.5: Mono-SLAM main iteration. When the new measurement v_1 occurs, the algorithm compute the probability region $\mathcal{P}_1^{\mathcal{C}_{y_i}}$ for all the possible camera positions defined the maximum translation Δy (a). The intersection of the locus of $\mathcal{P}_1^{\mathcal{C}_{y_i}}$ and the old $\mathcal{P}_0^{\mathcal{C}}$ computed at the previous step gives the new probability region $\mathcal{P}_1$ and the region $\mathcal{Y}_1 = \left[y_1^l, y_1^h\right]$ in which the real camera position y_1 lies (b).

4.4 Real application issues and final remarks

In this Chapter three examples of different methods capable to solve the reconstruction problem have been discussed from a very simplified point of view to allow the reader to better understand the complexity of the the problem and the main issues afflicting it. However, it is still necessary to come out from the ideal point of view and examine the problem also when these solutions have to face with real system. Before to introduce the last considerations, please note the introduction of the third dimension in the discussed problem does not lead to new theoretical issues and complicates the notation, whereby the examples still considers a 2D projections of the 3D environment.

First of all, remember that the model of the camera introduces some errors in the measurements, leading to the fact that the epipolar lines can not be established perfectly but are afflicts from uncertainty. This fact is schematized in Figure 4.6, which shows the fact that the region where the point lies is defined by the angle θ, which here is labeled as angular resolution of the camera. This error is due both to the model used for the camera and both from the pixel dimensions. Note also that, if the model is perfect or at least good, i.e., the errors due to the model are smaller then a pixel, the angular resolution can not decrease below a value due to the resolution in pixel of the camera.

Due to angular uncertainty given by θ, several considerations previously done fails. For instance, when considering this uncertainty in a ideal stereo system, the position of the triangulated point can not be measured with accuracy, moreover, due to this uncertainty, if the depth of the point is too far from the system, it is not measurable at all due to depth resolution propagation. In addition, a real stereo system is also affected by several additional uncertainties due to the fact that is very difficult to respect the geometrical alignments required. For this reason, stereo system are not so accurate as it could seems hence are not used for applications requiring high accuracy. On the other side, SfM approaches (including Mono-SLAM) are able to deal with this kind of uncertainty since they reduce it by fusing informations coming from more then two images, usually tens or hundreds.

A second very import source of issues affecting reconstruction systems based on vision is the matching phase which aims at finding the correspondences between two different images. Firstly, this procedure increases the angle uncertainty especially if the interesting point is viewed from distant views. Moreover, the image processing phase requires an huge amount of computational resources and takes a significant amount of time, especially if the algorithm uses very robust descriptors. This is a significant issue in the Mono-SLAM application, which have to satisfy real time constraints. Finally, the mismatches are probably the most important problem that this approaches have to deal with, in fact, a single mismatch can damage the complete reconstruction of the scene. Mismatches afflict especially the Mono-SLAM algorithm, in which very simple and not robust descriptors are used for real time constraints. Luckily, a particular technique, labeled as *active search*, is used both to avoid mismatch and to speed up the matching phase. It consists on reducing the size of the windows in which the interesting point should be located by using the informations available from the previous measurements. However, active search is not able to completely avoid mismatches, hence some sophisticated methods must be implemented to find and erase mismatches when they occur. All this concepts will be deepen in the next Chapters.

Finally, the camera motions itself is source of issues in this algorithm for two main reasons which occur when the camera is moving quickly. First of all, the camera can move in such a way that two consecutive frames do no overlap, leading to the impossibility to find matches and to reconstruct the scene, unfortunately this problem can not be handle in any way, but reduced if the camera has a wide field of view. Moreover, if the camera is moving too quickly, the captured images could be blurred due to motion blur and the matches can not be establish. During the work of this Master Thesis a novel approach able to handle the motion blur has been developed.

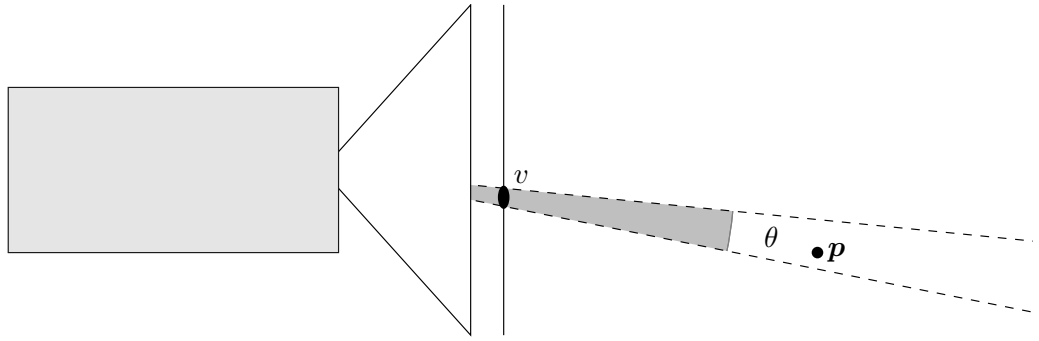


Figure 4.6: Angular resolution of the camera: the epipolar line is affected by an uncertainty defined by the angle θ.

Chapter 5

The Mono-SLAM algorithm

The impossible often has a kind of integrity which the merely improbable lacks

Douglas Adams

In this Chapter the theoretical analysis of the Mono-SLAM algorithm proposed by Davison et al. [1, 9, 21, 22] is presented. The majority of the results here presented are taken from Davison et al. papers and from the MSc thesis "An Analysis of Visual Mono-SLAM" [36, cap. 4], which provides a good analysis of the algorithm but affected by some mistakes. However, the mathematical formulation here presented is built to be more effecting by using matrix formulation each time it is possible. Moreover, original results added by this work will be highlight.

This Chapter is organized as follows: Section 5.1 introduces to Mono-SLAM idea and its benefits and limits. Section 5.2 explains the main idea of the Kalman Filter and the Extended Kalman Filter. Section 5.3 illustrates the composition of the state of the filter. In Sections 5.4 and 5.5 a mathematical analysis of the algorithm is presented, considering the main steps of the Extended Kalman Filter, i.e., prediction and update. After that, Section 5.6 provides a theoretical analysis and comparison of the two encoding techniques of interesting points used in the algorithm and illustrates the initialization phase of the interesting points. Finally, in Section 5.7 a version of the original algorithm dimensionally unconstrained.

5.1 An introduction to Mono-SLAM

Mono-SLAM is an algorithm proposed by Davison et al. which provides one of the first solution to Simuolaneus Localization And Mapping (SLAM) problem using only monocular Vision as sensor. It is implemented by means of the well know Extended Kalman Filter (EKF) algorithm.

As in classical probabilistic approaches based on laser scanner sensors, the robot pose is described as a stochastic variable with Gaussian distribution, while the map of the environment is sparse. In other words, the environment is described by means of a small number of *features*, i.e., measurable geometrical entities, for instance points as in classical Mono-SLAM or straight lines as in [10]. Also features are described as stochastic variables.

The main limitation of all monocular Vision system is the loss of the spatial scale of the scene, caused by the fact that monocular camera works with affine geometry (see Chapter 2). In classical

monocular vision approaches, the problem is solved observing an object with known dimension to initialize the scale of the map during the first execution of the algorithm, adding constraints to the system. Furthermore, initializing the scale leads to a more serious issues known as *scale drift*, i.e., if the scale is not continuously corrected by observing objects with known dimensions, it drift in time, leading to mistakes in measurements. However, in its most recent formulations, Mono-SLAM can be executed without this initialization phase, and it is able to reconstruct a coherent map and camera path where only a scale factor parameters is unknown. In this regard, Section 5.7 presents a variant of Mono-SLAM, based on [9], in which the state is formally independent from the scale.

The Mono-SLAM algorithm uses a sparse map to represent the environment. It reduces the amount of memory and computational costs required, avoiding the use of redundant information of the environment, but introduces the problem of choosing the right feature in terms both of matching robustness and geometric map consistence.

Moreover, and this is the most important characteristic of the algorithm, it uses a joint gaussian variable to describe both state and features in the map. This choice improve features matching. In fact, knowing where, in the next frame, the features should be located reduces dramatically computational time in features matching and helps in avoiding outliers [37–40]. On the other side, the bigger the size of the state vector is, the more computation power is required to execute the algorithm.

Finally, Mono-SLAM uses some important solution to speed up. For instance, it predicts the positions of the tracked points in the distorted image, instead of work directly on the distorted image as usually in computer vision application. In this way, all the computational power require to correct the distortion of the whole image is completely avioded

5.2 Kalman Filter

This Section introduces to the *Kalman Filter* (KF) and the *Extended Kalman Filter* (EKF) which is the algorithm at the base of Mono-SLAM. Please note that the aim of this Section is to give to reader a brief introduction of the filter to allow to understand the discussion provided in the following Sections, however, it does not provide a deep analysis on the filter and mathematical proofs. If the reader is interested in deepening this argument, [41] is highly recommended.

The Kalman Filter is a well known and widely used recursive Gaussian filter to estimate the state of continuous linear systems under uncertainty. The system is described by the following equation

$$\begin{aligned} \hat{\boldsymbol{\mu}}_{k+1} &= \mathbf{A}\hat{\boldsymbol{\mu}}_k + \mathbf{B}\boldsymbol{u}_k + \boldsymbol{v}_k, \\ \boldsymbol{y}_k &= \mathbf{C}\hat{\boldsymbol{\mu}}_k + \boldsymbol{w}_k, \end{aligned}$$

where $\hat{\boldsymbol{\mu}}_{k+1}$ is the (unmeasurable) state of the system, $\boldsymbol{u}_k$ is the inputs of the systems and $\boldsymbol{y}_k$ are the outputs of the system at each time k. $\boldsymbol{v}_k$ and $\boldsymbol{w}_k$ are the noises affecting state evolution and measurements respectively, and they are supposed to be joint gaussian signals having zero mean and covariance $\mathbf{V}$ and $\mathbf{W}$ respectively. They are supposed to be uncorrelated to each other.

According to Kalman Filter, the state vector $\hat{\boldsymbol{\mu}}_k$ is modeled by a joint-gaussian distribution with mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$, at any time k. What the state denotes is dependent on the application, of course. In Mono-SLAM, for example, the state will encode the 3D position of the camera in the world coordinate system, its orientation and velocities and the estimated positions of all observed features. The system will be observed at discrete points in time, where the current time is always referred to as k, while the previous time steps are $k-1$, $k-2$ etc. The system can be influenced in each time step by a set of inputs denoted as $\boldsymbol{u}_k$. Furthermore

Algorithm 5.1: Kalman Filter Prediction Step

Input: $\boldsymbol{\mu}_{k|k}$, $\boldsymbol{\Sigma}_{k|k}$, $\boldsymbol{u}_k$
Output: $\boldsymbol{\mu}_{k+1|k}$, $\boldsymbol{\Sigma}_{k+1|k}$

$$\boldsymbol{\mu}_{k+1|k} = \mathbf{A}\boldsymbol{\mu}_{k|k} + \mathbf{B}\boldsymbol{u}_k$$
$$\boldsymbol{\Sigma}_{k+1|k} = \mathbf{A}\boldsymbol{\Sigma}_{k|k}\mathbf{A}^T + \mathbf{V}$$

Algorithm 5.2: Kalman Filter Update Step

Input: $\boldsymbol{\mu}_{k|k-1}$, $\boldsymbol{\Sigma}_{k|k-1}$, $\boldsymbol{z}_k$
Output: $\boldsymbol{\mu}_{k|k}$, $\boldsymbol{\Sigma}_{k|k}$

$$\boldsymbol{e}_k = \boldsymbol{z}_k - \mathbf{C}\hat{\boldsymbol{\mu}}_{k|k-1}$$
$$\mathbf{S}_k = \mathbf{C}\boldsymbol{\Sigma}_{k|k-1}\mathbf{C}^T + \mathbf{W}$$
$$\mathbf{K}_k = \boldsymbol{\Sigma}_{k|k-1}\mathbf{C}^T\mathbf{S}_k^{-1}$$
$$\boldsymbol{\mu}_{k|k} = \boldsymbol{\mu}_{k|k-1} + \mathbf{K}_k\boldsymbol{e}_k$$
$$\boldsymbol{\Sigma}_{k|k} = (\mathbb{I} - \mathbf{K}_k\mathbf{C})\,\boldsymbol{\Sigma}_{k|k-1}$$

it is assumed that some sort of sensor exists which can be used to gain (noisy) measurements $\boldsymbol{y}_k$ about the system at time k. Each time step is structured in two phases. First comes the so called *prediction step* and afterwards the *update step*. The basic idea of the prediction is to estimate into which state the system should be transferred to from estimated state ($\boldsymbol{\mu}_{k-1|k-1}$, $\Sigma_{k-1|k-1}$) if actions at are executed. Once the next state ($\boldsymbol{\mu}_{k|k-1}$, $\Sigma_{k|k-1}$) is predicted, a measurement prediction has to be made. In order to do this, the predicted state $\boldsymbol{\mu}_{k|k-1}$ is used to compute the measurements $\boldsymbol{y}_k$ which are expected, if the system would in fact be in state ($\boldsymbol{\mu}_{k|k-1}$, $\Sigma_{k|k-1}$) . After the execution a measurement $\boldsymbol{z}_k$ is taken and the actual state of the system is compared with the predicted state. Both prediction and measurement influence the new estimated state ($\boldsymbol{\mu}_{k|k}$, $\Sigma_{k|k}$). Algorithms 5.1 and 5.2 shows the the previous description in a more compact way. In Algorithm 5.2, in particular, they are defined three new entities, i.e., the *innovation vector* $\boldsymbol{e}_k$, the *innovation matrix* $\mathbf{S}_k$ and the Kalman gain $\mathbf{K}_k$. The *innovation vector* is the difference between predicted and real measurements, and it represents the new information between the predicted and the real state, similarly, the innovation matrix represents the covariance of the innovation vector. On the other side, the Kalman gain $\mathbf{K}_k$ can be interpreted as a weight how strongly the actual measurement can influence the predicted state, resulting in the updated state.

Note that while the measurements $\boldsymbol{z}_k$ and the actions at can be directly observed, the rest of the system, especially state $\boldsymbol{x}_k$ can not be observed directly, but is estimated through sensor measurements and actions. If the true state could be observed there would be no need for an estimation and thus an EKF. This distinction is depicted in Figure 5.1.

The Extended Kalman Filter extends the basic idea to the original algorithm to non linear systems affected by noise described as

$$\hat{\boldsymbol{\mu}}_{k+1} = g\left(\hat{\boldsymbol{\mu}}_k, \boldsymbol{u}_k, \boldsymbol{v}_k\right),$$
$$\boldsymbol{y}_k = h\left(\hat{\boldsymbol{\mu}}_k, \boldsymbol{w}_k\right),$$

where $g(\cdot)$ and $h(\cdot)$ are nonlinear function called respectively *transition function* and *measurements function*. The EKF system simply linearize this two function at each instant k around the predicted

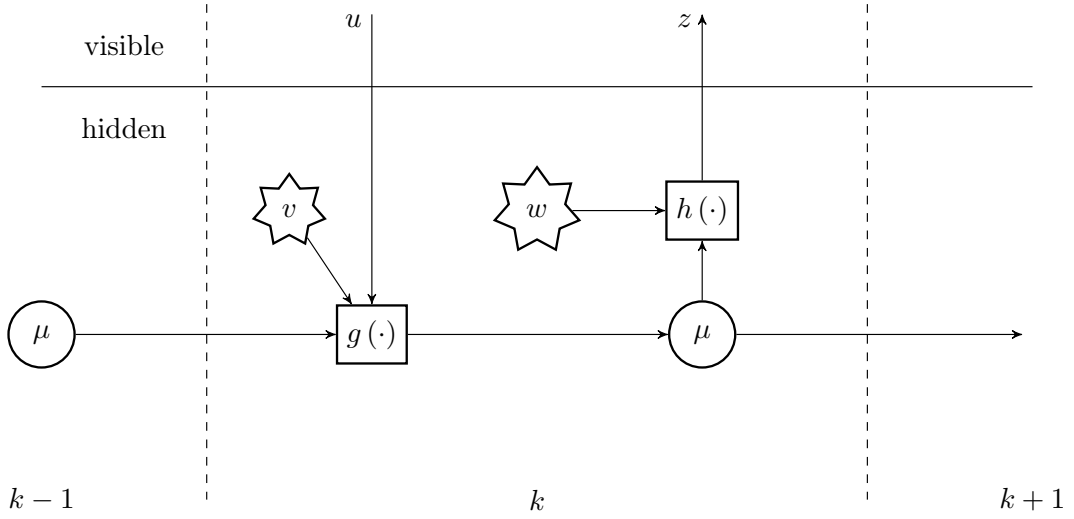


Figure 5.1: Schematic of a system evolution sequence. The states evolves according to function $g(\cdot)$ while the measurements depends from the state throw the function $h(\cdot)$. Both state and measurements are effected by noise (v and w). Note that only the inputs u and the outputs y are observable.

Algorithm 5.3: Extended Kalman Filter Prediction Step

Input: $\boldsymbol{\mu}_{k|k}$, $\boldsymbol{\Sigma}_{k|k}$, $\boldsymbol{u}_k$
Output: $\boldsymbol{\mu}_{k+1|k}$, $\boldsymbol{\Sigma}_{k+1|k}$

$\mathbf{G}_k = \frac{\partial g}{\partial \boldsymbol{\mu}}\Big|_{\boldsymbol{\mu}_{k|k}}$
$\mathbf{F}_k = \frac{\partial g}{\partial \boldsymbol{v}}\Big|_{\boldsymbol{\mu}_{k|k}}$

$\boldsymbol{\mu}_{k+1|k} = g\left(\boldsymbol{\mu}_{k|k}, \boldsymbol{u}_k\right)$
$\boldsymbol{\Sigma}_{k+1|k} = \mathbf{G}_k \boldsymbol{\Sigma}_{k|k} \mathbf{G}_k^T + \mathbf{F}_k \mathbf{V} \mathbf{F}_k^T$

and estimated state and applies the standard Kalman Filter to the linearized system, according to Algorithms 5.3 and 5.4.

Please note that, after the linearization of $g(\cdot)$ and $h(\cdot)$, the EKF basically corresponds to the well-studied Kalman Filter. Due to this fact and its simplicity and efficiency compared to other methods to estimate non-linear systems the EKF is currently one of the most popular approaches in this field. On the other side, linearization introduces some errors that are not modeled by the filter, the more the system is non linear, the bigger these error are, so that the EKF algorithm do not work properly when the system presents an big grade of non linearity.

Algorithm 5.4: Extended Kalman Filter Update Step

Input: $\boldsymbol{\mu}_{k|k-1}$, $\boldsymbol{\Sigma}_{k|k-1}$, $\boldsymbol{z}_k$
Output: $\boldsymbol{\mu}_{k|k}$, $\boldsymbol{\Sigma}_{k|k}$

$\mathbf{H}_k = \frac{\partial h}{\partial \boldsymbol{\mu}}\big|_{\boldsymbol{\mu}_{k|k-1}}$
$\mathbf{M}_k = \frac{\partial h}{\partial \boldsymbol{w}}\big|_{\boldsymbol{\mu}_{k|k-1}}$

$\boldsymbol{e}_k = \boldsymbol{z}_k - h\left(\hat{\boldsymbol{\mu}}_{k|k-1}\right)$
$\mathbf{S}_k = \mathbf{H}_k \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^T + \mathbf{M}_k \mathbf{W} \mathbf{M}_k^T$
$\mathbf{K}_k = \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}$
$\boldsymbol{\mu}_{k|k} = \boldsymbol{\mu}_{k|k-1} + \mathbf{K}_k \boldsymbol{e}_k$
$\boldsymbol{\Sigma}_{k|k} = (\mathbb{I} - \mathbf{K}_k \mathbf{H}_k)\, \boldsymbol{\Sigma}_{k|k-1}$

Note also that notation used in KF and EKF comes from the notation used in probability theory. For example, the state $\boldsymbol{\mu}_{k_1|k_2}$ means the estimation of the state $\boldsymbol{\mu}_{k_1}$ at time k_1 using all information available at time k_2. In the following discussion, the notation is simplified when possible.

5.3 State Representation

The main idea of EKF applied to the monocular visual SLAM problem is to describe the world and the camera positions at any instant by using a random multidimensional variable whose mean represents the "best" estimation of these entities.

Mathematically, the system state $\hat{\boldsymbol{\mu}}_k$, i.e., encapsulating both robot pose and map, is represented at any instant $t = k\Delta t$, where $f = 1/\Delta t$ is the camera frame-rate, as a jointly gaussian variable

$$\hat{\boldsymbol{\mu}}_k \sim \mathrm{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{5.1}$$

having mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. The state vector encapsulates the information about both camera and world state

$$\boldsymbol{\mu}_k = \begin{pmatrix} \boldsymbol{x}_k \\ \boldsymbol{f}_0 \\ \vdots \\ \boldsymbol{f}_m \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{5.2}$$

In particular, the camera is represented using the state vector $\boldsymbol{x}_k$, which contains the information needed to know the motion of the camera reference frame $\mathcal{C}$ with respect to the fixed world reference frame $\mathcal{W}$. On the other hand, the map is represented by a set of distinguishable landmarks, each one described by a state vector $\boldsymbol{f}_i$ containing its 3D position with respect to $\mathcal{W}$. The algorithm assumes the scene as rigid and each landmark as a stationary world point, i.e., the vectors $\boldsymbol{f}_i$ are not dependent from time.

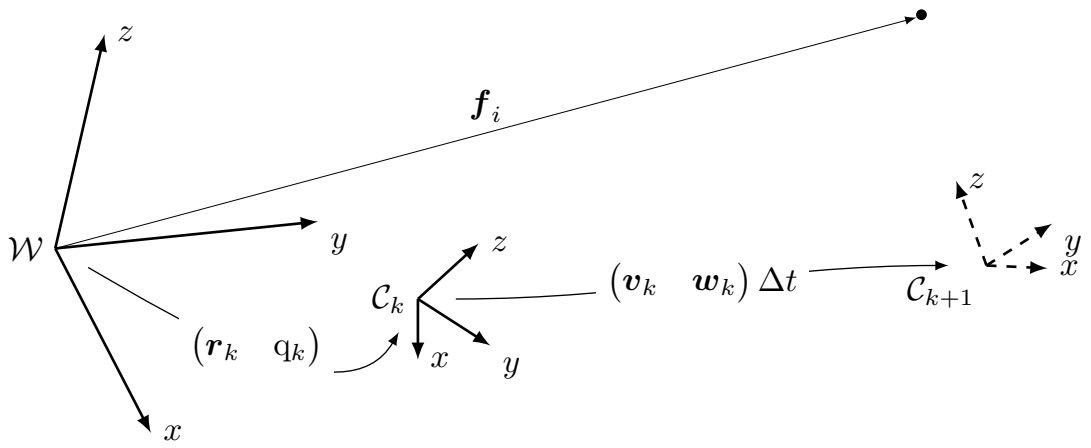


Figure 5.2: State representation in Mono-SLAM. The position and orientation of the camera reference frame $\mathcal{C}$ with respect to the world reference frame $\mathcal{W}$ are encapsulated respectively in $\boldsymbol{r}_k$ and q_k at every instant k, while the next pose of $\mathcal{C}$ are encapsulated in $\boldsymbol{v}_k$ and $\boldsymbol{w}_k$. The position in $\mathcal{W}$ of the interesting point is encapsulated in $\boldsymbol{f}_i$.

5.3.1 Camera Representation

As seen previously, information concerning the camera motion at any instant is encoded in the vector $\boldsymbol{x}_k$ built as

$$\boldsymbol{x}_k = \begin{pmatrix} \boldsymbol{r}_k \\ \mathrm{q}_k \\ \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{pmatrix}, \tag{5.3}$$

where $\boldsymbol{r} = \begin{pmatrix} x_c & y_c & z_c \end{pmatrix}^T$ encapsulates the $3D$ position of the optical center of the camera with respect to the world reference frame $\mathcal{W}$, q is the quaternion specifying the camera orientation relative to $\mathcal{W}$, $\boldsymbol{v} = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}^T$ is the linear speed of the camera optical center along the $\mathcal{W}$ coordinate system and $\boldsymbol{\omega} = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix}^T$ is the angular velocity of the camera relative to $\mathcal{W}$. This representation is better explained in Figure 5.2.

The initial state is chosen such that $\mathcal{C} \equiv \mathcal{W}$, i.e., the two coordinate systems are coincident, and the linear and angular speed are both zero:

$$\boldsymbol{x}_0 = \begin{pmatrix} \boldsymbol{r}_0 \\ \mathrm{q}_0 \\ \boldsymbol{v}_0 \\ \boldsymbol{\omega}_0 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathrm{q}(0) \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix},$$

where $\mathrm{q}(0) = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}^T$ is the identity quaternion, i.e., the quaternion associated to the identity matrix $\mathbb{I}_3$.

While the initial pose of the camera is not affected by error – $\mathcal{C}$ is not supposed to be coincident with $\mathcal{W}$ but it is actually coincident to $\mathcal{W}$ – the initial uncertainty affects only the velocity vectors.

Consequently, the covariance matrix $\mathbf{\Sigma}_0$ of the state is built as

$$\mathbf{\Sigma}_0 = \begin{pmatrix} 0 & 0 \\ 0 & \mathbf{P}_{n_0} \end{pmatrix} \in \mathbb{R}^{13\times 13}, \qquad \mathbf{P}_{n_0} = \begin{pmatrix} \sigma_{v_0}^2 \mathbb{I}_3 & \mathbf{0} \\ \mathbf{0} & \sigma_{\omega_0}^2 \mathbb{I}_3 \end{pmatrix}. \tag{5.4}$$

5.3.2 Features Representation

As seen previously, the Mono-SLAM algorithm assumes the world-map to be static, which implies that each feature can be represented by a time-independent state vector with respect to $\mathcal{W}$. There are in literature two different ways to represent features in the map. The first one is called *euclidian* or *depth* coding, and sometimes it is labeled as *XYZ* coding, while the second is know as *inverse depth* coding. Here the main characteristics of the two encodings will be analyzed, while in Section 5.6 a deeper mathematical analysis will explain the advantages and limits of both.

In this work, symbols $\boldsymbol{y}$ and $\boldsymbol{\psi}$ describe euclidian and inverse depth features respectively, while $\boldsymbol{f}$ refers to a generic feature when knowing the encoding is not relevant.

Euclidian Representation The simpler way to represent feature state is to store its 3D position vector with respect to $\mathcal{W}$:

$$\boldsymbol{y} = \begin{pmatrix} x & y & z \end{pmatrix}^T. \tag{5.5}$$

While euclidian representation is a simple and intuitive feature description, it can hardly manage features with low knowledge of their depth, that are features at the very first observations or features very far from the camera optical center, i.e., *features at infinity*. The usage of this kind of representation implies a preprocessing phase and a delay before adding features in the state vector and can not manage features at infinity because the infinity depth is not representable.

Inverse depth Representation To handle the problem of features initialization and features at infinity, Civera et al. in [42, 43] propose an alternative representation called *inverse depth*, which associates to each feature six parameters as

$$\boldsymbol{\psi} = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} & \theta & \phi & \rho \end{pmatrix}^T, \tag{5.6}$$

where $\boldsymbol{r}^{\mathcal{C}} = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} \end{pmatrix}^T$ specifies the optical center at the first observation of the features, θ and ϕ are respectively *azimuth* and *elevation* of the feature with respect to the image coordinate system and $\rho = 1/d$ is the so-called inverse depth of $\boldsymbol{f}$, where d is the distance of the feature from the optical center at its first observation, as shown in Figure 5.3.

The 3D position of the features in $\mathcal{W}$ can be computed as

$$\boldsymbol{y} = \boldsymbol{r}^{\mathcal{C}} + \frac{1}{\rho}\boldsymbol{\eta}(\theta, \phi), \tag{5.7}$$

$$\boldsymbol{\eta}(\theta, \phi) = \begin{pmatrix} \sin\theta\cos\phi & -\sin\phi & \cos\theta\cos\phi \end{pmatrix}^T, \tag{5.8}$$

as represented in Figure 5.4

The advantage of using inverse depth encoding is that it allows to compute a normalized vector $\boldsymbol{h}$ parallel to $\boldsymbol{y}$ so defined

$$\boldsymbol{h} = \frac{1}{d}\boldsymbol{y} = \rho \boldsymbol{r}^C + \boldsymbol{\eta}(\theta, \phi), \tag{5.9}$$

computable also in cases of features at infinity, i.e., with $\rho \to 0$.

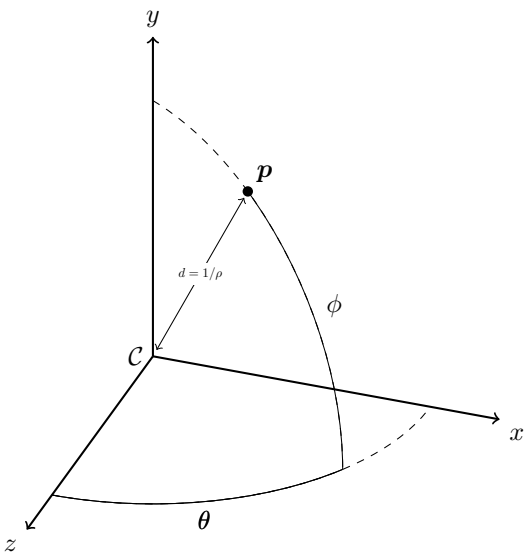


Figure 5.3: Azimuth and elevation in a camera reference frame. For a feature $\boldsymbol{f}$ in the $3D$ space the camera is able to measure azimuth θ and elevation ϕ angles, but from a single image the depth d can not be estimated.

Note that inverse depth features can not represent point with zero depth, while this implies $\rho \to \infty$. This is not a problem because a features with zero or very small depth implies a real point coincident or very near to the camera optical center, i.e., inside the optics of the camera. On the other hand, representing features at infinity allows the algorithm to work well also in open space where this kind of features are very common.

Note also that the dimension of the state vector is strongly related to the number of features and their encoding. Calling $m = m_\rho + m_d$ the number of features in the state, where m_ρ and m_d are the number of inverse depth and euclidian features respectively, the dimension of the state will be $n = 13 + 3m_d + 6m_\rho$.

5.4 Prediction

Prediction phase of EKF aims at *predicting*, i.e., estimating, the evolution of the state vector before the availability of new measurements using a model of the system. In particular in the Mono-SLAM problem, which hypothesizes that features are independent from time, the prediction step involves only the camera state and aims at estimating the pose of the camera in the future time instant $k+1$ according to the information available at the current time instant k using the state evolution model described below.

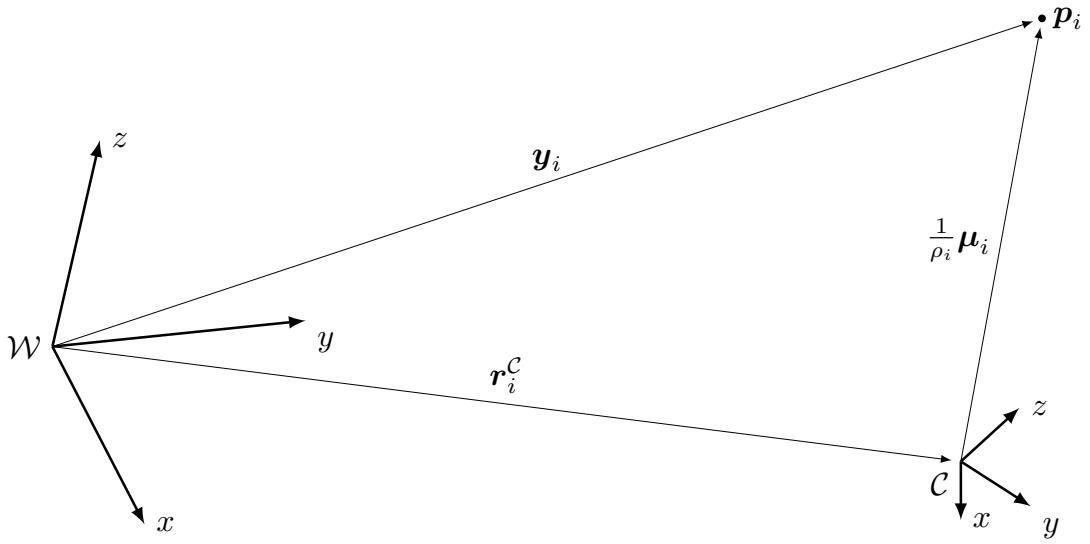


Figure 5.4: Inverse depth representation: the $3D$ position $\boldsymbol{y}_i$ representing the point $\boldsymbol{p}$ in decomposed in the two vectors $\boldsymbol{r}_i^{\mathcal{C}}$ and $d_i\boldsymbol{\mu}_i$, where $d = 1/\rho_i$ and $\boldsymbol{\mu}_i = \boldsymbol{\mu}(\theta_i, \phi_i)$. By using inverse depth encoding, features at infinity, i.e., points very far from the camera which is not able to measure their depths, can be represented.

5.4.1 Prediction model

The camera state transition is modelled using a constant speed model [1, 21]

$$\boldsymbol{x}_{k+1} = \boldsymbol{g}_v(\boldsymbol{x}_k) = \begin{pmatrix} \boldsymbol{r}_k + (\boldsymbol{v}_k + \boldsymbol{V}_k)\Delta t \\ \mathrm{q}_k \times \mathrm{quat}\left((\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t\right) \\ \boldsymbol{v}_k + \boldsymbol{V}_k \\ \boldsymbol{\omega}_k + \boldsymbol{\Omega}_k \end{pmatrix} \tag{5.10}$$

where $\boldsymbol{V}_k$ and $\boldsymbol{\Omega}_k$ are the noise vectors affecting respectively linear and angular speed, $\mathrm{q}_k \times \mathrm{quat}\left((\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t\right)$ represents the quaternion product, as in equation (5.11), between q_k and $\mathrm{quat}\left((\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t\right)$, which is the quaternion corresponding to the rotation $(\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k)\Delta t$ in axis-angle representation.

Given two quaternions q^a and q^a, the (extern) product (non commutative) $\mathrm{q}^a\mathrm{q}^b = \mathrm{q}^a \times \mathrm{q}^b$ is defined as

$$\begin{aligned} \mathrm{q}^a\mathrm{q}^b = \mathrm{q}^a \times \mathrm{q}^b &= \begin{pmatrix} q_r^a q_r^b + \boldsymbol{q}_v^a \cdot \boldsymbol{q}_v^b \\ q_r^a \boldsymbol{q}_v^b + q_r^b \boldsymbol{q}_v^a + \boldsymbol{q}_v^a \times \boldsymbol{q}_v^b \end{pmatrix} \\ &= \begin{pmatrix} q_r^a & -\boldsymbol{q}_v^{a\,T} \\ \boldsymbol{q}_v^a & q_r^a \mathbb{I}_3 + \mathbf{S}\left(\boldsymbol{q}_v^a\right) \end{pmatrix} \mathrm{q}^b = \boldsymbol{\Upsilon}(\mathrm{q}^a)\mathrm{q}^b \qquad (5.11a) \\ &= \begin{pmatrix} q_r^b & -\boldsymbol{q}_v^{b\,T} \\ \boldsymbol{q}_v^b & q_r^b \mathbb{I}_3 - \mathbf{S}\left(\boldsymbol{q}_v^b\right) \end{pmatrix} \mathrm{q}^a = \bar{\boldsymbol{\Upsilon}}(\mathrm{q}^b)\mathrm{q}^a, \qquad (5.11b) \end{aligned}$$

where q_r and $\boldsymbol{q}_v$ are respectively the real and vectorial part of quaternion q, operations $\boldsymbol{q}_v^a \cdot \boldsymbol{q}_v^b$ and $\boldsymbol{q}_v^a \times \boldsymbol{q}_v^b$ denotes respectively scalar and extern products between vectors $\boldsymbol{q}_v^a$ and $\boldsymbol{q}_v^b$, and $\mathbf{S}(\boldsymbol{v})$ denotes the skew-matrix

$$\mathbf{S}(\boldsymbol{v}) = \begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix} \tag{5.12}$$

associated to the vector $\boldsymbol{v} = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}^T$.

Given a vector $\boldsymbol{\alpha}$ representing a rotation in 3D space, to the quaternion $\text{quat}(\boldsymbol{\alpha})$ corresponding to this rotation, it is necessary remember that the vector in axis-angle rotation representation is given by $\boldsymbol{\alpha} = \langle \boldsymbol{a}, \alpha \rangle$, where $\alpha = \|\boldsymbol{\alpha}\|$ is the rotation angle and $\boldsymbol{a} = \boldsymbol{\alpha}/\alpha$ is the versor representing the rotation axis. The rotation quaternion is now simple to compute, as in the following equation

$$\text{quat}(\boldsymbol{\alpha}) = \begin{pmatrix} \cos\frac{\alpha}{2} \\ \boldsymbol{a}\sin\frac{\alpha}{2} \end{pmatrix} = \begin{pmatrix} \cos\frac{\|\boldsymbol{\alpha}\|}{2} \\ \frac{\boldsymbol{\alpha}}{\|\boldsymbol{\alpha}\|}\sin\frac{\|\boldsymbol{\alpha}\|}{2} \end{pmatrix}. \tag{5.13}$$

The world map, i.e., features, is supposed to be static, so there are no update for that part of the state. The complete transition function is

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{g}\left(\boldsymbol{\mu}_k\right) = \begin{pmatrix} \boldsymbol{g}_v\left(\boldsymbol{x}_k\right) \\ \boldsymbol{f}_0 \\ \vdots \\ \boldsymbol{f}_m \end{pmatrix}. \tag{5.14}$$

Since the noise affecting the state is not measurable, the prediction $\boldsymbol{x}_{k+1|k}$ in the real implementation of the algorithm is

$$\boldsymbol{x}_{k+1|k} = \begin{pmatrix} \boldsymbol{r}_{k|k} + \boldsymbol{v}_{k|k}\Delta t \\ \text{q}_{k|k} \times \text{quat}\left(\boldsymbol{\omega}_{k|k}\Delta t\right) \\ \boldsymbol{v}_{k|k} \\ \boldsymbol{\omega}_{k|k} \end{pmatrix}, \tag{5.15}$$

and

$$\boldsymbol{\mu}_{k+1|k} = \begin{pmatrix} \boldsymbol{x}_{k+1|k}^T & \boldsymbol{f}_0^T & \cdots & \boldsymbol{f}_m^T \end{pmatrix}^T. \tag{5.16}$$

Note that, in many application of the Mono-SLAM algorithm, the commands of the camera motions could be measurable, e.g., with a camera located on a mobile robot. In this scenario the information about commands would have a strong impact on the update function of the system. Despite that, the original algorithm aims at handling a very unconstrained problem in which noting except video information are available, so that no information about commands of the motion are considered. This means that the motion commands need to be treated as unknown (stochastic) variables that are encapsulated in the noise vector $\boldsymbol{n}_k$ affecting the state.

5.4.2 Physical interpretation of noise

While the camera is not supposed to move with constant speed motion, it is important to understand the physical meaning of the noise affecting the state

$$\boldsymbol{n}_k = \begin{pmatrix} \boldsymbol{V}_k \\ \boldsymbol{\Omega}_k \end{pmatrix}.$$

At any instant, the camera motion is affected by unknown $\boldsymbol{a}_k$ and $\boldsymbol{\alpha}_k$ accelerations respectively linear and rotational. In this terms the update of the speed of the can be written as

$$\begin{pmatrix} \boldsymbol{v}_{k+1} \\ \boldsymbol{\omega}_{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{pmatrix} + \begin{pmatrix} \boldsymbol{a}_k \\ \boldsymbol{\alpha}_k \end{pmatrix} \Delta t = \begin{pmatrix} \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{pmatrix} + \begin{pmatrix} \boldsymbol{V}_k \\ \boldsymbol{\Omega}_k \end{pmatrix}.$$

If the motion of $\mathcal{C}$ is smooth enough, the variation of velocity vectors can be approximated as stochastic variables having zero mean and variance given by the maximum value of the acceleration vectors $a_{max} = \max\|\boldsymbol{a}\|$ and $\alpha_{max} = \max\|\boldsymbol{\alpha}\|$ as

$$\mathbf{P}_n = \begin{pmatrix} \sigma_v^2 \mathbb{I}_3 & \mathbf{0} \\ \mathbf{0} & \sigma_\omega^2 \mathbb{I}_3 \end{pmatrix}, \tag{5.17}$$

with

$$\sigma_v = a_{max}\Delta t \qquad \sigma_\omega = \alpha_{max}\Delta t. \tag{5.18}$$

5.4.3 Uncertainty propagation

In order to compute the prediction of the covariance matrix of the state, using

$$\boldsymbol{\Sigma}_{k+1|k} = \mathbf{G}_k \boldsymbol{\Sigma}_{k|k} \mathbf{G}_k^T + \mathbf{Q}_k,$$

EKF requires the matrices $\mathbf{G}_k$, i.e., the linearization of the system in the actual position $\boldsymbol{\mu}_k$, and $\mathbf{Q}_k$, that is the additional noise covariance matrix.

System Linearization

$\mathbf{G}_k$ is given by the definition of linearization

$$\mathbf{G}_k = \frac{\partial \boldsymbol{g}\left(\boldsymbol{\mu}_k\right)}{\partial \boldsymbol{\mu}_k}\bigg|_{\boldsymbol{\mu}_k = \boldsymbol{\mu}_{k|k}} = \begin{pmatrix} \frac{\partial \boldsymbol{g}_v(\boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} & 0 \\ 0 & \mathbb{I} \end{pmatrix}\bigg|_{\boldsymbol{x}_k = \boldsymbol{x}_{k|k}}, \tag{5.19}$$

where

$$\frac{\partial \boldsymbol{g}_v\left(\boldsymbol{x}_k\right)}{\partial \boldsymbol{x}_k} = \begin{pmatrix} \mathbb{I}_3 & 0 & \mathbb{I}_3\Delta t & 0 \\ 0 & \frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{q}_k} & 0 & \frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\omega}_k} \\ 0 & 0 & \mathbb{I}_3 & 0 \\ 0 & 0 & 0 & \mathbb{I}_3 \end{pmatrix} \in \mathbb{R}^{13\times 13}. \tag{5.20}$$

The two jacobians related to quaternions are a little bit difficult to compute. From now, it is used $\boldsymbol{\omega} = \boldsymbol{\omega}_k$ and $\mathrm{h} = \mathrm{quat}\left(\left(\boldsymbol{\omega}_k + \boldsymbol{\Omega}_k\right)\Delta t\right)$ to simplify the notation.

Jacobian $\partial \mathrm{q}_{k+1}/\partial \mathrm{q}_k$ can be computed from equation (5.11b)

$$\frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{q}_k} = \frac{\partial}{\partial \mathrm{q}_k}\left(\mathrm{q}_k \times \mathrm{h}\right) = \bar{\boldsymbol{\Upsilon}}(\mathrm{h}), \tag{5.21}$$

while $\partial \mathrm{q}_k(k+1)/\partial \boldsymbol{\omega}$ is so computed

$$\frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\omega}} = \frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{h}} \frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}}, \tag{5.22}$$

where jacobian $\partial \mathrm{q}_{k+1}/\partial \mathrm{h}$ comes directly from equation (5.11a)

$$\frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{h}} = \frac{\partial}{\partial \mathrm{h}}\left(\mathrm{q}_k \times \mathrm{h}\right) = \boldsymbol{\Upsilon}(\mathrm{q}_{k+1}). \tag{5.23}$$

and

$$\frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}} = \Delta t \frac{\partial \mathrm{h}}{\partial \left(\boldsymbol{\omega}\Delta t\right)} = \Delta t \frac{\partial \mathrm{q}\left(\boldsymbol{\alpha}\right)}{\partial \boldsymbol{\alpha}}$$

is a little bit hardly to be compute. First of all, note that, from equation (5.13) if follows

$$\frac{\partial \mathrm{quat}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{\alpha}} \cos \frac{\|\boldsymbol{\alpha}\|}{2} \\ \frac{\partial}{\partial \boldsymbol{\alpha}} \left(\boldsymbol{\alpha} \frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} \right) \end{pmatrix}. \tag{5.24}$$

The two entries of the defined jacobian are given by

$$\frac{\partial}{\partial \boldsymbol{\alpha}} \cos \frac{\|\boldsymbol{\alpha}\|}{2} = \frac{\partial \cos \frac{\|\boldsymbol{\alpha}\|}{2}}{\partial \|\boldsymbol{\alpha}\|} \frac{\partial \|\boldsymbol{\alpha}\|}{\partial \boldsymbol{\alpha}} = -\frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{2\|\boldsymbol{\alpha}\|} \boldsymbol{\alpha}^T, \tag{5.25}$$

and

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\alpha}} \left(\boldsymbol{\alpha} \frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} \right) &= \frac{\partial \boldsymbol{\alpha}}{\partial \boldsymbol{\alpha}} \frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} + \boldsymbol{\alpha} \frac{\partial}{\partial \|\boldsymbol{\alpha}\|} \left(\frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} \right) \frac{\partial \|\boldsymbol{\alpha}\|}{\partial \boldsymbol{\alpha}} \\ &= \mathbb{I}_3 \frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} + \boldsymbol{\alpha} \frac{\frac{\|\boldsymbol{\alpha}\|}{2} \cos \frac{\|\boldsymbol{\alpha}\|}{2} - \sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|^2} \frac{\boldsymbol{\alpha}^T}{\|\boldsymbol{\alpha}\|} \\ &= \mathbb{I}_3 \frac{\sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|} + \boldsymbol{\alpha}\boldsymbol{\alpha}^T \frac{\frac{\|\boldsymbol{\alpha}\|}{2} \cos \frac{\|\boldsymbol{\alpha}\|}{2} - \sin \frac{\|\boldsymbol{\alpha}\|}{2}}{\|\boldsymbol{\alpha}\|^3}, \end{aligned} \tag{5.26}$$

finally leading to equation (5.27) after substituting $\boldsymbol{\alpha}$ with $\boldsymbol{\omega}\Delta t$ and simplificating the various terms Δt:

$$\frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}} = \begin{pmatrix} -\frac{\Delta t}{2\|\boldsymbol{\omega}\|} \sin \frac{\|\boldsymbol{\omega}\|\Delta t}{2} \boldsymbol{\omega}^T \\ \mathbb{I}_3 \frac{\sin \frac{\|\boldsymbol{\omega}\|\Delta t}{2}}{\|\boldsymbol{\omega}\|} + \frac{\frac{\|\boldsymbol{\omega}\|\Delta t}{2} \cos \frac{\|\boldsymbol{\omega}\|\Delta t}{2} - \sin \frac{\|\boldsymbol{\omega}\|\Delta t}{2}}{\|\boldsymbol{\omega}\|^3} \boldsymbol{\omega}\boldsymbol{\omega}^T \end{pmatrix}. \tag{5.27}$$

In practical cases, the rotation speed $\boldsymbol{w}$ could be a zero vector, e.g., after the initialization of the state. For this reason equation (5.27) can not be used in this form, as it presents 0/0 operations. In the references, the practical problem is solved initializing $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ with a small $\epsilon > 0$ instead of exactly 0 value. On the contrary, here it is proposed an alternative form of equation (5.27) which handles $\boldsymbol{\omega} = 0$ cases:

$$\frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}} = \frac{\Delta t}{2} \begin{pmatrix} -\sin \frac{\|\boldsymbol{\omega}\|\Delta t}{2} \boldsymbol{n}_\omega^T \\ \mathbb{I}_3 \operatorname{Sinc} \frac{\|\boldsymbol{\omega}\|\Delta t}{2} + \left(\cos \frac{\|\boldsymbol{\omega}\|\Delta t}{2} - \operatorname{Sinc} \frac{\|\boldsymbol{\omega}\|\Delta t}{2} \right) \boldsymbol{n}_\omega \boldsymbol{n}_\omega^T \end{pmatrix}, \tag{5.28}$$

where $\operatorname{Sinc}(x)$ is the function defined as

$$\operatorname{Sinc} x = \begin{cases} \frac{\sin x}{x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0 \end{cases}, \tag{5.29}$$

and $\boldsymbol{n}_\omega = \boldsymbol{\omega}/\|\boldsymbol{\omega}\|$ is the versor parallel to $\boldsymbol{\omega}$. Note that $\boldsymbol{n}_0$ is not well defined, but in this case the factors that multiply it are zero. In this particular case, the jacobian is still computable and the results of the computation is

$$\left. \frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}} \right|_{\boldsymbol{w}=\mathbf{0}} = \frac{\Delta t}{2} \begin{pmatrix} \mathbf{0}^T \\ \mathbb{I}_3 \end{pmatrix}.$$

Noise propagation

Since the noise $\boldsymbol{n}_k$ affects only the camera state vectors $\boldsymbol{x}_k$, the covariance matrix of the additional noise $\mathbf{Q}_k$ can be written as

$$\mathbf{Q}_k = \begin{pmatrix} \mathbf{Q}'_k & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{n \times n}, \tag{5.30}$$

with

$$\mathbf{Q}'_k = \frac{\partial \boldsymbol{g}_v}{\partial \boldsymbol{n}} \mathbf{P}_n \frac{\partial \boldsymbol{g}_v}{\partial \boldsymbol{n}}^T \in \mathbb{R}^{13 \times 13}. \tag{5.31}$$

The jacobian $\frac{\partial \boldsymbol{g}_v}{\partial \boldsymbol{n}}$ can be decomposed as

$$\frac{\partial \boldsymbol{g}_v}{\partial \boldsymbol{n}} = \begin{pmatrix} \mathbb{I}_3 \Delta t & 0 \\ 0 & \frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\Omega}_k} \\ \mathbb{I}_3 & 0 \\ 0 & \mathbb{I}_3 \end{pmatrix} \tag{5.32}$$

where it is simple to demonstrate that

$$\frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\Omega}_k} = \frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\omega}_k} \in \mathbb{R}^{13 \times 6}. \tag{5.33}$$

computed in equation (5.22).

When algorithm is implemented in code, computational resources can be saved considering that the matrix $\partial \boldsymbol{g}_v / \partial \boldsymbol{n}$ coincides with the second part of the matrix $\partial \boldsymbol{g}_v(\boldsymbol{x}_k) / \partial \boldsymbol{x}_k$ defined in equation (5.20).

The system evolution model is almost linear except in the quaternion component update, as seen also from the two jacobian matrices defined in equation (5.20) and (5.32), which are unavailable. The non linearity is caused by the impossibility to represent a rotation in 3D space as a linear function. In particular, rotations are a very complex and difficult field in 3D geometry application and standard solutions to represent them do not exist in literature. While in this work quaternions are used for their efficience in terms of computational resources required, several alternative solutions to handle rotation exist. For more information about the very interesting topic of representing rotation in 3D space, a very interesting readings are [44–46].

5.5 Update Step

The update step of EKF aims at refining the state vector by means of the innovation vector, i.e., the difference between the measurements available and their estimations. As the prediction is based on the model of the state evolution, the update phase is built on the model of the measurement function. In particular in the Mono-SLAM algorithm, the EKF compares the measurements of the positions in pixel of each feature in the new frame with their estimated positions obtained by the prediction of the state.

5.5.1 Measurement Function

As seen in Chapter 2 and according to camera models, an observed point on the image sensor defines an epipolar line in the scene, represented by a directional vector $\boldsymbol{h}^{\mathcal{C}} = \begin{pmatrix} h_x & h_y & h_z \end{pmatrix}^T$ in $\mathcal{C}$.

The vector $\boldsymbol{h}^{\mathcal{C}}$ depends both from the camera state $\boldsymbol{x}_k$ and from the feature it refers to, and it presents two different forms according to the feature coding considered $\boldsymbol{f} = \{\boldsymbol{y}, \boldsymbol{\psi}\}$, in particular

$$\boldsymbol{h}^{\mathcal{C}}(\boldsymbol{y}, \boldsymbol{x}_k) = \mathbf{R}^* (\boldsymbol{y} - \boldsymbol{r}_k), \tag{5.34a}$$

$$\boldsymbol{h}^{\mathcal{C}}(\boldsymbol{\psi}, \boldsymbol{x}_k) = \mathbf{R}^* \left(\rho \left(\boldsymbol{r}^{\mathcal{C}} - \boldsymbol{r}_k\right) + \boldsymbol{\eta}(\theta, \phi)\right), \tag{5.34b}$$

where $\mathbf{R}^* = \mathbf{R}_{\mathcal{C}}^{\mathcal{W}}$ is the rotation matrix from $\mathcal{C}$ to $\mathcal{W}$ computable from the quaternion q_k using the relation

$$\mathbf{R}^* = \mathbf{R}_{\mathcal{C}}^{\mathcal{W}} = \left(\mathbf{R}_{\mathcal{W}}^{\mathcal{C}}\right)^{-1} = \mathbf{R}(\bar{\mathrm{q}}_k). \tag{5.35}$$

Given a quaternion q, the corresponding rotation matrix $\mathbf{R}(\mathrm{q}_k)$ is given by

$$\mathbf{R}(\mathrm{q}) = \begin{pmatrix} q_r^2 + q_x^2 - q_y^2 - q_z^2 & -2q_rq_z + 2q_xq_y & 2q_rq_y + 2q_xq_z \\ 2q_rq_z + 2q_xq_y & q_r^2 - q_x^2 + q_y^2 - q_z^2 & -2q_rq_x + 2q_yq_z \\ -2q_rq_y + 2q_xq_z & 2q_rq_x + 2q_yq_z & q_r^2 - q_x^2 - q_y^2 + q_z^2 \end{pmatrix}. \tag{5.36}$$

Note that the same point encoded in the two different ways produces two different vectors $\boldsymbol{h}_{\psi}^{\mathcal{C}} \neq \boldsymbol{h}_{y}^{\mathcal{C}}$. This is not a problem because these vectors represent not a point but a direction in which the point takes place. In particular note that the two vectors are parallel.

Given the vector $\boldsymbol{h}^{\mathcal{C}}$, the expected projection in the image of the camera can be computed by means of the projection function $\underline{h}(\cdot)$ descibed in Section 2.5 in equation (2.29) when referring to the model proposed by Davison et al., or in equation (2.34) according to the model implemented in this work.

The measurements function is so given by

$$\boldsymbol{h}^*(\boldsymbol{\mu}) = \begin{pmatrix} \underline{h}\left(\boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_1, \boldsymbol{x}_k)\right) \\ \vdots \\ \underline{h}\left(\boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_m, \boldsymbol{x}_k)\right) \end{pmatrix}. \tag{5.37}$$

When the position of the interesting point in the 3D space comes out of the field of view of the camera, it is projected outside the image plane. For this reason, the real measurements vector $\boldsymbol{h}$ is given by a refinition phase of the just defined vector $\boldsymbol{h}^*$ where the points that are outside the field of view are removed. Moreover,the measurements of some features correctly projected in the image can be not available since the matching phase could fail, e.g., when occlusions occur. For this reason, in the EKF algorithm the vector $\boldsymbol{h}$ is related only to that features which has been correctly matched in the matching phase.

5.5.2 Uncertanty propagation

The same consideration holds for the jacobian matrix $\mathbf{H}^*$ of the vector $\boldsymbol{h}^*$. The matrix is computed for all the features in the state that should be projected in the next image, then real matrix $\mathbf{H}$ will be constructed by removing rows of $\mathbf{H}^*$ corresponding to features for which matches are not available.

The matrix $\mathbf{H}^*$ is given by

$$\mathbf{H}^* = \frac{\partial \boldsymbol{h}^*(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} = \begin{pmatrix} \frac{\partial}{\partial \boldsymbol{\mu}} \underline{h}\left(\boldsymbol{h}_1^{\mathcal{C}}\right) \\ \vdots \\ \frac{\partial}{\partial \boldsymbol{\mu}} \underline{h}\left(\boldsymbol{h}_m^{\mathcal{C}}\right) \end{pmatrix} = \begin{pmatrix} \frac{\partial \underline{h}\left(\boldsymbol{h}_1^{\mathcal{C}}\right)}{\partial \boldsymbol{h}_1^{\mathcal{C}}} \frac{\partial \boldsymbol{h}_1^{\mathcal{C}}}{\partial \boldsymbol{\mu}} \\ \vdots \\ \frac{\partial \underline{h}\left(\boldsymbol{h}_m^{\mathcal{C}}\right)}{\partial \boldsymbol{h}_m^{\mathcal{C}}} \frac{\partial \boldsymbol{h}_m^{\mathcal{C}}}{\partial \boldsymbol{\mu}} \end{pmatrix} \in \mathbb{R}^{2m \times n}, \tag{5.38}$$

where the jacobian $\partial \underline{h}\left(\boldsymbol{h}^{\mathcal{C}}\right)/\partial \boldsymbol{h}^{\mathcal{C}}$ is defined in Section 2.5 in equations 2.30 and 2.36 with reference respectively to the model proposed by Davison et al. and the model implemented in this work, while

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \boldsymbol{\mu}} = \begin{pmatrix} \frac{\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_i, \boldsymbol{x}_k)}{\partial \boldsymbol{x}_k} & 0 \dots 0 & \frac{\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_i, \boldsymbol{x}_k)}{\partial \boldsymbol{f}_i} & 0 \dots 0 \end{pmatrix} \in \mathbb{R}^{2 \times n}. \tag{5.39}$$

The jacobian $\frac{\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_i, \boldsymbol{x}_k)}{\partial \boldsymbol{x}_k}$ can also be written as

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}\left(\boldsymbol{f}_i, \boldsymbol{x}_k\right)}{\partial \boldsymbol{x}_k} = \begin{pmatrix} \frac{\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_i, \boldsymbol{x}_k)}{\partial \boldsymbol{r}_k} & \frac{\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f}_i, \boldsymbol{x}_k)}{\partial \boldsymbol{q}_k} & 0 \end{pmatrix} \in \mathbb{R}^{2 \times 13}. \tag{5.40}$$

Computing the three jacobians needs to take into account whether $\boldsymbol{f}_i$ is in euclidian or inverse depth encoding.

The $\partial \boldsymbol{h}^{\mathcal{C}}(\boldsymbol{f})/\partial \boldsymbol{r}_k$ is given by

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}\left(\boldsymbol{y}\right)}{\partial \boldsymbol{r}_k} = -\mathbf{R}^*, \tag{5.41a}$$

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}\left(\boldsymbol{\psi}\right)}{\partial \boldsymbol{r}_k} = -\rho \mathbf{R}^*. \tag{5.41b}$$

The jacobian $\partial \boldsymbol{h}^{\mathcal{C}}/\partial \mathrm{q}$ can be scomposed as

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \mathrm{q}_k} = \frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \bar{\mathrm{q}}_k} \frac{\partial \bar{\mathrm{q}}_k}{\partial \mathrm{q}_k}, \tag{5.42}$$

where the matrix $\partial \bar{\mathrm{q}}_k/\partial \mathrm{q}_k$ is simply

$$\frac{\partial \bar{\mathrm{q}}_k}{\partial \mathrm{q}_k} = \begin{pmatrix} 1 & 0 \\ 0 & -\mathbb{I}_3 \end{pmatrix}, \tag{5.43}$$

while

$$\frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \bar{\mathrm{q}}_k} = \begin{pmatrix} \frac{\partial \mathbf{R}^*}{\partial \bar{q}_r}\boldsymbol{d} & \frac{\partial \mathbf{R}^*}{\partial \bar{q}_x}\boldsymbol{d} & \frac{\partial \mathbf{R}^*}{\partial \bar{q}_y}\boldsymbol{d} & \frac{\partial \mathbf{R}^*}{\partial \bar{q}_z}\boldsymbol{d} \end{pmatrix}, \tag{5.44}$$

where $\boldsymbol{d}$ depends on feature coding as follows

$$\boldsymbol{d}\left(\boldsymbol{y}\right) = \boldsymbol{y} - \boldsymbol{r}_k, \qquad \boldsymbol{d}\left(\boldsymbol{\psi}\right) = \rho\left(\boldsymbol{r}^{\mathcal{C}} - \boldsymbol{r}_k\right) + \boldsymbol{\eta}\left(\theta, \phi\right). \tag{5.45}$$

The partial derivates of the matrix $\mathbf{R}^*$ with respect to the quaternion components follows directly from equation (5.36) as follows

$$\frac{\partial}{\partial q_r}\mathbf{R}(\mathrm{q}) = 2\begin{pmatrix} q_r & -q_z & q_y \\ q_z & q_r & -q_x \\ -q_y & q_x & q_r \end{pmatrix}, \tag{5.46a}$$

$$\frac{\partial}{\partial q_x}\mathbf{R}(\mathrm{q}) = 2\begin{pmatrix} q_x & q_y & q_z \\ q_y & -q_x & -q_r \\ q_z & q_r & -q_x \end{pmatrix}, \tag{5.46b}$$

$$\frac{\partial}{\partial q_y}\mathbf{R}(\mathrm{q}) = 2\begin{pmatrix} -q_y & q_x & q_r \\ q_x & q_y & q_z \\ -q_r & q_z & -q_y \end{pmatrix}, \tag{5.46c}$$

$$\frac{\partial}{\partial q_z}\mathbf{R}(\mathrm{q}) = 2\begin{pmatrix} -q_z & -q_r & q_x \\ q_r & -q_z & q_y \\ q_x & q_y & q_z \end{pmatrix}. \tag{5.46d}$$

Lastly

$$\frac{\partial \boldsymbol{h}(\boldsymbol{y})}{\partial \boldsymbol{y}} = \mathbf{R}^*, \tag{5.47a}$$

$$\frac{\partial \boldsymbol{h}(\boldsymbol{\psi})}{\partial \boldsymbol{\psi}} = \mathbf{R}^* \begin{pmatrix} \rho \mathbb{I}_3 & \begin{pmatrix} \cos\theta\cos\phi \\ 0 \\ -\sin\theta\cos\phi \end{pmatrix} & \begin{pmatrix} -\sin\theta\sin\phi \\ -\cos\phi \\ -\cos\theta\sin\phi \end{pmatrix} & \left(\boldsymbol{r}^{\mathcal{C}} - \boldsymbol{r}_k\right) \end{pmatrix} \tag{5.47b}$$

In contrast to system state evolution function, the measurement function just presented is strongly non linear. The non linearity are given first of all by the distortion effects introduced by the optical systems of the camera and moreover by the homogeneous nature camera systems, as seen in Chapter 2. Please note that the EKF filter works by linearizing the functions here presented. For this reason, if the non linearities of the system are too pronounced, then the EKF filter fails.

5.5.3 Implementation

Once obtained all the matrix needed by the EKF filter, the Kalman gain $\mathbf{K}_k$ and the innovation matrix $\mathbf{S}_k$ can be computed using the formula

$$\mathbf{S}_k = \mathbf{H}_k \boldsymbol{\Sigma}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k, \tag{5.48a}$$

$$\mathbf{K}_k = \bar{\boldsymbol{\Sigma}}_k \mathbf{H}_k^T \mathbf{S}_k^{-1}, \tag{5.48b}$$

where the matrix $\mathbf{R}_k$ represents the camera sensor noise in pixels, and it is given by

$$\mathbf{R}_k = \sigma_R^2 \mathbb{I}. \tag{5.49}$$

Since the errors in measurements are given mainly by the discretization errors due to digitalization, $\sigma_R^2 = 1$ can be assumed for the majority of the cameras.

To complete the Update Step it is now only required to apply the EKF Algorithm over the state vector.

The new state here found needs an additional treatment not considered in EKF algorithm, this because of the unit quaternion q_k in the state vector is not a summable entity as supposed by EKF. In simpler words, after the state transition the unity of $\mathrm{q}_{k|k}$ is not guarantee and it is necessary to normalize it according to

$$\mathrm{q}_{k|k} \leftarrow \operatorname{norm} \mathrm{q}_{k|k} = \frac{\mathrm{q}}{\|\mathrm{q}\|}. \tag{5.50}$$

Consequently, covariance matrix $\boldsymbol{\Sigma}_k$ need to be update using the equation

$$\boldsymbol{\Sigma}_{k|k} \leftarrow \begin{pmatrix} \mathbb{I}_3 & 0 & 0 \\ 0 & \frac{\partial \operatorname{norm} \mathrm{q}}{\partial \mathrm{q}} & 0 \\ 0 & 0 & \mathbb{I} \end{pmatrix} \boldsymbol{\Sigma}_{k|k} \begin{pmatrix} \mathbb{I}_3 & 0 & 0 \\ 0 & \frac{\partial \operatorname{norm} \mathrm{q}}{\partial \mathrm{q}} & 0 \\ 0 & 0 & \mathbb{I} \end{pmatrix}^T. \tag{5.51}$$

Where the jacobian $^{\partial \,\mathrm{norm}\,\mathrm{q}_k}/_{\partial \mathrm{q}_k}$ is given by

$$\begin{aligned}\frac{\partial\,\mathrm{norm}(\mathrm{q})}{\partial \mathrm{q}} &= \frac{\partial}{\partial \mathrm{q}}\frac{\mathrm{q}}{\|\mathrm{q}\|} = \frac{\partial}{\partial \mathrm{q}}\left(\mathrm{q}\frac{1}{\|\mathrm{q}\|}\right)\\ &= \frac{1}{\|\mathrm{q}\|}\frac{\partial \mathrm{q}}{\partial \mathrm{q}} + \mathrm{q}\left(\frac{\partial}{\partial\|\mathrm{q}\|}\frac{1}{\|\mathrm{q}\|}\right)\frac{\partial\|\mathrm{q}\|}{\partial \mathrm{q}}\\ &= \frac{1}{\|\mathrm{q}\|}\mathbb{I}_3 - \mathrm{q}\frac{1}{\|\mathrm{q}\|^2}\frac{\mathrm{q}^T}{\|\mathrm{q}\|}\\ &= \frac{1}{\|\mathrm{q}\|^3}\left(\|\mathrm{q}\|^2\mathbb{I}_3 - \mathrm{q}\mathrm{q}^T\right).\end{aligned} \tag{5.52}$$

Now that the actual state $\boldsymbol{\mu}_{k|k}$ and its covariance are $\boldsymbol{\Sigma}_{k|k}$ have been corrected, the prediction phase of the extended Kalman filter can be performed. Note that it is more convenient to perform the prediction operations before that the new frame is available. In this way the system is ready to perform the refinement updating step as soon as the new measurements are available.

5.5.4 Features Matching

Now that all the step of the EKF algorithm has been discussed, an important part of the Mono-SLAM algorithm not included in the EFK filter needs to be exploited. In fact, the EKF filter supposes that the measurements come from a sensor able to measure them directly. Unfortunately, the camera sensor is not able to measure the coordinate in the image of a given point in the space, so that this job is committed by a part of the algorithm labeled as features matching.

The task of features matching is to find the position of the interested features when a new frame income. The matches are found between the new frame and the descriptors of the features stored in memory when the new features in initialized. As seen in Chapter 3, all the techniques which do not require too much computational efforts to be execute in real time are poor in robustness. Luckily, using a probabilistic approach helps this phase since the region where the features is searched can be reduced to the region where the feature is much probable to lie in. In other words, the matches searching are performed in a region around the prediction of the measurements which size is related to the covariance matrix of that features, in fact it is very intuitive that the bigger the uncertainty of the features is, the wider the searching region should be.

In the Mono-SLAM algorithm, that region is given by projecting the $k^2\boldsymbol{\Sigma}_i$ ellipsoid related to the prediction of the feature f_i in the image plane, where k is a number specifying the size of the region, usually $k = 2$ or $k = 3$, i.e., 95% or 99% of probability to find the feature respectively. Please note that the size of this region is given by $k^2\mathbf{S}_i$, where $\mathbf{S}_i$ is the 2×2 submatrix of $\mathbf{S}_k$ related to the measurements estimation of $\boldsymbol{f}_i$. Hence, the matching is performed on the ellipsoid $\mathcal{S}_i$ having center in $\underline{p}_i$ and size $k^2\mathbf{S}_i$ mathematically described as follows:

$$\mathcal{S}_i = \left\{\underline{p} \in \mathcal{I} \middle| \left(\underline{p} - \underline{p}_i\right)^T \mathbf{S}_i^{-1}\left(\underline{p} - \underline{p}_i\right) \leqslant k^2\right\}.$$

If the feature can be matched inside this search region using an appropriate comparison, then the feature is considered to be successfully matched. The image coordinates $\underline{z}_i$, denoting the position of the match, are appended to measurement vector $\boldsymbol{z}_k$ and the feature will contribute towards the correction of the estimates mean $\boldsymbol{\mu}_k$ and covariance $\boldsymbol{\Sigma}_k$. Otherwise the predicted measurement for $\boldsymbol{f}_i$ will be removed from $\boldsymbol{h}_k^*$, since no corresponding measurement in $\boldsymbol{z}_k$ is present. Subsequently, the corresponding rows of matrix $\mathbf{H}_k^*$ are deleted.

Since this methods, known as *active search*, makes the algorithm to be much more robust, it allows also to use very simple and fast features descriptors to perform matching. For this reason, in the implementation by Davison et al. but also in the one of this Master Thesis, the simple patch matching, explained in Section 3.1 are used.

5.6 Inverse depth and XYZ codings

As seen previously, there are two different ways to represent features in the Mono-SLAM algorithm, both presenting advantages and limits. In particular, inverse depth encoding is very useful when the depth of a certain features is not well known, for instance because it is very far from the camera (a feature at infinity) or it is just detected for the first time. On the other hand, depth encoding is simpler and requires less computational effort and memory and it is preferable when the depth of the features is well known. This Section, based on the works of Civera et al. [42, 47], presents a mathematical analysis of the advantages in using inverse depth encoded features and an algorithm that allows to switch from inverse depth to euclidian encoding when it is more convenient.

5.6.1 Feature Linearity

As know, EKF linearizes the model of the system and applies the standard Kalman Filter (KF). In particular, the more linear the equations are, the better an EKF performs. According to this consideration, Civera et al. show that the linearity analysis of the measurement equation support the superiority of the inverse depth encoding.

Linearized Propagation of a Gaussian

Let us consider a stochastic variable $\boldsymbol{x} \sim \mathrm{N}(\boldsymbol{\mu}_x, \boldsymbol{\Sigma}_x)$ and a nonlinear function f. The new stochastic variable $\boldsymbol{y} = f(\boldsymbol{x})$ can be approximate as a joint gaussian variable as it follows:

$$\boldsymbol{y} \sim \mathrm{N}(\boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y), \quad \boldsymbol{\mu}_y = f(\boldsymbol{\mu}_x), \quad \boldsymbol{\Sigma}_y = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\boldsymbol{\mu}_x} \boldsymbol{\Sigma}_x \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\mu_x}^{T}. \tag{5.53}$$

As shown in Figure 5.5, the approximation is better if the function is linear in an interval around $\boldsymbol{\mu}_x$, in particular the size of this interval is strongly related to $\boldsymbol{\Sigma}_x$. From now on, to simplify the mathematical treatment, all the variables will be considered mono dimensional. The bigger σ_x the wider the interval has to be to cover a significant fraction of the random variable x values. Civera et al. fix the linearity interval to 95% confidence region defined by

$$\mathcal{I} = [\mu_x - 2\sigma_x, \mu_x + 2\sigma_x]\,.$$

If the function is linear in an interval, the first order derivative is constant on that interval

$$\frac{\partial f}{\partial x}(\mu_x + \delta x) \approx \left.\frac{\partial f}{\partial x}\right|_{\mu_x}. \tag{5.54}$$

To analyze the variations of the first derivate, consider the first order Taylor expansion

$$\frac{\partial f}{\partial x}(\mu_x + \delta x) \approx \left.\frac{\partial f}{\partial x}\right|_{\mu_x} + \left.\frac{\partial^2 f}{\partial x^2}\right|_{\mu_x} \delta x. \tag{5.55}$$

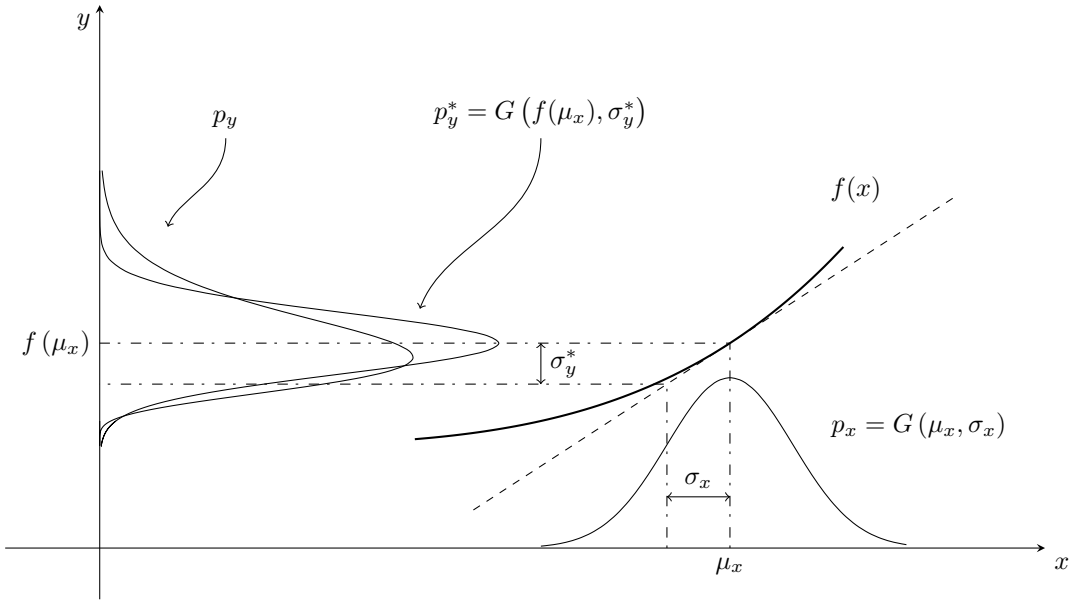


Figure 5.5: Gaussian propagation. The stochastic gaussian variable x is mapped on the variable y by mean of the function $f(\cdot)$. The real probabilistic distribution of y is labeled by p_y. The linearization approximates p_y with the gaussian distribution p_y^* obtained by linearizing the function $f(\cdot)$ in μ_x.

From equations (5.54) and (5.55) follows that for a linear function holds

$$\left.\frac{\partial f}{\partial x}\right|_{\mu_x} \gg \left.\frac{\partial^2 f}{\partial x^2}\right|_{\mu_x} \delta x \qquad \forall \delta x \in \mathcal{I} - \mu_x,$$

i.e, the second derivate is negligible with respect to the first derivate for each point in the interval, and in particular in the extrema. The linearity index of a function is so defined as

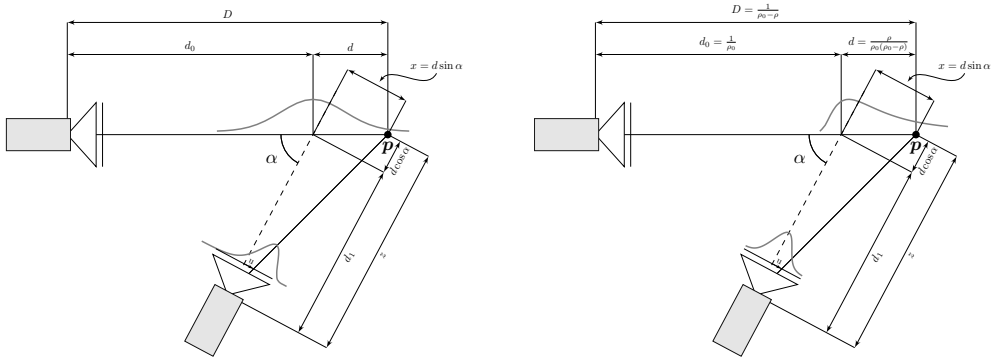
$$L = \left| \frac{\left.\frac{\partial^2 f}{\partial x^2}\right|_{\mu_x} 2\sigma_x}{\left.\frac{\partial f}{\partial x}\right|_{\mu_x}} \right|. \tag{5.56}$$

When $L \approx 0$, the function is almost linear and gaussianity is preserved.

Measurement Equation Linearity

Here it is presented a simplified measurement equation model for a mobile camera observing a scene point which successfully codes the departure from linearity.

Let us consider a camera with unitary focal length $f = 1$ projecting a point into the image plane. For simplicity only one axis of the image plane is consider, obviously the same considerations apply to the other axis. According to the camera pinhole model, the projected point $\underline{p} = \begin{pmatrix} u \end{pmatrix}$ of



(a) Depth coding

(b) Invese depth coding

Figure 5.6: Uncertainty propagation in depth and inverse depth.

the point $\boldsymbol{p} = \begin{pmatrix} x & z \end{pmatrix}^T$ is given by

$$u = \frac{x}{z}.$$

The camera observes the same point from two different points of view. The first observation detects the ray where the point belongs to, so the only location error of the point is in depth. Then, the camera observes the point at a distance d_1 with a parallax angle α which is approximated by the angle between the two optical axises.

The linearity propagation of the measurement equation in the two different codings will now be analyzed. The following discussion is schematized in Figure 5.6.

Linearity in Euclidian Encoding Consider a feature $\boldsymbol{y}$ describing a point $\boldsymbol{p}$ in euclidian coding. Depth is initialized with a value d_0 and an error d assumed to be Gaussian with zero mean, i.e. $d \sim \mathrm{N}(0, \sigma_d^2)$. The actual depth is given by $D = d_0 + d$.

From Figure 5.6a it comes that

$$x = d\sin\alpha, \qquad z = d_1 + d\cos\alpha,$$

hence the projected point is given by

$$u = \frac{d\sin\alpha}{d_1 + d\cos\alpha}. \tag{5.57}$$

The u variable is a function depending on the Gaussian variable d. The first and second order derivates are

$$\frac{\partial u}{\partial d} = \frac{d_1\sin\alpha}{(d_1 + d\cos\alpha)^2}, \qquad \frac{\partial^2 u}{\partial d^2} = \frac{-2d_1\sin\alpha\cos\alpha}{(d_1 + d\cos\alpha)^3},$$

and the linearity is computed from equation (5.56)

$$L_d = \left| \frac{\frac{\partial^2 u}{\partial d^2}\big|_{d=0} 2\sigma_d}{\frac{\partial u}{\partial d}\big|_{d=0}} \right| = \frac{4\sigma_d}{d_1}\left|\cos\alpha\right|. \tag{5.58}$$

If $\alpha \to 0$ then $\cos\alpha \to 1$ so that $L_d \approx \frac{4\sigma_d}{d_1}$. The most important factor influencing L_d value is obviously the variance σ_d. While in initialization and for features at infinity the depth variance has to cover a large interval of values, it comes out that σ_d is big and so the linearity index is $L_d \gg 0$. In other words, for low parallax features the projection function of the features in the image plane is strongly non linear and the approximation to a gaussian distribution does not hold. On the contrary, if the variance σ_d is small, i.e., when the feature is observed with big parallax, the linearity index is small and the gaussianity is maintained.

Linearity in Inverse Depth Encoding Consider the same scene but coded using inverse depth, as in Figure 5.6b. The ρ_0 parameter is affected by an error $\rho \sim \mathrm{N}(0, \sigma_\rho^2)$. The three variable D, d and d_0 described in the previous paragraph are given by

$$D = \frac{1}{\rho_0 - \rho}, \quad d_0 = \frac{1}{\rho_0}, \quad d = \frac{\rho}{\rho_0\left(\rho_0 - \rho\right)}. \tag{5.59}$$

The coordinates of the point $\boldsymbol{p}$ are now well defined

$$x = \frac{\rho \sin\alpha}{\rho_0\left(\rho_0 - \rho\right)}, \qquad z = d_1 + \frac{\rho\cos\alpha}{\rho_0\left(\rho_0 - \rho\right)},$$

and the projection on the image plane can be computed as follows

$$u = \frac{\rho\sin\alpha}{d_1\rho_0\left(\rho_0 - \rho\right) + \rho\cos\alpha}. \tag{5.60}$$

The first and second order derivates of u are given by

$$\frac{\partial u}{\partial \rho} = \frac{\rho_0^2 d_1 \sin\alpha}{\left(\rho_0 d_1\left(\rho_0 - \rho\right) + \rho\cos\alpha\right)^2},$$
$$\frac{\partial^2 u}{\partial \rho^2} = \frac{-2\rho_0^2 d_1\left(\cos\alpha - d_1\rho_0\right)\sin\alpha}{\left(\rho_0 d_1\left(\rho_0 - \rho\right) + \rho\cos\alpha\right)^3}.$$

Hence, the linearity index is

$$L_\rho = \left|\frac{\frac{\partial^2 u}{\partial\rho^2}\Big|_{\rho=0} 2\sigma_\rho}{\frac{\partial u}{\partial\rho}\Big|_{\rho=0}}\right| = \frac{4\sigma_\rho}{\rho_0}\left|1 - \frac{d_0}{d_1}\cos\alpha\right|. \tag{5.61}$$

For small displacements of the camera, $d_0 \approx d_1$ and $\alpha \approx 0$. Moreover, since ρ appears in denominator, small values of its variance are sufficient to express a large confidence region of D, that implies small value of the variance σ_ρ. Hence, $L_\rho \approx 0$ i.e., a very linear function in the 95% confidence region

$$D = \left[\frac{1}{\rho_0 + 2\sigma_\rho}, \frac{1}{\rho_0 - 2\sigma_\rho}\right].$$

Note that, also when the depth is precisely known, i.e., for small values of σ_ρ, the linearity index is near zero.

Encodings Comparison Some consideration about the two linearity indexes L_d and L_ρ are now necessary. As seen, $L_d < \frac{4\sigma_d}{d_1}$ and $L_\rho \approx 0$ hold everywhere. Hence, the inverse depth works well in both case of features with a good knowledge of depth and with features at infinity or just initialized, while euclidian features are linear only in case of good knowledge of the depth. However, euclidian codings has the good advantage of using only 3 parameters instead of 6, reducing computational efforts required. For this reasons, inverse depth features are useful in initialization and for features at infinity, while XYZ coding is preferable otherwise.

5.6.2 Inverse Depth to Euclidian conversion

While both codings present advantages and disadvantages in separated cases, a mixed approach that uses inverse depth features for point with big uncertainty on the depth and euclidian coding otherwise is possible. Regardingly, in [47] is proposed a method to switch from inverse depth to euclidian coding. In particular Civera et al. use a linearity threshold for L_d to decide if the gaussianity of u is maintained and then if switching is possible. In [47, sect. III-A], they fix the threshold as follows

$$L_d = \frac{4\sigma_d}{d_1} |\cos\alpha| < 10\%. \tag{5.62}$$

The main idea is (i) initialize all features in inverse depth coding, (ii) compute the L_d index for each inverse depth feature after the EKF algorithm and (iii) convert the features having an index below the threshold proposed in (5.62).

Computing L_d Cosider an inverse depth feature $\boldsymbol{\psi}$ and its covariance matrix $\boldsymbol{\Sigma}_\psi$. According to Figure 5.6 and equations (5.7) and (5.8), length d_1 is simply the distance of the features to the optical center of the camera

$$d_1 = \|\boldsymbol{d}\|, \qquad \boldsymbol{d} = \boldsymbol{y} - \boldsymbol{r}_k \tag{5.63}$$

while $\cos\alpha$ is the angle between the versor $\boldsymbol{\eta} = \boldsymbol{\eta}(\theta, \phi)$ and the vector $\boldsymbol{d}$

$$\cos\alpha = \frac{\boldsymbol{d}^T \cdot \boldsymbol{\eta}}{\|\boldsymbol{d}\|\|\boldsymbol{\eta}\|} = \frac{\boldsymbol{d}^T \cdot \boldsymbol{\eta}}{d_1}. \tag{5.64}$$

The value σ_d is computable from equation (5.53)

$$\sigma_d = \left|\frac{\partial}{\partial\rho}\frac{1}{\rho}\right| \sigma_\rho = \frac{\sigma_\rho}{\rho^2}. \tag{5.65}$$

The L_d value is now well defined and it can be comparated with the threshold.

Conversion Mechanism If L_d is below the switching threshold the feature is converted in euclidian encoding according to equation (5.7) and substituted in the state vector $\boldsymbol{\mu}_k$. It is now necessary to update the covariance matrix $\boldsymbol{\Sigma}_k$

$$\boldsymbol{\Sigma}_k \leftarrow \mathbf{J}\boldsymbol{\Sigma}_k\mathbf{J}^T, \qquad \mathbf{J} = \begin{pmatrix} \mathbb{I} & 0 & 0 \\ 0 & \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\psi}} & 0 \\ 0 & 0 & \mathbb{I} \end{pmatrix}, \tag{5.66}$$

where

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\psi}} = \begin{pmatrix} \mathbb{I}_3 & \frac{1}{\rho}\begin{pmatrix} \cos\theta\cos\phi \\ 0 \\ -\sin\theta\cos\phi \end{pmatrix} & \frac{1}{\rho}\begin{pmatrix} -\sin\theta\sin\phi \\ -\cos\phi \\ -\cos\theta\sin\phi \end{pmatrix} & -\frac{1}{\rho^2}\boldsymbol{\eta}(\theta,\phi) \end{pmatrix}. \tag{5.67}$$

The new state and the new covariance matrix can now be used as normally in the next EKF steps to predict the evolution of the camera movement with less computational effort required.

5.6.3 Features Initialization

A new feature, detected for the first time at instant k, is initialized in inverse depth coding as follows

$$\boldsymbol{\psi} = \boldsymbol{\psi}\left(\boldsymbol{r}_k, \mathrm{q}_k, \underline{p}_{\psi}, \rho_0\right) = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} & \theta & \phi & \rho \end{pmatrix}^T, \tag{5.68}$$

where $\underline{p}_{\psi} = \begin{pmatrix} u_d & v_d \end{pmatrix}^T$ is the position of the feature in the image and ρ_0 is the initial inverse depth estimation.

The first three coordinates of the feature state represent the position of the optical center at the first observation, so that

$$\boldsymbol{r}^{\mathcal{C}} = \begin{pmatrix} x^{\mathcal{C}} & y^{\mathcal{C}} & z^{\mathcal{C}} \end{pmatrix}^T = \boldsymbol{r}_k. \tag{5.69}$$

To compute azimuth θ and elevation ϕ it is necessary to calculate the directional vector $\boldsymbol{h}^{\mathcal{W}}$ representing the line connecting the optical center to the feature in the world RF, i.e., the locus on which the feature can be found

$$\boldsymbol{h}^{\mathcal{W}} = \mathbf{R}\left(\mathrm{q}_k\right)\boldsymbol{h}^{\mathcal{C}}, \tag{5.70}$$

where $\boldsymbol{h}^{\mathcal{C}} = \boldsymbol{h}\left(\underline{p}_{\psi}\right)$ is obtained using the unprojection function of the camera defined in Chapter 2 in equations (2.32) when referring to the model used by Davison et al. and (2.35) when referring to the model implemented in this work.

The value of θ and ϕ can now be computed as

$$\begin{pmatrix} \theta \\ \phi \end{pmatrix} = \begin{pmatrix} \arctan\left(h_x^{\mathcal{W}}, h_z^{\mathcal{W}}\right) \\ \arctan\left(-h_y^{\mathcal{W}}, \sqrt{(h_x^{\mathcal{W}})^2 + (h_z^{\mathcal{W}})^2}\right) \end{pmatrix}. \tag{5.71}$$

The parameter ρ can not be estimate because the camera can not measure depth (or inverse depth), for this reason it is initialized with a predefined initial inverse depth estimation $\rho = \rho_0$. According to [42], the parameters ρ and σ_ρ is initialized in order to consider also the point at infinity, for instance, they could be initialized with $\rho_0 = 0.1$ and $\sigma_\rho = 0.5$, which gives a 95% confidence region $[\rho_0 - 2\sigma_\rho; \rho_0 + 2\sigma_\rho] = [-0.9; 1.1]$ containing the point at infinity.

The description of the new feature $\boldsymbol{\psi}$ is now well defined and it can be appended to the state vector $\boldsymbol{\mu}$. It is now necessary to updated the covariance matrix $\boldsymbol{\Sigma}_k$ to handle the new feature. The old covariance matrix will be indicated with $\boldsymbol{\Sigma}'_k$ from now.

The new covariance matrix can be computed as

$$\boldsymbol{\Sigma}_k = \mathbf{J}\begin{pmatrix} \boldsymbol{\Sigma}'_k & 0 & 0 \\ 0 & \mathbf{Q}_I & 0 \\ 0 & 0 & \sigma_\rho^2 \end{pmatrix}\mathbf{J}^T \in \mathbb{R}^{(n+6)\times(n+6)} \tag{5.72}$$

with

$$\boldsymbol{\Sigma}'_t \in \mathbb{R}^{n\times n} \qquad \mathbf{J} \in \mathbb{R}^{(n+6)\times(n+3)} \qquad \mathbf{Q}_I \in \mathbb{R}^{2\times 2}.$$

The matrix $\mathbf{Q}_I$ contains the variance of the image measurement noise and has the same form of $\mathbf{R}_k$ in equation (5.49).

The matrix $\mathbf{J}$ is constructed by a $n \times n$ identity matrix and the partial derivate of function $\boldsymbol{\psi}$ in equation (5.68) as

$$\mathbf{J} = \begin{pmatrix} \mathbb{I}_n & & & & 0 & \\ \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{r}_k} & \frac{\partial \boldsymbol{\psi}}{\partial \mathrm{q}_k} & 0\cdots 0 & \frac{\partial \boldsymbol{\psi}}{\partial \underline{h}_{\psi}} & \frac{\partial \boldsymbol{\psi}}{\partial \rho} \end{pmatrix} \tag{5.73}$$

The computation of the partial derivates are below treated.

From equation (5.69) it follows directly

$$\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{r}_k} = \begin{pmatrix} \mathbb{I}_3 \\ 0 \end{pmatrix} \in \mathbb{R}^{6\times 3}, \qquad \frac{\partial \boldsymbol{\psi}}{\partial \rho} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}^T. \tag{5.74}$$

The Jacobians $^{\partial \boldsymbol{\psi}}/_{\partial \mathrm{q}_k}$ and $^{\partial \boldsymbol{\psi}}/_{\partial \underline{h}_\psi}$ are given by

$$\frac{\partial \boldsymbol{\psi}}{\partial \mathrm{q}_k} = \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{h}^{\mathcal{W}}} \frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \mathrm{q}_k}, \tag{5.75}$$

$$\frac{\partial \boldsymbol{\psi}}{\partial \underline{h}_\psi} = \frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{h}^{\mathcal{W}}} \frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \boldsymbol{h}^{\mathcal{C}}} \frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \underline{h}_u} \frac{\partial \underline{h}_u}{\partial \underline{h}_\psi}, \tag{5.76}$$

with

$$\frac{\partial \boldsymbol{\psi}}{\partial \boldsymbol{h}^{\mathcal{W}}} = \begin{pmatrix} 0 & 0 & 0 & \frac{\partial \theta}{\partial \boldsymbol{h}^{\mathcal{W}}}^T & \frac{\partial \phi}{\partial \boldsymbol{h}^{\mathcal{W}}}^T & 0 \end{pmatrix}^T \in \mathbb{R}^{6\times 3}. \tag{5.77}$$

The partial derivates of the two angles with respect to $\boldsymbol{h}^{\mathcal{W}}$ can be derivated from equation (5.71), while $^{\partial \boldsymbol{h}^{\mathcal{W}}}/_{\partial \mathrm{q}_k}$ can be computed from equation (5.70). The resulting Jacobians are specified as

$$\frac{\partial \theta_i}{\partial \boldsymbol{h}^{\mathcal{W}}} = \frac{1}{\left(h_x^{\mathcal{W}}\right)^2 + \left(h_z^{\mathcal{W}}\right)^2} \begin{pmatrix} h_z^{\mathcal{W}} & 0 & -h_x^{\mathcal{W}} \end{pmatrix} \tag{5.78}$$

$$\frac{\partial \phi_i}{\partial \boldsymbol{h}^{\mathcal{W}}} = \frac{1}{\left(h_x^{\mathcal{W}}\right)^2 + \left(h_y^{\mathcal{W}}\right)^2 + \left(h_z^{\mathcal{W}}\right)^2} \begin{pmatrix} \frac{h_x^{\mathcal{W}} h_y^{\mathcal{W}}}{\sqrt{(h_x^{\mathcal{W}})^2 + (h_z^{\mathcal{W}})^2}}, \\ -\sqrt{\left(h_x^{\mathcal{W}}\right)^2 + \left(h_z^{\mathcal{W}}\right)^2} \\ \frac{h_y^{\mathcal{W}} h_z^{\mathcal{W}}}{\sqrt{(h_x^{\mathcal{W}})^2 + (h_z^{\mathcal{W}})^2}}, \end{pmatrix}^T \tag{5.79}$$

$$\frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \mathrm{q}_k} = \begin{pmatrix} \frac{\partial \mathbf{R}(\mathrm{q}_k)}{\partial q_r} \boldsymbol{h}^{\mathcal{C}} & \frac{\partial \mathbf{R}(\mathrm{q}_k)}{\partial q_x} \boldsymbol{h}^{\mathcal{C}} & \frac{\partial \mathbf{R}(\mathrm{q}_k)}{\partial q_y} \boldsymbol{h}^{\mathcal{C}} & \frac{\partial \mathbf{R}(\mathrm{q}_k)}{\partial q_z} \boldsymbol{h}^{\mathcal{C}} \end{pmatrix}. \tag{5.80}$$

The last jacobian $^{\partial \boldsymbol{\psi}}/_{\partial m}$ is so computed:

$$\frac{\partial \boldsymbol{\psi}}{\partial \underline{h}_\psi} = \begin{pmatrix} 0 & 0 & 0 & \frac{\partial \theta}{\partial \underline{h}_\psi} & \frac{\partial \phi}{\partial \underline{h}_\psi} & 0 \end{pmatrix}, \tag{5.81}$$

where, as usual, the two jacobian can by decomposed as

$$\frac{\partial \theta}{\partial \underline{h}_\psi} = \frac{\partial \theta}{\partial \boldsymbol{h}^{\mathcal{W}}} \frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \underline{h}_\psi}, \tag{5.82}$$

$$\frac{\partial \phi}{\partial \underline{h}_\psi} = \frac{\partial \phi}{\partial \boldsymbol{h}^{\mathcal{W}}} \frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \underline{h}_\psi}. \tag{5.83}$$

The only remaining jacobian $^{\partial \boldsymbol{h}^{\mathcal{W}}}/_{\partial \underline{h}_\psi}$ is given by equation

$$\frac{\partial \boldsymbol{h}^{\mathcal{W}}}{\partial \underline{h}_\psi} = \mathbf{R}(\mathrm{q}) \frac{\partial \boldsymbol{h}^{\mathcal{C}}}{\partial \underline{h}_\psi}, \tag{5.84}$$

where $^{\partial \boldsymbol{h}^{\mathcal{C}}}/_{\partial \underline{h}_\psi}$ is computed in Capter 2 in equations 2.32 and 2.37 respectively for Davison et al. model and the one implemented in this work.

Over the previously explained advantages in initializing features directly in inverse depth coding, there is an other benefit which has been not well discussed yet. That is, in particular, the possibility of the inverse depth coded feature to be put in the EKF state directly without a preprocessing phase. In fact, since at the very first observation of the features the depth is completely unknown, depth features can not be put in the state directly. The first implementations of Mono-SLAM, indeed, use a temporary stack in which features just detected are put to be preprocessed. The preprocessing phase consists in a standard triangulation algorithm able to estimate the depth position of the features. When the depth of the feature is enough precise, the feature is pushed from this stack and put in the EKF state directly in euclidian coding. Using the inverse depth formulation, this preprocessing phase is avoided.

5.6.4 State Management: Features Deletion

Features deletion mechanism, that consists in removing features from the state if they are not needed, plays an important role in the algorithm while it is not taken into account by theoretical descriptions of EKF and its applications. In fact, without any deletion mechanism state vector $\boldsymbol{\mu}_k$ would be ever-growing, slowing down the performance of the EKF and the Mono-SLAM itself. Moreover, considering not robust features in matching mechanism, may lead to mismatching and catastrophic events such as losing the state evolution.

Reliable features which are matched repeatedly should of course not be deleted, but there may be features that will be repeatedly rejected by matching phase after their initialization. Such features will not contribute to the state estimation in any way, but result in additional computational effort. If such features are removed from the state vector, new promising features may be added without endangering run-time constraints imposed by real time constraints. An effective mechanism to detect if a feature should be deleted consists in measuring the ratio of successful matches to the number of match attempts. If this ratio is below a given threshold, the feature will be deleted. Through this mechanism is quite simple it preserves stable features outside of the current field of view. To add to robustness the matching ratio should only be considered, after an initial number of matches was attempted. Of course this basic idea can be extended in various manners. For example if a certain number of successful matches are recorded for a feature its match ratio may be reduced (since the high number of matches implies that for certain camera positions the feature is quite stable and thus improves the overall estimation).

An alternative solution consists in removing features having covariance larger than a certain threshold. A too big covariance, in fact, implicates that the feature is not robust (which lead to noisy measurements and so the enlargement of the covariance matrix) or that the feature is not matched correctly, so that the measurements are not available and the covariance are not updated.

The deletion from the EKF itself is much more simple than the addition of a new feature. If point $\boldsymbol{\psi}_i$ is to be deleted it just needs to be removed from the current state vector and the covariance.

Finally, the deletion mechanism may introduce some robustness to a non static environment. If some distinct features are detected on a moving object they will be added to the EKF like any other feature. However if the estimation of the camera movement is stable enough a feature on a moving object will repeatedly not be matched in its predicted search region and thus be deleted again after a short time.

5.7 Dimensionless Monocular-SLAM

As said previously, all the monocular approaches loss the scale of the environment in which the camera is moving. However, it has been shown in [42] that using an inverse depth parametrization when features are initialized, it allows the complete algorithm to properly work without any knowledge of the scale of the environment. In other words, the reconstructed scene and camera motion will be coherent each other, but expressed in a unit of measurement totally unknown. For this reason, in [9], authors have decided to separate completely the reconstructed map and camera state from their dimension, i.e. space and time, according to the well known Buckingham's Π Theorem [48] stating that the physical laws are independent of units.

This Section is devoted to present the dimensionless version of the previously presented Mono-SLAM algorithm, introducing the new dimensionless variables of the algorithm and reformulating the equations which change from the previously discussed word.

Note that in the follow discussion, a dimensionless variable corresponding to a variable a will indicate with the symbol $\mathring{a}$.

The new dimensionless state takes the following form:

$$\mathring{\boldsymbol{\mu}}_k = \begin{pmatrix} \mathring{\boldsymbol{x}}_k \\ \mathring{\boldsymbol{f}}_0 \\ \vdots \\ \mathring{\boldsymbol{f}}_m \end{pmatrix}, \tag{5.85}$$

where the $\mathring{\boldsymbol{x}}_k$ vector loses both time and space units, while the map, i.e., $\mathring{\boldsymbol{f}}_i$ loses only the space unit. In particular, the new camera state is related to the old one as follows:

$$\mathring{\boldsymbol{x}}_k = \begin{pmatrix} \mathring{\boldsymbol{r}}_k \\ \mathring{\mathrm{q}}_k \\ \mathring{\boldsymbol{v}}_k \\ \mathring{\boldsymbol{\omega}}_k \end{pmatrix} = \begin{pmatrix} \boldsymbol{r}_k/d \\ \mathrm{q}_k \\ \boldsymbol{v}_k \Delta t/d \\ \boldsymbol{\omega}_k \Delta t \end{pmatrix}, \tag{5.86}$$

where the d parameters is the space units (unknown) which is removed from the state. Also the time unit, i.e., Δt is removed from the state to enhance the independence of the algorithm to the camera frame rate.

On the other side, dimensionless features are related to the old ones by removing the d units from the dimensionfull features, as follows:

$$\begin{cases} \mathring{\boldsymbol{y}} = \boldsymbol{y}/d \\ \mathring{\boldsymbol{\psi}} = \begin{pmatrix} \boldsymbol{r}^{\mathcal{C}}/d & \theta & \phi & \rho d \end{pmatrix}^T. \end{cases} \tag{5.87}$$

Note that in both cases the parameters measuring to angle, i.e., q, θ and ϕ, are not reformulated since they are already dimensionless.

The new prediction function takes the following form

$$\mathring{\boldsymbol{x}}_{k+1} = \boldsymbol{g}_v\left(\mathring{\boldsymbol{x}}_k\right) = \begin{pmatrix} \mathring{\boldsymbol{r}}_k + \mathring{\boldsymbol{v}}_k + \mathring{\boldsymbol{V}}_k \\ \mathrm{q}_k \times \mathrm{quat}\left(\mathring{\boldsymbol{\omega}}_k + \mathring{\boldsymbol{\Omega}}_k\right) \\ \mathring{\boldsymbol{v}}_k + \mathring{\boldsymbol{V}}_k \\ \mathring{\boldsymbol{\omega}}_k + \mathring{\boldsymbol{\Omega}}_k \end{pmatrix}, \tag{5.88}$$

where the time units related to speeds is removed.

The jacobian used to linearize the prediction function is the following:

$$\frac{\partial \boldsymbol{g}_v\left(\mathring{\boldsymbol{x}}_k\right)}{\partial \boldsymbol{x}_k} = \begin{pmatrix} \mathbb{I}_3 & 0 & \mathbb{I}_3 & 0 \\ 0 & \frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{q}_k} & 0 & \frac{\partial \mathrm{q}_{k+1}}{\partial \mathring{\boldsymbol{\omega}}_k} \\ 0 & 0 & \mathbb{I}_3 & 0 \\ 0 & 0 & 0 & \mathbb{I}_3 \end{pmatrix} \in \mathbb{R}^{13\times 13}. \tag{5.89}$$

From now on, it is used $\mathring{\boldsymbol{\omega}} = \mathring{\boldsymbol{\omega}}_k$ and $\mathrm{h} = \mathrm{quat}\left(\mathring{\boldsymbol{\omega}}_k + \mathring{\boldsymbol{\Omega}}_k\right)$ to simplify the notation.

Jacobian ${}^{\partial \mathrm{q}_{k+1}}/_{\partial \mathrm{q}_k}$ has the same expression of the jacobian in equation (5.21), while ${}^{\partial \mathrm{q}_k(k+1)}/_{\partial \boldsymbol{\omega}}$ is so computed

$$\frac{\partial \mathrm{q}_{k+1}}{\partial \boldsymbol{\omega}} = \frac{\partial \mathrm{q}_{k+1}}{\partial \mathrm{h}} \frac{\partial \mathrm{h}}{\partial \boldsymbol{\omega}}, \tag{5.90}$$

where jacobian ${}^{\partial \mathrm{q}_{k+1}}/_{\partial \mathrm{h}}$ has been already computed in equation (5.23) and

$$\frac{\partial \mathrm{h}}{\partial \mathring{\boldsymbol{\omega}}} = \frac{1}{2}\begin{pmatrix} -\sin\frac{\|\mathring{\boldsymbol{\omega}}\|}{2}\boldsymbol{n}_{\mathring{\boldsymbol{\omega}}}^T \\ \mathbb{I}_3 \,\mathrm{Sinc}\,\frac{\|\mathring{\boldsymbol{\omega}}\|}{2} + \left(\cos\frac{\|\mathring{\boldsymbol{\omega}}\|}{2} - \mathrm{Sinc}\,\frac{\|\mathring{\boldsymbol{\omega}}\|}{2}\right)\boldsymbol{n}_{\mathring{\boldsymbol{\omega}}}\boldsymbol{n}_{\mathring{\boldsymbol{\omega}}}^T \end{pmatrix}, \tag{5.91}$$

very similar the the corresponding dimensionfull jacobian of equation (5.28).

The update and feature initialization step remain the same of the original algorithm in which every variable is substituted with the corresponding dimensionless one. A description of the geometric meaning of the dimensionless state is provided in Section 4.2.

The dimensionless reformulation brings two benefits to the algorithm. First of all, the system and the equations are a little bit simplified, so that, during implementation, some mistakes can be avoided. Moreover, the most important benefit is that the scale of the algorithm can be inserted in the state of the EKF filter and consider as a stochastic variables. If the scale is initialized as usually, i.e., with observing an object of known dimension, by inserting it in the EKF state the scale drift can be estimated, since it is related to the value of the variance of the state. On the other hand, when scale is inserted in the EKF state, its value can be filtered if some dimensional measurements are available, such as the wheels odometry of the robot.

Chapter 6

The Proposed Solution

We're just enthusiastic about what we do

Steve Jobs

Real time constrained computer vision applications are limited in both maximum and minimum admissible frame rates. Low frame rate will seriously hamper the performance of the algorithm, simply because too sequential frames will be too much different to measure something from that, e.g., if the majority of tracked points comes out of the field of view of the camera during the period of the camera, then tracking this points will be impossible and the algorithm will fail. On the other hand, if the frame rate is too high, all the computations required by the algorithm may be not performed when the new frame incomes, and the algorithm will fail too. Usually a normal camera's frame rate is about $15Hz$ or $30Hz$, so that all computations required by the algorithm must be performed in $60ms$ or $30ms$. Mono-SLAM makes no difference and, in this very strict time window several operations should be performed, such as matching of tracked points, finding new point to track if needed, performing the Extended Kalman Filter algorithm (as seen in Chapter 5) and moreover results must be visualized. For this reason it is very important to speed up the algorithm as much as possible.

Hence implementation and practical solutions to perform operations are a crucial aspect of the goodness of the algorithm, and it is very important to adopt some special solutions in order to make the algorithm to perform better. On the other hand, the algorithm requires also to be robust, i.e., it needs to work also in not ideal conditions. Robustness is another crucial aspect of practical implementation, and it often requires to use more complex algorithm and more computational resources than the minimum ones required by theory. This additional computation power required is due, for instance, to perform consistency checks on updated state or to rejects outliers in measurements.

This Chapter aims at describing and discussing the solutions and the approaches used to implement the algorithm introduced in the previous Chapters, and at describing the obtained results. The Chapter is organized as follows: Section 6.1 describes the frameworks, libraries and devices used to build up the developed algorithm. Section 6.2 reports some important solutions used to improve robustness of the algorithm, including a novel approach to handle motion blur here proposed for the first time. Section 6.3 describes the architecture of the implemented algorithm. Section 6.4 describes the results obtained by the developed algorithm, underling open issues still affecting the approach.

6.1 Frameworks and Libraries

Due to the complexity of the various approaches, algorithms and systems, nowadays it is impossible for everyone to build any working software without using some yet developed libraries and frameworks. The proposed solution are developed by using three important open source projects, that are the *Robotic Operating System* (ROS), the *OpenCV library* and the *Eigen Library*.

6.1.1 ROS: the Robotic Operating System

ROS[1] is an open source framework that provides libraries and tools to help software developers create robot applications, such as hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

ROS is geared toward a Unix-like System, and nowadays the only supported OS is the Linux Operative System. A ROS system works by using two important entities: *nodes* and *topics*. A node is an executable application within ROS, it can be implemented using C/C++ or Python languages and works as a common program within linux system. Since a robot system often requires high level solution to communication, ROS provides a useful way to manage inter node communication, i.e., the *topics*. A topic is simply a file that can be written by a node (the publisher) and can be read by each node that has been subscribed to it.

Nodes and topics allow ROS to be a perfect platform for cloud robotics and multi-robot systems, since nodes running in different machines can communicate each-other using topics. A particular node, labeled as **roscore**, is in charge of collect the various topics and share them throw nodes.

In this work, ROS is the base framework supporting the code. In fact, the Mono-SLAM algorithm is implemented as a ROS node subscribed to a topic where camera handler node publishes frames and publishing the output information (camera path, features in 3D space, covariances etc.) on various topics.

Some already developed ROS nodes has been used in the implementation of the system, such as **gscam**[2] and **camera1294**[3], which are nodes able to capture and public the images coming from an USB or firewire camera respectively, and **rviz**[4], which is a node very useful for visualization purpouse.

6.1.2 OpenCV Library

OpenCV[5] (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real-time computer vision. It focuses mainly on real-time image processing and provides several modules to perform features extraction and matching, motion tracking, camera calibration etc. The library provides C/C++ and Python interface, and it is supported by ROS.

In this work, OpenCV has been used mainly to perform camera calibration, as described in Section 2.6 and as tool for input/output and visualization purpose in the implementation of the algorithm. In particular, it is used to read the new frame coming from the camera node, draw all stuff on the image useful to visualize what algorithm is doing, e.g., the probability region in the

[1] `http://www.ros.org`

[2] `http://wiki.ros.org/gscam`

[3] `http://wiki.ros.org/camera1394`

[4] `http://wiki.ros.org/rviz`

[5] `http://opencv.willowgarage.com`

next frame and the position of the feature in the current one, and to perform blurring and patch matching operations. Please note that the algorithm used to match patches has been written *ex novo* since OpenCV does not allow to perform cross correlation on an ellipsoid region, it actually allows to perform cross correlation on a rectangular image. A very good description of the library is provided in [49].

6.1.3 Eigen Library

The open source Eigen[6] library is an high-level C/C++ library of template headers for linear algebra, matrix and vector operations, numerical solvers and related algorithms. In this work it has been used to represent matrices and vectors required to EKF algorithm and to perform operations between this entities. In particular, Eigen provides functions for computing eigenvectors and eigenvalues of a matrix (very useful to draw the ellipsoids in 2D and 3D spaces), for perform matrix operations using fast algorithm, such as Cholesky decomposition (please refer to [50] for more details) which helps to speed up the inversion of a positive definite matrix.

6.2 Improve Robustness

Robustness is the ability of a system to continue to operate despite abnormalities, i.e., the ability of not failing in not ideal conditions. As seen in the previous Chapters, the Mono-SLAM main algorithm and the several approaches used to construct it require some important constraints about the surrounding environment that often cannot be satisfied in real applications. For instance, the EKF filter suppose the noise affecting the system to be gaussian while, of course, it has not a gaussian distribution In particular, noise affecting the state is a model of the unmeasurable acceleration that the robot or the user impose to the camera. Moreover, patch matching supposes the patches searched to be extracted by the image while, in reality, the patches comes from previous frames. In addition, the Mono-SLAM algorithm requires the environment to be static and without occlusions, that are obviously constraints that cannot be satisfied in real applications.

Some simple solutions can be implemented to improve robustness. For instance, matching is accepted only if the best cross correlation score is above a certain threshold. This simple expedient is able to handle occlusions when they occur. On the other hand, more complex approaches must be exploit to handle serious problems that occurs in real applications.

This Section is devoted to present two important algorithm used to handle outliers in patch matching and to improve performance of the main algorithm when the camera is moving quickly. The second issue is solved using a novel approach to handle motion blurring in the Mono-SLAM algorithm.

6.2.1 1-Point RANSAC for EKF: Handling Outliers

One of the most important causes of failure in Mono-SLAM is due to the outliers in measurements, that occur when a feature is not correctly matched in a new frame. If not handled, the mismatch is filtered by the algorithm and the updated state may be corrupted. To avoid this issue, robust approaches check the consistency of data against the global model assumed to be generating the data themselves, and discard as spurious any of them that do not fit into it. Among several solutions

[6] `http://eigen.tuxfamily.org`

to perform the above mentioned tasks, the *Random Sample Consensus* (RANSAC) algorithm [51] is one of the most used, especially in the field of computer vision.

RANSAC

RANSAC is an iterative method to estimate parameters of a mathematical model from a set of observed data which contain outliers. It is a non-deterministic algorithm, i.e., it produces a reasonable result only with a certain probability. This probability increasing as more iterations are allowed.

RANSAC algorithm requires as input the mathematical model and the data (which may contain outliers) to fit the model itself. In the first step, RANSAC builds up hypotheses for the model parameters by randomly selecting a certain number of measurements (the minimum necessary to estimate all the parameters of the model). For instance, if the task is fitting some points in 2D space with a line, two points randomly chosen are used to generate hypotheses. To each hypothesis a *consensus set* is associated, i.e., the set of all measurements that fit the estimated model well. For instance, again referring to the example of line fitting, the consensus set can be the set of all points far from the line (the hypothesis) under a given threshold. Hypotheses involving one or more outliers are assumed to receive small consensus, hence, the algorithm chooses the hypothesis with bigger consensus set.

The number of hypotheses n_h necessary to ensure that at least one spurious-free hypothesis has been tested with probability p can be computed as follows:

$$n_h = \frac{\log(1-p)}{\log\left(1-(1-\epsilon)^m\right)}, \tag{6.1}$$

where ϵ is the outlier ratio and m the minimum number of data points necessary to instantiate the model. The usual approach is to adaptively compute this number of hypotheses at each iteration, assuming the inlier ratio is the support set by the total number of data points in this iteration

After that the best hypothesis has been selected, its consensus set is considered the set of inlier of the model, and its elements are used to estimate the model.

1-Point RANSAC for EKF

In [52], Civera et al. have proposed an integration of the RANSAC algorithm in the EKF filter. The peculiarity of the algorithm is that it includes the prior probabilistic information from the EKF in the RANSAC model hypothesize stage. In Algorithm 6.1 it is provided a version of the 1-Point RANSAC in pseudo-code, the following discussion is referred to it.

1-Point RANSAC is made of two big parts, the first one is in charge at selecting the *low-innovation inliers* while the second one the *high-innovation inliers*.

The *low-innovation inliers* selection is performed by executing the RANSAC algorithm using a single feature (point) to generate hypotheses and construct the best consensus set. Unlike the classical RANSAC algorithm, the hypotheses are generated using also the information given by the EKF filter. In particular, a new hypothesis is generated by performing the standard EKF update with only the selected point as measurements. After that, consensus set are built up by collecting all measured points which are inside a certain probability ellipsoid (given by a threshold) centered in the predicted measurements obtained with the computed hypothesis.

Low-innovation inliers are elements of the consensus set of the best hypothesis after RANSAC execution. They are assumed to be generated by the true model since they are at a small distance from the most supported hypothesis. The remaining points could be both inliers and outliers, even if they are far from the supported hypothesis. This is due to the fact that the point chosen to

Algorithm 6.1: 1-Point RANSAC for EKF

Input: $\boldsymbol{x}_{k|k-1}$, $\boldsymbol{\Sigma}_{k|k-1}$, $\boldsymbol{z}_k$
Output: $\boldsymbol{x}_{k|k}$, $\boldsymbol{\Sigma}_{k|k}$

```
// Execution of RANSAC algorithm to select low-innovation inliers
```
$\boldsymbol{z}^{LI} \leftarrow []$
for $i \leftarrow 2$ ***to*** n_h **do**
  $z_i \leftarrow$ `selectRandomMatch` $(\boldsymbol{z}_k)$
  $\boldsymbol{x}_i \leftarrow$ `EKF_Update` $\left(\boldsymbol{z}_k, \boldsymbol{x}_{k|k-1}, \boldsymbol{\Sigma}_{k|k-1}\right)$ `/* Note: NO covariance update */`
  $\boldsymbol{h}_i \leftarrow$ `EKF_Predict` $(\boldsymbol{x}_i)$
  $\boldsymbol{z}_i^{LI} \leftarrow$ `generateConsensusSet` $\left(\boldsymbol{z}, \boldsymbol{h}_i, \chi^{LI}\right)$
  if `size` $\left(\boldsymbol{z}_i^{LI}\right) >$ `size` $\left(\boldsymbol{z}^{LI}\right)$ **then**
    $\boldsymbol{z}^{LI} \leftarrow \boldsymbol{z}_i^{LI}$
    $\epsilon \leftarrow 1 - \frac{\texttt{size}(\boldsymbol{z}^{LI})}{\texttt{size}(\boldsymbol{z}_k)}$
    $n_h \leftarrow \frac{\log(1-p)}{\log(1-(1-\epsilon))}$
  end
end

```
// Partial Update using low-innovation inliers
```
$\left[\boldsymbol{x}_{k|k}, \boldsymbol{\Sigma}_{k|k}\right] \leftarrow$ `EKF_Update` $\left(\boldsymbol{z}^{LI}, \boldsymbol{x}_{k|k-1}, \boldsymbol{\Sigma}_{k|k-1}\right)$

```
// Self compatibility check to select high-innovation inliers
```
$\boldsymbol{z}^{HI} \leftarrow []$
for ***each*** $z_i^{HI} \in \boldsymbol{z}_k / \boldsymbol{z}^{LI}$ **do**
  $[\boldsymbol{h}_i, \mathbf{S}_i] \leftarrow$ `EKF_PointPrediction` $\left(\boldsymbol{x}_{k|k}, \boldsymbol{\Sigma}_{k|k}, i\right)$
  if $(\boldsymbol{z}_i - \boldsymbol{h}_i)^T \mathbf{S}_i^{-1} (\boldsymbol{z}_i - \boldsymbol{h}_i) < \chi^{HI}$ **then**
    `add` z_i `to` $\boldsymbol{z}^{HI}$
  end
end

```
// Partial Update using high-innovation inliers
```
$\left[\boldsymbol{x}_{k|k}, \boldsymbol{\Sigma}_{k|k}\right] \leftarrow$ `EKF_Update` $\left(\boldsymbol{z}^{HI}, \boldsymbol{x}_{k|k}, \boldsymbol{\Sigma}_{k|k}\right)$

generate the best hypothesis could not contain all the information necessary to correctly update the state. For instance, it has been explained that distant points are useful for estimating camera rotation, while close points are necessary to estimate translation (see Chapter 5). In the RANSAC hypotheses generation step, a distant feature would generate a highly accurate 1-point hypothesis for rotation, while translation would remain inaccurately estimated. Other distant points would, in this case, have low innovation and would vote for this hypothesis. But as translation is still inaccurately estimated, nearby points would presumably exhibit high innovation even if they are inliers.

High-innovation inliers are selected after a partial EKF update step involving only the low-innovation inlier selected by RANSAC. After this partial update, most of the correlated error in the EKF prediction is corrected and the covariance is greatly reduced. This high reduction will be exploited for the recovery of high-innovation inliers: as correlations have weakened, consensus for the set will not be necessary to compute and individual compatibility will suffice to discard inliers from outliers. For each points discarded by the RANSAC algorithm, it is checked if it is individually compatible by veryfing whether it lies in a fixed-sized multiple of its covariance probability region or not. If the check is passed, the point is added to the high-innovation inliers set. After that all points are checked, a second partial update involving the high-innovation inliers is performed.

6.2.2 Handling Motion Blur

In some cases, when performing patch matching, camera movements may be so pronounced, and so it would be hard for any classifier to recognize a feature even if it is known, with a reasonable uncertainty, where the feature itself is located. This problem is caused by motion blurring. To give an example, imagine that the feature to be recognized is represented by a point, e.g., a star in the sky as in Figure 6.1a, and the input of the SLAM algorithm is captured by a camera translating very fast. In this case, the illusory movement of the considered scene will be too large even if the camera is recording with a high temporal resolution (i.e., the shutter time is small). So, it is probable for the expected point to be actually recorded as it were a straight line, as in Figure 6.1b. In such a case, it will be impossible to recognize it.

Classical and related works in the field of visual navigation always consider negligible the shutter time and frames to be tracked as perfectly on focus. Here it is presented a novel approach able to overcome this limitation, thus allowing an easy identification of blurred features.

One possible approach to solve the motion blurring problem related to Mono-SLAM could be to pre-process each recorded frame by a deblurring filter, to recover the real latent image behind it. After that, it could be possible to identify and match features even using regular and sharp patches. However, this approach will be overly time consuming, while Mono-SLAM algorithm is required to fulfill real-time constraints. In addition, obtained results may be unacceptable, since real blur effects due to camera movements (or shake) may hardly be perfectly modeled. Similar considerations hold on even if considering narrowing the deblur problem to the regions in the frame in which features are expected to be found.

In this Master Thesis, problems related to blur effect are addressed from a novel point of view. In fact, they have been studied in order to ease the task of Mono-SLAM, and at the same time improving its reliability. The approach proposes to blur the patches used to recognize features, in order to line up observed features and their expected appearance. This novel approach presents various advantages:

- considered patches are very small and in such tight spaces the hypothesis of spatial-invariance can be applied without loss of generality;

(a) Original image (b) Blurred image

Figure 6.1: Example of motion blurring. Sharp (a) and blurred (b) version of a star in the sky.

- the operation of convolution modeled by (3.1) can be performed very quickly;
- information about rotational and translational speed given by the EKF are used to deduce the PSF, and so to calculate the expected feature shape through a simple convolution – in the same way as it is expressed in (3.1).

The solution here presented has been developed with the Student Giuseppe Airò Farulla. In his Master Thesis [53] he studies and design solutions for restoring blurred images.

Blur kernel prediction

To predict the kernel used to blur (when needed) patches, the following assumptions have been made: (i) the camera capturing time T is constant and known, (ii) the blur kernel is linear. The second hypothesis is perfectly suitable for small patches.

The matrix representing the PSF is deduced for each frame from information about speed given by the EKF filter. For the subsequent blurring task the kernel is identified by a sparse matrix, with non-zero values identifying a straight line representing the direction of camera movements. The length and direction of this line is calculated by computing the 3D displacement of the interest point $\boldsymbol{p}$ in the capturing time interval T using the estimated linear and angular speed as follows

$$\boldsymbol{p}^* = \begin{pmatrix} \mathbf{R}(\boldsymbol{\omega} T) & \boldsymbol{v} T \\ 0 \cdots 0 & 1 \end{pmatrix} \boldsymbol{p} \qquad \rightarrow \qquad \Delta \boldsymbol{p} = \boldsymbol{p}^* - \boldsymbol{p}, \tag{6.2}$$

and projecting it on the image plane using the projection function $\underline{h}(\cdot)$ as follows

$$\Delta \underline{p} = \underline{h}(\boldsymbol{p}^*) - \underline{h}(\boldsymbol{p}), \tag{6.3}$$

see also Figure 6.2. $\mathbf{R}(\boldsymbol{\omega} T)$ is the rotation matrix associated to the angle $\boldsymbol{\omega} T$. Please note that, if considering the dimensionless reformulation, the T parameter should be replaced by the ratio between T and ΔT.

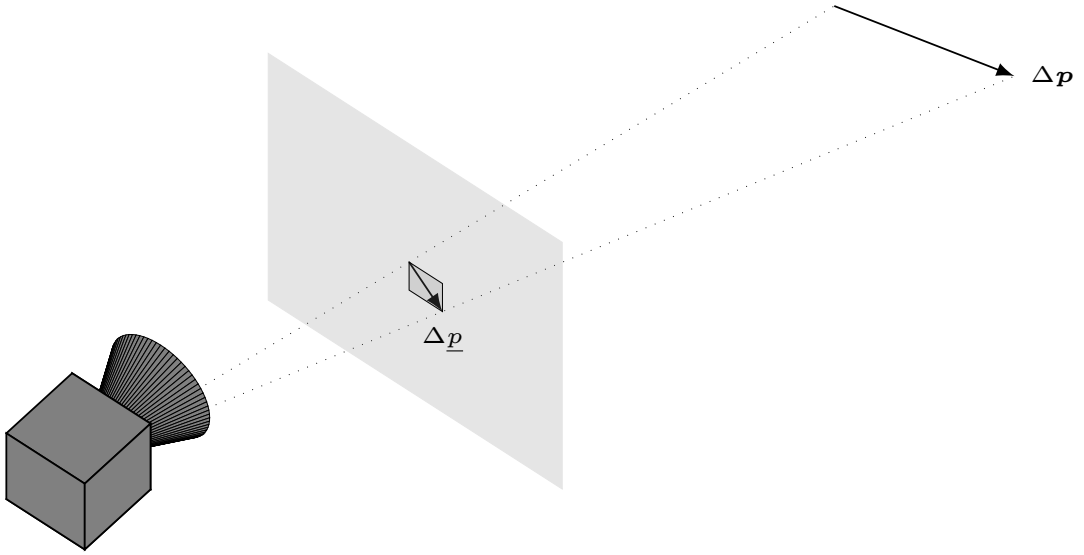


Figure 6.2: Kernel computation. The displacement of the considered point $\Delta \boldsymbol{p}$ during the shutter time T is projected on the image plane. The blurring kernel (the dark gray rectangular) is the smallest matrix containing entirely the projection $\Delta \underline{p}$.

The PSF is the smallest possible matrix containing the $\Delta \underline{p}$ and presenting non-zero entries along $\Delta \underline{p}$ direction. Non-zero values have to be normalized in order to make their sum equal to one; this will ensure that the convolution process will not affect the luminosity of the patch (otherwise the result would be brighter or darker).

Blurring prediction algorithms

Algorithm 6.2 presents the pseudo-code of the operations that are used in the proposed solution to predict the blurred patch. It requires as input information contained in the predicted state $\boldsymbol{\mu}_{k+1|k}$. It performs the following operations: $\Delta \underline{p}$ is computed as in equation (6.3) and its length is compared with a threshold to verify if the blurring is not negligible (line 2). If blurring is not negligible (line 3), the patch is blurred (lines 4, 5) and the result of this operation is stored into p'; otherwise, p' will be equal to p.

6.3 Implementation

The proposed solution has been developed according to the architecture described in Figure 6.3. The camera used as input device sends each new captured frame to the *capture and preprocess* block, that allocates the captured frame f_k, processes it in order to remove noise and to reduce the amount of data to be processed (e.g., if the image is too big, its size is reduced) and sends it to the *Mono-SLAM* main block, that is in charge of processing the frame to reconstruct the camera motion. The output of the *Mono-SLAM* main block is the estimation of the state of the system.

Algorithm 6.2: Blur prediction

Input: original patch p, $\boldsymbol{\mu}_{k|k}$
Output: predicted patch p'
$p' = p$
$\Delta\underline{p} \leftarrow$ `compute_dispacement` $\left(p, \boldsymbol{\mu}_{k|k}\right)$
if $\|\Delta\underline{p}\| > K\_MIN\_SIZE$ **then**
  $PSF \leftarrow$ `compute_kernel` $\left(\Delta\underline{p}\right)$
  $p' \leftarrow$ `blur_patch` $\left(p, PSF\right)$
end

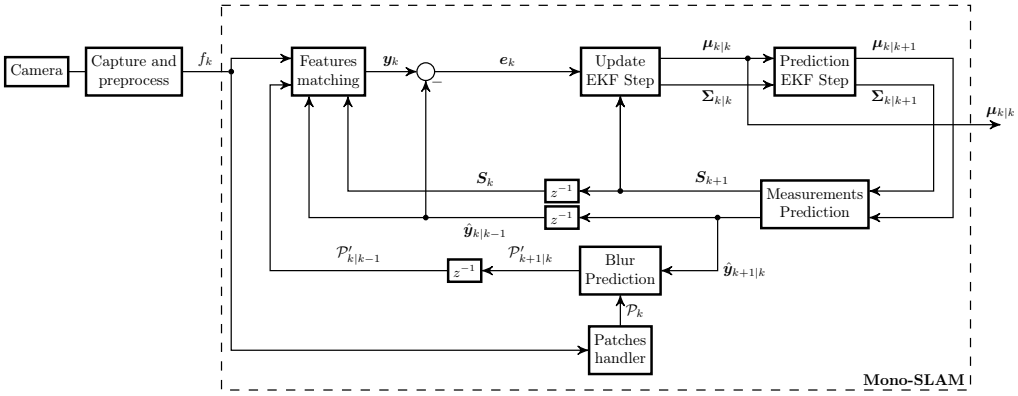


Figure 6.3: Architecture of the proposed solution. The camera used sends each new captured frame to the *capture and preprocess* block, that allocates the captured frame f_k and sends it to the *Mono-SLAM* main block, that returns the estimation of the state of the system.

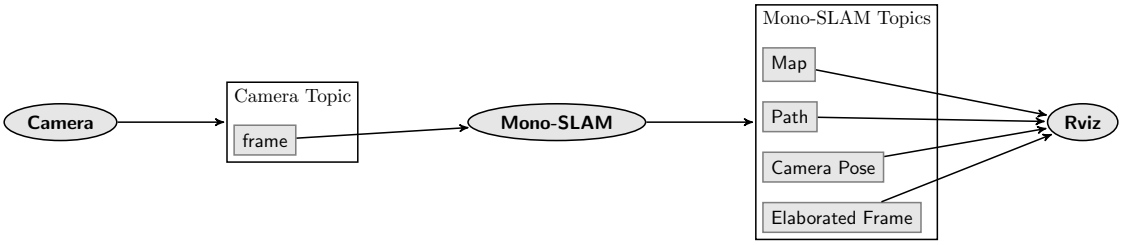


Figure 6.4: ROS graph of the developed solution. The **camera** node publish the video stream of a camera on a topic that is read by the **Mono-SLAM** main node. **Mono-SLAM** elaborates informations and publishes four topics with the obtained results. In the Map topic the 3D position of each features, with its covariance, is published. Path contains the trajectory of the camera, Camera Pose the actual pose of the camera, with its covariance, and Elaborated Frame contains the input frame with drawn some information such as features' positions. The **rviz** ROS node is used to visualize the obtained results.

(a) A standard frame (b) 1-Point RANSAC algorihtm (c) Prediction

Figure 6.5: Some frames elaborated by the algorithm. The algorithm draw on the current frame the position of the correct matched patch and the ellipsoids where each patch should lies in the next frame. (a) shows a standard elaborated frame. (b) shows the 1-Point RANSAC algorithm working: blue patches are low-innovation inliers, green patches are high-innovation inliers and red patches are rejected measurements. (c) shows enhances prediction.

The *features matching* block is in charge of matching the predicted features contained in the set $\mathcal{P}'_{k|k-1}$ in the new frame. For performance reasons, as seen in Chapter 5 matches for all features are searched in the areas (modeled as an ellipsoid) in which there is high probability of finding them. This block uses information contained in the predicted measurements $\boldsymbol{y}_{k|k-1}$ and its related covariance $\mathbf{S}_k$. Its output is the measures vector $\boldsymbol{y}_k$.

When $\boldsymbol{y}_k$ has been computed, the algorithm performs the standard EKF filter. Firstly, the innovation vector $\boldsymbol{e}_k = \boldsymbol{y}_k - \boldsymbol{y}_{k|k-1}$ is computed; secondly, the *update* and *prediction* steps are performed to build the updated state $\boldsymbol{\mu}_{k|k}$ with related covariance $\boldsymbol{\Sigma}_{k|k}$ and the predicted state $\boldsymbol{\mu}_{k+1|k}$ with related covariance $\boldsymbol{\Sigma}_{k+1|k}$, respectively.

The predicted state is used to build, for each patch in the set $\mathcal{P}_k$, the set of blurred patches $\mathcal{P}'_{k+1|k}$ expected in the next frame by iteratively running the Algorithm 6.2 for all elements in the set $\mathcal{P}_k$.

The *patches handler* block is encharged of managing the $\mathcal{P}_k$ set. In particular, it is used to find the best features to track at the very beginning (i.e., when the first frame is being acquired) and when old patches have been deleted because they are no more useful (e.g., when they go outside the framing) and must be replaced.

Finally, note that z^{-1} represents a delay equal to the inverse of the camera frame rate, useful to store predicted information until a new frame has been acquired.

The algorithm has been completely implemented in ROS as a node labeled **Mono-SLAM**. The node is subscribed to a topic publishing the video stream coming from a camera, published by **camera1394** or **gscam** ROS nodes. **Mono-SLAM** publish information about its computation results that can be used as input for other nodes. In the experiments, the output of **Mono-SLAM** are visualized using **rviz** ROS node. This is visualized in Figure 6.4.

Some examples of the output of the program are visualized in Figures 6.5 and 6.6. Figure 6.5 shows some frames elaborated by the algorithm. Several information are visualized in that frame:

- correct matched patches are highlighted by drawing blue rectangles,
- features selected as low-innovation inliers are marked in blue,
- features selected as high-innovation inliers are marked in green,

- features discarded by the 1-point RANSAC algorithm are marked in red,
- the high probability region in which each feature should lie in the next frame are highlighted by an ellipsoid.

Moreover, Figure 6.6 shows some examples of visual output provided by **rviz** ROS node. In the 3D environment, it is visible the camera moving according to the output of the algorithm and the map reconstruct by the algorithm. Each 3D feature is visualized together with its covariance region. The green covariance refers to features in inverse depth coding while the red ones refer to features in euclidian representation. Please note that, due to limits of **rviz**, the covariance of the inverse depth features is represented by projecting the features in the euclidian space, i.e., always as an ellipsoid, instead of correctly represent that covariance in the inverse depth space, i.e., a generic conic shape.

6.4 Results

In this Section, the results obtained by the implemented solution are presented and discussed. Several experiments have been done to evaluate the solution. First of all, some experiments have been done to verify the correct working of single modules of the algorithm. After that, two main experiments have been done to evaluate the whole algorithm.

All the experiments done are here summarized: (i) to visualize the correct working of the blur prediction module, the algorithm has been launched with a very big patch size; (ii) to evaluate robustness, to experiments have been done in which moving objects (in particular an hand and a person) were in the environment; (iii) camera motion was imposed by an hand to evaluate the correct working of the algorithm in case of camera freely moving in 3D space; (iv) to evaluate the correct working of the algorithm in big environments, two experiments have been performed using data available from the Rawseeds projects, which provides data and ground truth to evaluate robotic navigation in big environments.

6.4.1 Experiments Setup

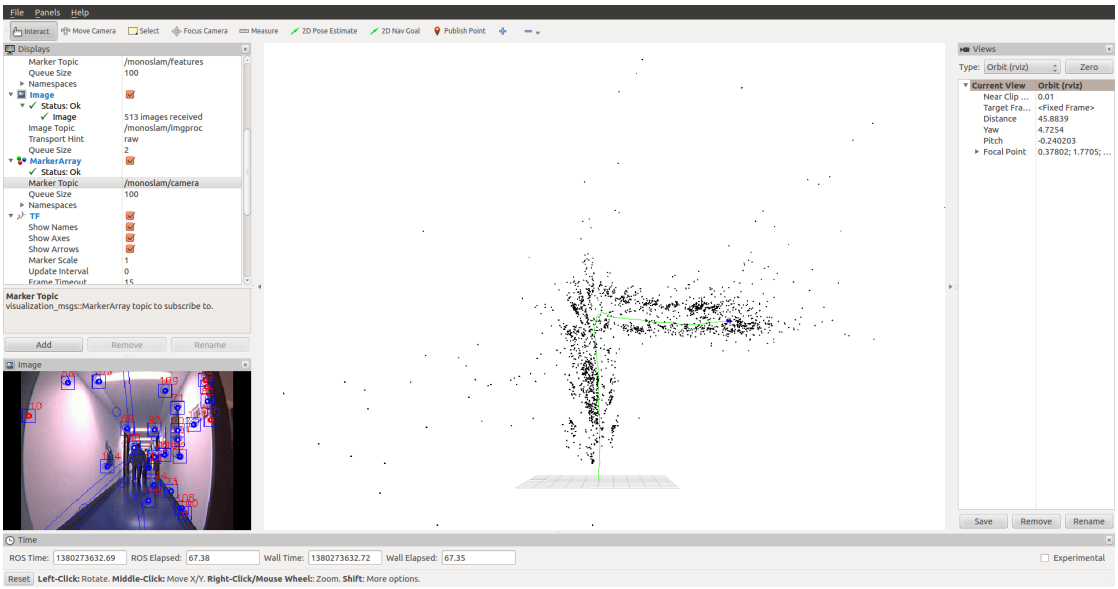
For reproducibility purpose, experiments has been done using video streams captured using the **ROSbag**[7], which provides an useful tool for recording from and playing back to ROS topics. In this way, the same experiment can be performed several time in the same conditions. Moreover, to evaluate the algorithm in big environments, the dataset available in the Rawseeds Project[8] has been used.

A configuration file was build up to set the various parameters of the algorithm without the need of recompiling the code. Different values for each parameters has been tested. In Table 6.1 some examples of the chosen parameters are reported. Unfortunately, this configuration need to be changed according to the camera system and the type of motion imposed to the camera.

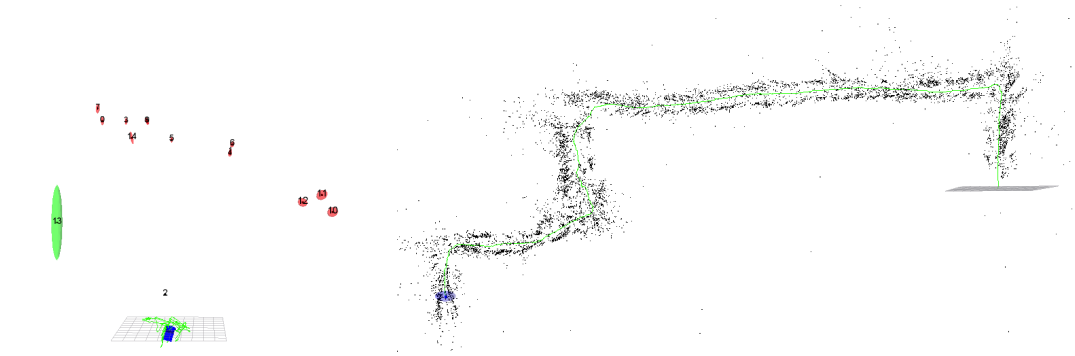
Experiments performed in RRG lab have been done using a firewire industrial camera. Since available rovers do not have a firewire connection, that experiments have been performed with the camera connected to a desktop computer.

[7] `http://wiki.ros.org/rosbag`

[8] `http://www.rawseeds.org/home/`



(a) A screen-shoot of **rviz**



(b) Features covariances

(c) A map

Figure 6.6: The **rviz** ROS node is able to give to users a visual output of the data processed by the algorithm. (a) shows a screen-shoot of the complete **rviz** environment: the program shows the images elaborated by Mono-SLAM and the map builded by the algorithm, including also the pose of the camera and its trajectory (in green). (b) shows the same example zoomed, in which it is possible to see the that to each features it is drawn also the ellipsoid representing the covariance of the features position in 3D space. Note that green ellipsoids are associated to inverse depth feature while red ellipsoids to euclidian features. (c) shows a bigger map reconstructed by the algorithm.

Table 6.1: Two cases of configuration parameters for the algorithm. Case 1 refers to the experiments done using the camera available in RRG at Polito. Case 2 refers to the experiments done using the Rawseeds dataset.

parameter	**Case 1**	**Case 2**	**Definition**
Mono-SLAM paramters			
σ_v	0.02	0.01	eq. (5.18), page 68
σ_w	0.01	0.005	eq. (5.18), page 68
ρ_0	0.1	0.4	eq. (5.68), page 80
σ_{ρ_0}	0.25	0.25	eq. (5.72), page 80
σ_R	2	1	eq. (5.49), page 73
1-Point RANSAC paramters			
χ^{LI}	$2\sigma_R$	$2\sigma_R$	Algorithm 6.1
χ^{HI}	1	1	Algorithm 6.1
camera paramters			
f_x	563.21765	194.8847	Chapter 2
f_y	558.45293	194.8847	Chapter 2
u_0	347.75115	172.1608	Chapter 2
v_0	246.19144	125.0859	Chapter 2
k_1	-0.45720	-0.3375	Chapter 2
k_2	0.30980	0.1392	Chapter 2
k_3	-0.13950	-0.0289	Chapter 2
p_1	-0.00265	0.0004	Chapter 2
p_2	0.00078	0	Chapter 2
other paramters			
patch size	21×21	21×21	Chapter 3
K_MIN_SIZE	3	3	Algorithm 6.2
T	0.2	0.2	eq. (6.2), page 91

Figure 6.7: Results of motion blur predicition. Each image shows the original patch (left), the predicted patch affected by motion blur (center) and the real matched patch (right).

6.4.2 Evaluation of blur prediction

The blur prediction module is in charge of predicting the motion blur of a given patch to better perform the patch matching in the next frame. To evaluate this module, an experiment has been done in which very big patches are used (101×101 pixels), in order to better visualize the results of the blur prediction. The camera is moving very quickly in order to enhance the motion blur, and some patches are captured. Figure 6.7 shows some examples of the output of this module.

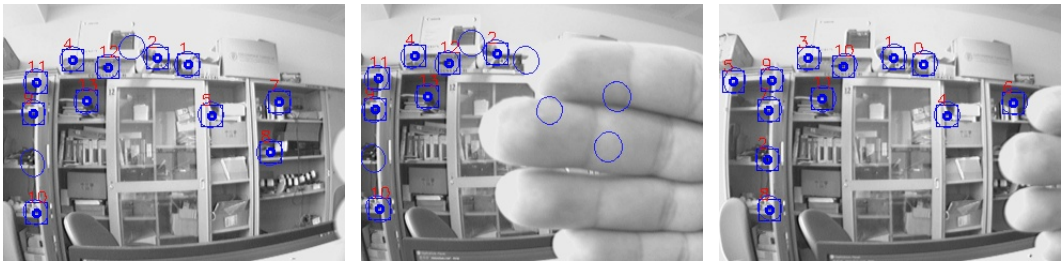
From the images it is easy to visualize that the algorithm is able to predict the motion blur which is very similar to the real blurring of the next frame. Note that the result of the blurring module is strongly dependent from the parameter T of the algorithm, modeling the aperture time of the camera. Obviously, the bigger this number is, the more pronounced will be the results of the blurring filter. In this implementation, several tests has been done to find the best value vales for this parameter. According to the results, $T = 0.2$ has been chosen.

6.4.3 Evaluation of the algorithm in non ideal environment

Two experiments has been done to verify the correct working of the algorithm in non ideal environment, i.e., dynamic environment and environment presenting occlusion.

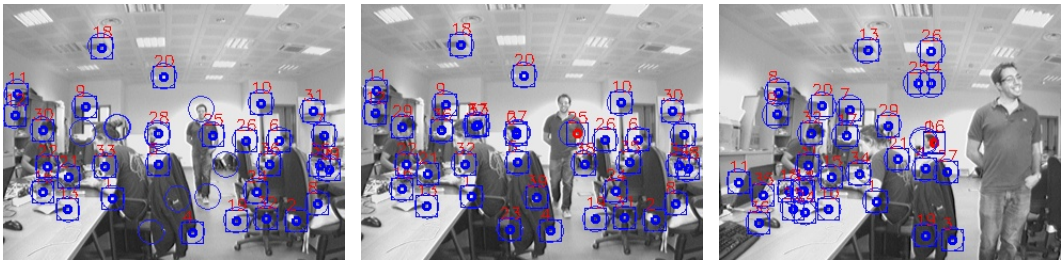
The first one consists in introducing some object in the scene when the algorithm is running, as an hand in case of Figure 6.8. The figure shows how the implemented solution is able to reject points occluded by an object and to perform match again when the object is removed from the field of view of the camera. In the second experiment, an object, initially stationary in the environment, is moved after that some feature belong to it are reconstructed. In Figure 6.9 a person start moving in the filed of view of the camera after that some points on it has been matched by the algorithm. RANSAC algorithm is able to rejects features moving and the matching phase correctly does not match features occluded by it.

In both case the algorithm is able to discard features occluded by the moving object and features moving together which it ignore them when performing update.



(a) (b) (c)

Figure 6.8: Three frames extracted from a complete sequence showing robustness to occlusions of the implemented algorithm. In (a) the algorithm is not perturbed and it is working normally. In (b) a hand is introduced in the scene, note that the algorithm is able to discard patches occluded by the hand. In (c) the hand is removed from the scene and the algorithm works normally again.



(a) (b) (c)

Figure 6.9: Three frames extracted from a complete sequence showing robustness to dynamic environments of the proposed solution. In this sequence, a person is normally working in the environment when the algorithm is running. In all the frames the algorithm is able to discard patches belonging to or occluded by the person. Note in particular that in (b) feature 25 is rejected by RANSAC and in (c) features occluded are completely removed by the state.

6.4.4 Hand held camera experiment

The complete algorithm working has been tested in a general case where the motion of the camera is imposed by an hand. In this case, the are not priors available on the motion, since the camera can move freely in 3D space with 6 DoF (Degree of Freedom). Of course, to respect the initial hypotheses done by the algorithm, the motion of the camera have to be enough slow.

Since it is very difficult to evaluate the algorithm, the motion has been imposed in order to "draw" geometrical shapes within the space, in order to have a visual validation of the correct working of the algorithm. Some visual results are provided in Figures 6.10.

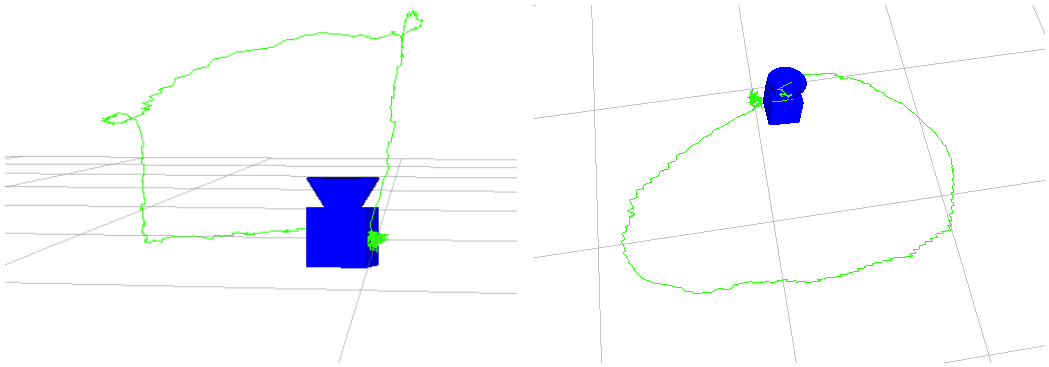


Figure 6.10: Hand held camera moving in 3D space. Two example of shapes reconstructed by the algorithm.

6.4.5 Rover experiments

The latter validation of the algorithm has been carried out using the datasets freely available from the Rawseeds Project[9] [54]. This project provides high-quality multi-sensor datasets, with associated ground truth, of rovers moving in very big environments, both indoor and outdoor. It provides several raw data coming from different sensors, such as GPS, laser scanner, several camera and so on. For each sensor, calibration data are also provided.

The experiments have been carried out using video stream coming from the front camera of the rover in both indoor[10] and outdoor[11] environments.

The trajectory of the robot, given in output by the implemented solution, has been compared with the ground truth available from the datasets, that are: the trajectory obtained from a system based on industrial cameras and visual tags mounted on the robot [55], not available for the all locations of the robot; the trajectory of the robot coming from a standard SLAM algorithm based on laser scanner sensors; and the data coming from the GPS for the outdoor dataset.

For indoor experiments, the *Bicocca_ 2009-02-26a* dataset has been used, where the robot is moving in an indoor dynamic environment. Some results obtained from this dataset are shown in Figure 6.11. From the results it is possible to note that the algorithm is able to perform with good accuracy in small and medium scale (some tens of meters), while in large scale the errors accumulated by the algorithm makes the measured position very far form the real position.

For outdoor experiments *Bovisa_ 2008-10-04* has been used. Here the ground trough is represented by GPS data that, unfortunately, are not available anywhere. Also this second experiment

[9] `http://www.rawseeds.org/home/`

[10] into a pair of building belonging to the Università di Milano-Bicocca, in Milan, Italy, available on internet at the following link `http://www.rawseeds.org/rs/datasets/view/6`

[11] in the Politecnico di Milano campus located in via Durando, Milan, Italy, available on internet at the following link `http://www.rawseeds.org/rs/datasets/view/7`

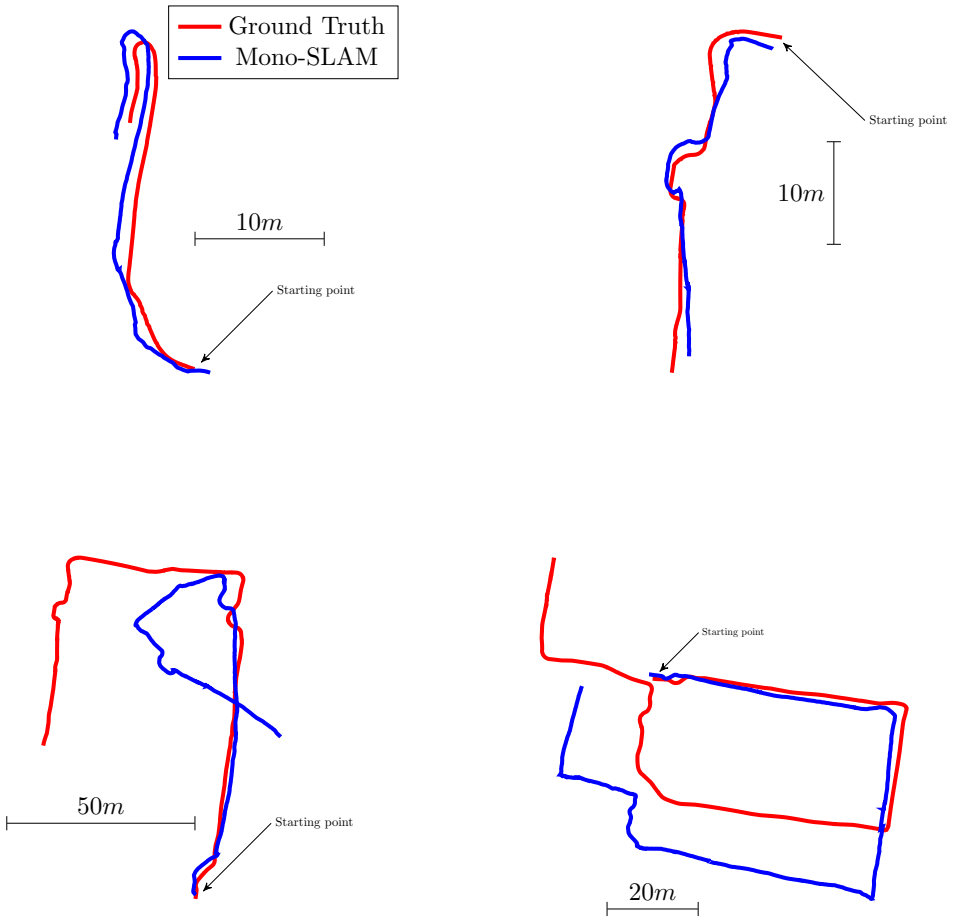


Figure 6.11: Evaluation of the implemented algorithm using *Bicocca_2009-02-26a* dataset. Comparison between the implemented algorithm and the extended ground truth in Milano-Bicocca. Note that the small errors on estimated angles lead to a big difference in trajectories.

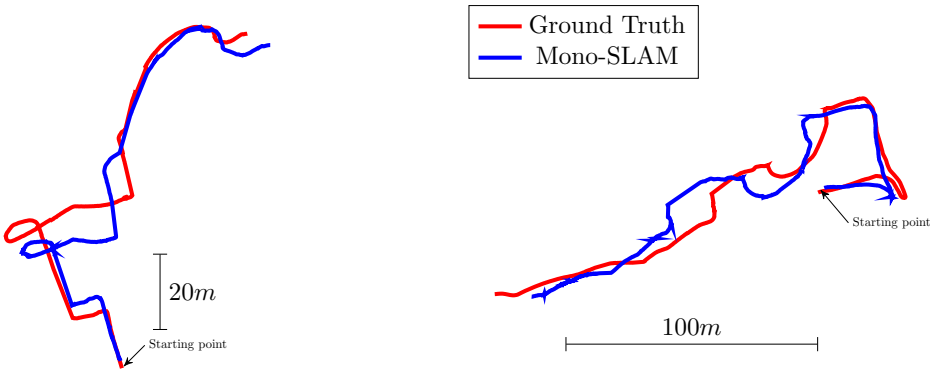


Figure 6.12: Evaluation of the implemented algorithm using *Bovisa_2008-10-04* dataset. Comparison between the implemented algorithm and the extended ground truth in Milano-Bovisa. Note that the small errors on estimated angles lead to a big difference in trajectories.

(see Figure 6.12) shows that the algorithm works very well in small and medium scale, while it drift too much from the ground truth in large scale.

Comparisons between outputs of the implemented algorithm and ground truth from datasets show that the algorithm is able to measure with good accuracy the trajectory of the robot in short periods of time, however, when the algorithm is perform for very long time, the errors accumulate and the measured trajectory become very different from the real one. This phenomena is very common in all algorithm performing SLAM, and are usually corrected thanks to a solution labeled as *loop closing*. Loop closing is a technique able to correct the whole trajectory measured by the SLAM algorithm when the robot notice that it has come back in a place it has already visited. Unfortunately, loop closure is a very difficult task in case of visual navigation, and it has not been jet implemented in this solution. However, implementation of the loop closure solution is still in program as future work.

An other issue affecting the algorithm in big environments are the fact that sometime it is not able to measure correctly the rotation performed by the robot. However, it must be note that this errors appears because in such cases the environment do not presents enough features to correctly perform the navigation or when big moving objects occludes the field of view of the camera.

Chapter 7

Conclusions

The most exciting phrase to hear in science, the one that heralds new discoveries, is not "Eureka!" but "That's funny..."

Isaac Asimov

This Master Thesis provides a detailed analysis of Mono-SLAM along with the underlying methods from image processing in Chapter 3 and camera models in Chapter 2. Chapter 1 provides an introduction of the Master Thesis. In particular, Section 1.1 presents a brief overview on the robot localization problem in general and Section 1.2 presents a state of the art analysis on the visual navigation solution. Some examples about solutions of the reconstruction problem, i.e., model the 3D environment using information coming from 2D images are presented in Chapter 4. The basic idea of the extended Kalman filter was briefly introduced in Section 5.2 and elaborated by the exemplary application Mono-SLAM in the remainder of Chapter 5. The achieved results using an OpenCV and Eigen based ROS implementation were presented in Chapter 6.

This final Chapter aims at discussing the obtained results (presented in Section 6.4) and presenting open issues affecting the solution implemented for this Master Thesis (Section 7.1) and presenting the future work in program to improve the algorithm (Section 7.2).

7.1 Final Remarks and Open Issues

This Master Thesis demonstrates that it is possible to realize a navigation system completely based on vision instead of more expensive technologies, but it shows that this approach needs some refinements to be able to work well in real environments. In fact, the solution is still affected by some open issues, as seen in the previous Chapter.

First of all, it is important to note that the performances of developed algorithm, due to the EKF filter, are very influenced by the parameters of the algorithm, in particular the noise covariance, which is also strongly correlated to the motion imposed to the camera (that is very difficult to estimate *a priori*) and the type of camera itself. For instance, in a wide angle camera, projections of displacements are shorter that projections of the same displacement in a normal camera, due to the difference in the projection function. This phenomena influence a lot the appearance of the motion, and consequently the best suitable covariance, so that it is very hard to select the best covariance to give to the algorithm before its execution.

Secondly, experience with this algorithm has shown that wide angle cameras are more suitable then normal cameras to be used by this solution. In fact, using wide angle cameras allows to observe a very wide part of the environments at the same time: more features can be detected and they can be tracked for more time, with an important benefit to robustness.

Thirdly, the developed algorithm is still not enough robust to work in real environment. In this Master Thesis some solutions to improve robustness have been developed and implemented, and actually the algorithm is able to work in dynamic environment and to manage occlusions. However, there are still some problems when recovering the estimated state of the model when the algorithm fails. In such situations, the covariance of the state increases too much to allow a correct reconstruction of the environment.

Moreover, when the map to be reconstructed is considerately large, the computational time increases too much and the real time constraints are not met. To solve this issue, several solutions can be implemented. The simpler one is to pop out those features in the state having very small covariance but continue to use them for matching and so measurements. In fact, it has been demonstrate that the EKF algorithm is $\mathcal{O}\left(k^{2.4}+n\right)$ of complexity, where n is the size of the state and k is the number of measurements. Hence, using this approach, the dimension of the state is strongly reduced and the algorithm can perform quicker. On the other side, reducing the dimension of the measurements vector is possible by selecting in some way those features having high innovation that can strongly influence the state, and do not perform update on features bringing the same information to the state. Similar approaches has been already proposed in [37–40], but not yet implemented in this solution. A different solution to that problem consists in building up several independent sub maps having small enough dimensions to meet real time constraints; then, linking these maps using a graph that can also be used perform high level map management tasks.

In addition, the key points matching phase is still an open problem in this solution. In their implementation, as seen in Chapter 5, Davison et al. propose to use simple patch matching in order to find correspondence between frames. In this implementation, it has been demonstrated that this solution is not enough robust to allow long execution, since cross correlation is not invariant to rotation and homogeneous transformation. This means that if the camera moves too far from the point in which a patch has been first observed, then the algorithm is not able to match it in the new frame. On the other side, dynamic handling of patches, i.e., if a patched is substituted with his match in a new frame, makes the algorithm to fail since in this case patch matching may drift. For these reasons, more complex solutions have to be exploited. One of that may consist in using more complex features to perform matching, which will strongly impact the computational efforts required. On the other side, a more smart solution may be using the information on the EKF to predict the future appearance of the patches, as the proposed solution already does when predicts blurring. A very similar solution has been proposed in [56].

Finally, the problem of loop closure is still open. Managing loop closure will allow localization to be more precise in very large environment, however, it is still a very difficult task in the field of computer vision. Algorithms able to recognize loop closings already exist, but they are not able to be performed in real time due to their complexity. Despite that, solutions able to recognize and handle loop closings are a good topic for future work.

7.2 Future Works

Solutions and new approaches to handle issues still affecting the proposed solution (such as a way to recognize and manage loop closings) will be developed as soon as possible. Moreover, extend the solution to a scenario in which other sensors are available such as accelerometers magnetometers,

gyroscopes, wheels odometry and other sources of informations is foreseen as future work. In addition, the implementation of the Mono-SLAM algorithm working on an humanoid robot has will be considered when the algorithm developed will be enough robust to perform in real time without fails. In this case, considering the input commands sent to the robot is fundamental, however, an other complex problem is introduced due to the complex motion of the head of an humanoid robot when it walks.

A publication for EWDTS 2013[1] conference call has already been submitted, regarding the novel solution to handle motion blur proposed in Section 6.2.2. About this solution, it has been shown that the proposed approach to handle motion blur works under the condition that the when the patch is first detected, it is sharp. If a new features is initialized with a patch affected by motion blur, the proposed solution does not work. To extend this approach in case of features initialized when robot is moving quickly, it is necessary to correct motion blurring first to apply the proposed algorithm. In this case, blurring correction is more difficult since, when features is initialized, its 3D position is not well defined, hence it is not possible to estimate correctly the displacement of the feature in the capturing, i.e., the blurring kernel. An implementation of a recursive algorithm where the blur correction is performed each step until the patch become sharp is in program.

Finally, more complex applications of the Mono-SLAM algorithm are under consideration. These applications concern multi-robot navigation, in which a team of robot are able to construct a shared map; an extension of the implemented algorithm able to use high-level semantic information; and an extension of the solution able to perform multi camera navigation.

[1] `http://www.ewdtest.com/conf/`

Bibliography

[1] 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*, 29 (6):1052–1067, 2007.

[2] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[3] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[4] Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[5] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[6] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[7] Katrin Pirker, Matthias Rüther, Gerald Schweighofer, and Horst Bischof. Gpslam: Marrying sparse geometric and dense probabilistic visual mapping. In *BMVC*, pages 1–12, 2011.

[8] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[9] Javier Civera, Andrew Davison, and J Montiel. Dimensionless monocular slam. *Pattern Recognition and Image Analysis*, pages 412–419, 2007.

[10] Paul Smith, Ian Reid, and Andrew Davison. Real-time monocular slam with straight lines. In *British Machine Vision Conference*, volume 1, pages 17–26, 2006.

[11] Robert O Castle, Georg Klein, and David W Murray. Combining monoslam with object recognition for scene augmentation using a wearable camera. *Image and Vision Computing*, 28(11):1548–1556, 2010.

[12] Robert O Castle, DJ Gawley, Georg Klein, and David W Murray. Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4102–4107. IEEE, 2007.

[13] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects.

[14] R.F. Riesenfeld. Homogeneous coordinates and projective planes in computer graphics. *IEEE Computer Graphics and Applications*, 1(1):50–55, 1981. ISSN 0272-1716. doi: http://doi.ieeecomputersociety.org/10.1109/MCG.1981.1673814.

[15] Rafael Gonzalez and Richard Woods. *Digital Image Processing (3rd Edition)*. Prentice Hall, 3 edition, 2007. ISBN 013168728X.

[16] David Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003. ISBN 0131911937.

[17] Eugene Hecht. *Optics*. Addison-Wesley, 2001. ISBN 0805385665.

[18] Berthold Horn. *Robot Vision (MIT Electrical Engineering and Computer Science)*. The MIT Press, mit press ed edition, 1986. ISBN 0262081598.

[19] TA Clarke and JG Fryer. The development of camera calibration methods and models. *The Photogrammetric Record*, 16(91):51–66, 1998.

[20] Duane C Brown. Decentering distortion of lenses. *Photometric Engineering*, 32(3):444–462, 1966.

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

[22] Andrew J Davison. Slam with a single camera. In *Workshop on Concurrent Mapping and Localization for Autonomous Mobile Robots, in conjunction with ICRA*, 2002.

[23] Tjalling J Ypma. Historical development of the newton-raphson method. *SIAM review*, 37 (4):531–551, 1995.

[24] Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106–1112. IEEE, 1997.

[25] Zhengyou Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

[26] Y.-L. You and M. Kaveh. A regularization approach to joint blur identification and image restoration. *Image Processing, IEEE Transactions on*, 5(3):416–428, 1996. ISSN 1057-7149. doi: 10.1109/83.491316.

[27] Franklin C Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 207–212. ACM, 1984.

[28] Carlo Tomasi and Jianbo Shi. Good features to track. *CVPR94*, pages 593–600, 1994.

[29] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[30] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[31] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[32] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[33] Herbert Baya, Andreas Essa, Tinne Tuytelaarsb, and Luc Van Goola. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[34] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *Computer Vision–ECCV 2012*, pages 214–227. Springer, 2012.

[35] 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*, pages 2564–2571. IEEE, 2011.

[36] Sven Albrecht. An analysis of visual mono-slam. Master's thesis, Universitat Osnabrück, 2009.

[37] Andrew J Davison. Active search for real-time vision. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 66–73. IEEE, 2005.

[38] Ankur Handa, Margarita Chli, Hauke Strasdat, and Andrew J Davison. Scalable active matching. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1546–1553. IEEE, 2010.

[39] Margarita Chli and Andrew Davison. Active matching. *Computer Vision–ECCV 2008*, pages 72–85, 2008.

[40] Margarita Chli and Andrew J Davison. Active matching for visual tracking. *Robotics and Autonomous Systems*, 57(12):1173–1187, 2009.

[41] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, 2005.

[42] Javier Civera, Andrew J Davison, and J Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.

[43] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[44] Simon L Altmann. Rotations, quaternions, and double groups. 1986.

[45] ERIK I Verriest. On three-dimensional rotations, coordinate frames, and canonical forms for it all. *Proceedings of the IEEE*, 76(10):1376–1378, 1988.

[46] Jay P Fillmore. A note on rotation matrices. *IEEE Computer Graphics and Applications*, 4 (2):30–33, 1984.

[47] Javier Civera, Andrew J Davison, and José Marıa Martınez Montiel. Inverse depth to depth conversion for monocular slam. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2778–2783. IEEE, 2007.

[48] Edgar Buckingham. On physically similar systems; illustrations of the use of dimensional equations. *Physical Review*, 4(4):345–376, 1914.

[49] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'reilly, 2008.

[50] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[51] 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*, 24(6):381–395, 1981.

[52] Javier Civera, Oscar G Grasa, Andrew J Davison, and JMM Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[53] Giuseppe Airò Farulla. Design of a high performance core for removing blur effect on images. Master's thesis, Politecnico di Torino, 2013.

[54] Andrea Bonarini, Wolfram Burgard, Giulio Fontana, Matteo Matteucci, Domenico Giorgio Sorrenti, and Juan Domingo Tardos. Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets. In *In proceedings of IROS*, volume 6, 2006.

[55] Simone Ceriani, Giulio Fontana, Alessandro Giusti, Daniele Marzorati, Matteo Matteucci, Davide Migliore, Davide Rizzi, Domenico G Sorrenti, and Pierluigi Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4): 353–371, 2009.

[56] Nicholas Molton, Andrew J Davison, and Ian Reid. Locally planar patch features for real-time structure from motion. In *BMVC*, pages 1–10, 2004.

List of Figures

List of Tables

List of Algorithms