An interactive approach to SLAM

A thesis submitted to The University of Sheffield for the degree of Master of Philosophy

Erick Noe Amezquita Lucio

Department of Automatic Control and Systems Engineering

October 2016

Acknowledgements

I would like to thank the following persons: My family, to whom I owe simply a lot. My friend Wenceslao, to whom I bothered more than once just to talk. My lecturer Timoteo, which had always great thoughts for me. To the people who believed in me, including Helena, Ingrid and my Supervisors Tony J. Dodd and Jun Liu, together with Manuel Bandala. Sui, Mint, Allen and D'Fersen who are also great adventure companions and never left me behind. As well as all other people who had brought me an smile in my life.. including Lisa KM[*].

> *"Theory is when one knows everything but nothing works. Practice is when everything works but nobody knows why. In our lab, theory and practice go hand in hand: nothing works and nobody knows why."* — Graduate-student Folklore.

[*]Albeit I still keep my shield on... just in case.

Abstract

A novel methodology for Simultaneous Localisation and Mapping (SLAM) is researched, allowing for interactive object selection and description. Current research focusses on SLAM applications, efficiency and larger area mapping. However, often SLAM relies on local, algorithm dependent and automatically selected features which require dense representations for meaningful map generation.

Vision-SLAM has gained interest thanks to affordable image sensors, which also allow flexible feature representations beyond points like curves or planes. However, these are still automatically created and they need to be found in the surroundings. Despite this, indoor applications allow to expect certain objects, e.g. products in shops. These approaches rely on databases for detection, requiring maintenance outside of operating cycle if new objects are to be added. Feature description often relies on the same databases. Some works address this by means of overlaid annotations, relegating an user to input data in a selected feature. However, these remain predefined by the algorithm. This also does not improve SLAM and thus the annotations are limited to be shown according to camera pose.

Therefore, user involvement is minimal in SLAM which can be explained by fully automated trends in engineering. This discards potential user object differentiation and description, ignoring improvements in SLAM through user shaped high-level objects. Active user interaction can provide semantics for better feature representation and overall map description, whereas high-level objects can improve the inner workings of the SLAM algorithm. This includes better feature matching with included high-level object parameter tweaking in real-time, e.g avoiding fully autonomous approaches that incorrectly guess depth in features. Yet, this does not exclude full automation with user input reused in further runs, with improved description and semantics.

Two approaches are presented to achieve user input in SLAM: The first makes use of active contours with an Extended Kalman Filter (EKF), whereas the second uses particle filtering coupled with a novel Hough-Bresenham line detector. These implementations rely on General Purpose computing on Graphics Processing Units (GPGPU) to offer real-time performance. Results show improved stability and semantics compared to a baseline vision-SLAM with inverse depth parametrisation approach, opening a new research branch within SLAM.

Contents

List of Figures

Chapter 1

Introduction

Robots have made their way into everyday life, first by performing stationary, monotonous and automated tasks in industry and recently in domestic or outdoors applications, such as indoor vacuum cleaners, automated driving cars, drones used for leisure or surveillance and even rovers designed to explore the surface of other planets (Han et al., 2016).

Amongst these scenarios there are limitations in robot task execution flexibility, e.g. those that account for safety limit robot dexterity. However, in scenarios were safety is not much of a concern, constrained mobility by lack of robot perception limits automated task adaptability in them. A contrasting case would be an user acting exclusively as an operator, e.g. when using high mobility drones. In this case there is no task automation as the user maintains full control at all times, with the caveat of requiring constant attention (Scholtz, 2002).

As the engineering trend of fully automated devices continue, more robots are expected to dynamically adapt their behaviour in any environment they are deployed. At the same time they are not expected to come into a sudden stop whenever a condition is found, which has not been accounted in its programming. Computational advances slowly allow to remove this limitations by means of artificial intelligence (Kim et al., 2013). However, the concept of environment perception in robots is still needed for autonomous approaches and remains highly sought, as it allows adaptability in them to perform a task.

The above suggest robot self-localisation awareness as an important characteristic, as it can be used to extend the capabilities of a robot. This helps in industry as it allows a robot to increase mobility in assigned areas of operation, by allowing a power assisted tasks to be executed with flexibility of place. On the other hand it provides freedom and efficiency of operation on domestic uses, e.g. an autonomous home vacuum cleaner that creates a map on the fly (Ackerman and Guizzo, 2015).

This motivates Simultaneous Localisation and Mapping (SLAM) research, which allows a robot to sense and map its surroundings through perceived landmarks. Sensor readings are compared against a map stored in memory, if there is not a previously inserted map the robot is limited to assume a starting position when turned on. Using this assumption does not tell the robot where this starting position is nor what it means in a map.

However, even when the robot has been programmed with defined objectives or landmarks in the form of a map, it is still limited in the tasks it can perform. For example, the robot can perform obstacle avoidance using its built map, but assuming that the robot is required to retrieve an object or visit a place, recognition algorithms capable of using the data obtained from the surroundings are needed.

In SLAM, information is often retrieved from the environment in the form of very localised and generic features like points, which if considered alone they are limited in their semantics or meaning and thus require clusters of them to perform recognition (Carlone and Censi, 2014, Li et al., 2014, Reinoso et al., 2014, Deusch et al., 2015). Usually landmark or object information is programmed outside of the robot operation cycle, i.e. offline and thus requires reprogramming for any change not accounted in the surroundings.

Real-time object or place recognition can be of help in these circumstances, robots can make use of such capabilities but some implementations are restricted to an object database. This is built also outside of operation cycle and at the same time limits what objects the robot can recognise (Gross et al., 2008, Civera et al., 2011, Ranganathan and Lim, 2011).

In persons, object segmentation and differentiation is taken for granted compared to robots, as the constant contact with different objects through the course of years allows for accurate object categorisation and recognition, in the moment of perception by the human senses (Su, 2012). Therefore, a research opportunity within robot self-localisation and mapping is missing. This expands perception in robots, involving active user participation beyond just a supervising task. At the same time it removes the need of offline programming or setting information prior robot operation, allowing user driven adaptation in unknown surroundings without going out of operation cycle.

Fusing human recognition capabilities with SLAM would allow the user to be part of the algorithm, making a robot more adaptable to a new environment. This involves real-time user semantics and descriptive input, by means of using assisted tools such as active contours which are capable of tracking objects through user initialisation, or by setting prior information in real-time which is valuable to particle filtering techniques. As such this becomes the main basis for this investigation, which is assisting SLAM through user interaction.

1.1 A Short SLAM Description and Missing Links

Robot SLAM is an intense area of research, as it allows to bring further applications to what was once very constrained or limited in robotic sensing (Durrant-Whyte and Bailey, 2006, Bailey and Durrant-Whyte, 2006). SLAM is a methodology that allows a robot to recognise and match its location in the environment, by means of landmarks acquired from it allowing pose estimation relative to them. In this sense both industrial and domestic robots obtain benefits from perceiving their surroundings, as they can dynamically adjust their movements, avoiding obstacles or find areas in where specific tasks must be carried out.

The basic problem of SLAM can be described by thinking of a robot, which has been put in an unknown environment in order to accomplish two main tasks whilst moving: to map the surroundings and locate itself within the created map. Given this scenario a robot registers different salient features often in the form of points, which become map landmarks if they can be repeatedly found with the robot sensors. Both vehicle and landmark locations are considered together, as a common error is produced between them yielding correlation, Figure 1.1 (Smith et al., 1986).

The popularity of this approach comes from allowing a robot to give a better estimation of other landmarks, using readings which possess little error from an earlier registered landmark when the robot had little uncertainty. This is usually performed using the Extended Kalman Filter (EKF) (Durrant-Whyte and Bailey, 2006), allowing for estimation (robot position) and correction through repeated landmark observation.

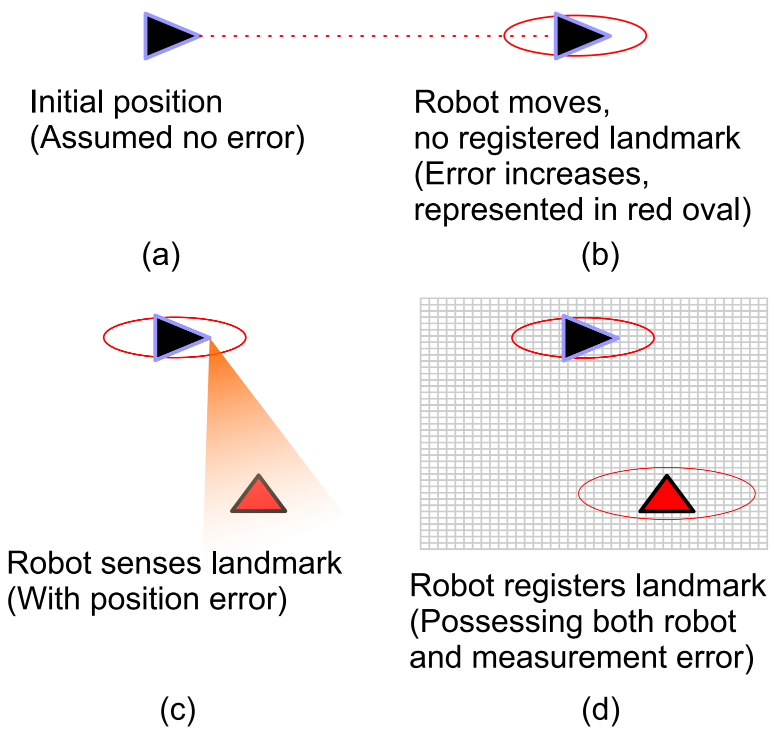


Figure 1.1: Error correlation between robot and landmark in EKF SLAM. The initial robot position is assumed with no uncertainty or error in its position (a). If the robot moves its uncertainty will increase as its own sensors might drift from the real measurement (b), thus accumulating error. This error is present when the robot observes a landmark (c), the latter is added into the map with a correlated error (d).

For a feature to become a valid landmark in SLAM it must be constantly predicted from different robot positions until it is out of range, a feature failing this test is discarded (Davison and Kita, 2001). Successfully added landmarks then allow robot localisation after long periods of absence, as each one of them helps to update the robot position as well as the landmark location relative to the previous robot pose. As the same measurement from a landmark is used to update other landmarks positions, the correlation between these grows even bigger (Durrant-Whyte and Bailey, 2006).

The nature of detecting, aggregating and re-observing landmarks is completely automated with no particular criteria in their selection. This ignores semantics of any sort that might give surrounding context, or feature state indicating present object condition, e.g. broken desk in office. This is in part because points are good salient features when there are computational constraints, as in the case of small vehicles where processing power is not readily available. However, points are limited in any descriptive property as they are very localised features, which might represent any object in the surroundings. This SLAM paradigm has been and continues being studied for more than a decade, as it possesses an elegant mathematical solution.

As such, there is a considerable amount of research performed in SLAM which compile ideas from many different areas. Some examples represent new ideas and challenges in SLAM, including optimisation of its computational resources, improvements using new estimation techniques (Li et al., 2014), implementations in other environments outside indoors (Lee et al., 2014), coupling of multiple sensors (Luo and Lai, 2014), localisation within ambient magnetic fields (Jung et al., 2015) to name a few (Durrant-Whyte and Bailey, 2006, Bailey and Durrant-Whyte, 2006).

Despite all the research performed in different areas of SLAM there is still a missing link between feature acquisition and meaning, state or surrounding context. This is something that cannot be easily programmed into a robot, as it would require an artificial intelligence capable of evaluating many factors in its surroundings, getting more close to human recognition.

Through years of experience or training a person can infer context or describe object's state often instantly. Ideally, user given semantics would allow to expand the usefulness of SLAM: A task can be programmed beforehand if an objective is known, e.g. locating a place or handling objects. However, if any task is required beyond moving from one point to another or outside of an object database, a person could input descriptive information or assign an object in the moment of observation. This could be potentially useful in search and rescue missions, where often there are unexpected scenarios. The given user information could be relayed to other robots or persons as well, which could assist the operation at hand.

Therefore, this work aims to relate user input into SLAM complementing an often fully automated paradigm. This does not only rely on offline object labelling approaches, but rather incorporates user interaction within the SLAM cycle in order to unify object selection with real-time semantics, exploiting user experience combined with pose estimation in order to increase map usability.

This information can potentially help collaborative scenarios involving other robots or persons, as it relays important points or objects of interest avoiding redundancy in surrounding recognition. This enables fast specialised task execution for other robots with limited SLAM capabilities, or other users in need to perform in-place objectives.

At the same time this means that an user will not be able to directly use points, as these are very localised and unless chosen by an algorithm not reliable for repeatable observation. This also aims to inspire further research, in which a robot can perform different tasks with no offline re-programming. This includes the possibility of aiding other users and the robot itself through interactive input whilst on algorithm execution.

This research is presented by introducing the concept, whilst looking into user guided tools which might allow for interactive input in self localisation and mapping. As such, the following sections briefly describe the general state of SLAM, with some investigations that execute other tasks besides localisation and mapping. This includes implementations which do not use localised features, introduce environmental semantics or enable some form of user interactivity in them.

1.2 User Input Hints and Local Feature Avoidance in SLAM

Human interaction into the SLAM methodology has been little considered in the research literature. Examples point to slight hints by showing an user setting a reference feature with known dimensions, thus providing valuable prior information for the robot (Davison et al., 2007). Applications with augmented reality (AR) include interactive virtual object placement with camera movements affecting its perspective (Skrypnyk and Lowe, 2004). An active approach is proposed in which an user is guided to move the camera in forward, backward, right, left, up, down and stay positions. This allows to obtain more information from landmarks which possess big uncertainty (Vidal-Calleja et al., 2006).

Outside of the above examples SLAM approaches tend to automatically acquire rigid, algorithm dependent and local features such as points which offer no context or meaning in them. Often they are extracted from the environment using different sensors, e.g. sonar, electromagnetic, laser range finders or cameras. The recent affordability and quality increase of image sensors and the vast information that an image provides gained vision-SLAM momentum, which also began using points as main salient features (Munguia and Grau, 2007).

Image feature acquisition often consists of algorithms that extract points from patches, e.g. the high contrast point detector, Scale Invariant Features Tracking (SIFT), Speeded Up Robust Features (SURF), or Features from Accelerated Segment Test (FAST) (Shi and Tomasi, 1994, Skrypnyk and Lowe, 2004, Bay et al., 2008, E. Rosten and Drummond, 2010). Therefore, dense feature acquisition and clustering is required in order to avoid sparse representations. This is because points are unable to offer any description due its localised and limited nature.

However, an image provides more information than other sensors, allowing for more flexibility in feature acquisition. Only a few examples have been presented as alternatives to localised features with dense representations, these employ curve fitting or object recognition techniques (Pedraza et al., 2007, 2009, Ali and Nordin, 2010, Rao et al., 2012). Curves are selected because otherwise points cannot represent certain environments, such as those containing smooth curved walls. Object recognition is used because a database can contain more data about the object itself, e.g. chairs with predefined dimensions, as they provide better feature association compared to points.

This thesis aims to present benefits in SLAM with interactive semantic or descriptive input, whilst also introducing a feature selection based on user preference by exploiting feedback from a constant video feed. With this better environment description can be attained, useful for both user and robot thus enabling further collaborative applications in SLAM. The challenges of this approach will be explored along the way in this document, as they are the basis for novelty and contributions of this research.

1.3 Aims and Objectives of User Involvement in SLAM

This investigation revolves around creating a new branch within SLAM research, one that involves user interaction in SLAM. However, this by itself aims to introduce three concepts: high-level feature creation, interactive object labelling and the combination of them to promote user interactivity in SLAM.

High-level feature creation avoids using localised features like points, this is mainly based on objects as they offer meaning and context within a map. At the same time, these features rely on user selection in the moment of their observation. This implies that there is no previously created object database, which allows to avoid offline reprogramming of the robot and at the same time allowing input semantics or description.

Objectives:

- Involve an user in the creation of high-level features (Munguia and Grau, 2007), without obtaining data too abstract that it cannot be used for tracking purposes, i.e. provides no measurable quantities or simply that it is not possible to use as a landmark.
- Enable either semantics or descriptive input in the feature, as this information is valuable within a map.

User interaction requires a method to input information, which will be used for high-level feature creation but also for semantics and description. This also includes any other form of input that can help in SLAM (Pedraza et al., 2007, Rao et al., 2012).

Objectives:

- Devise an interface which allows to see the created high-level feature in real-time.

- Allow to modify the high-level feature in real-time. This is in order to improve re-observation, allowing to correct user initialisation deviations.

Perform interactive SLAM using the created high-level features coupled with an estimator (Pupilli and Calway, 2005, Montiel et al., 2008), allowing to obtain position estimates for robot and high-level features by re-observation of the latter.

Objectives:

- Use information from the high-level feature entered by the user in order to perform localisation, which requires repeatable observation from different positions of the feature.

- Display position and descriptive information about the feature and robot in real-time. Performance will be evaluated according to tracking capabilities using a high-level feature, which will change in perspective with changes in robot position.

Finally, another major aim of this investigation is to make future propositions, which can only be accomplished with active user input. As an example, this could include setting a course of action instead of fully automatic robot exploration. This is because sometimes a person has already a better set of decisions leading to intuition, which tells about how to proceed in a particular scenario. This sort of thinking behaviour is not exactly easy to program into a robot, for this is what makes us human; be our decision a mistake or success. Therefore, the following section describes the contributions of this research related to this aims.

1.4 An Interactive SLAM Contribution

This research contributes a novel human-robot cooperative approach for vision-SLAM, in which a human is part of the algorithm discerning features of interest from an image sequence. This leads to offloading computational resources with better and meaningful feature selection compared to a baseline SLAM algorithm.

At the same time user semantic labelling and state input in the moment of SLAM runtime are introduced, such as “this is a door”, “this is an office” or “office door is broken" which can be of use for another user or robot. This does not require an external module but merely user intervention, this hinted in Goal-Directed Navigation but instead of human interaction an additional device is used to detect particular features (Davison and Murray, 2002).

Users are considered an important part of SLAM for this research, as object segmentation and differentiation has huge computational penalties in computers but becomes something natural in persons, as years of experience through human senses allows to quickly discern them (Su, 2012).

A constant flow of images is proposed to be used for interactive feature selection and tracking, where filtering is involved whilst accounting for speed. These features may not only involve object location, their usage will be expanded in order camera position as well. Therefore, the following contributions are detailed and summarised:

- The primary contribution is that, to the best of our knowledge, this work is the first implementation to incorporate interactivity in a SLAM algorithm through user feature selection. Other SLAM implementations have not included the ability to select objects from the surroundings for tracking, even if they are simple in nature as investigated in Chapter 2.

- A compilation of the mathematical foundations used in monocular SLAM, including an inverse depth parametrisation approach in Chapter 3. This is as many works assume these foundations are known, as seen in much of the research presented in Chapter 2.

- User semantics and description in algorithm runtime is shown in Chapter 4. Labelling of an object avoids the need of offline databases for object matching. The current state of a feature and semantics can also be described right after it has been selected, e.g. when selecting a computer screen it can be labelled as one, indicating that it probably belongs to an office. This improves map usefulness which might also be helpful in cooperative scenarios.

- A first implementation of real-time Gradient Vector Flow (GVF) forces driving active contours for object tracking in SLAM is seen in Chapter 4. As a first hypothesis in the presented interactive approach, active contours is a known user assisted tool capable of following silhouettes from objects. However, this ability comes with serious performance penalties. Thanks to

General Purpose computing on Graphics Processing Units (GPGPU), it is possible to attain real-time speed. Tracking information is then obtained at video rate speed, which is later on given to an EKF monocular SLAM implementation.

- A first particle filtering implementation involving user interaction, allowing to provide object tracking and capable of estimating camera position is presented in Chapter 5. In this scenario the user is allowed to describe an object through its edges, which then are represented using an novel line detector based on Hough and Bresenham algorithms. User input is seen through a vertex approach based on geometric constrains over the object, and also with real-time manual feature modification.

1.5 Thesis Overview

Chapter 2 describes previous and current insight into SLAM research. Section 2.1 introduces the basic SLAM idea, whilst Section 2.1.1 describes current SLAM research. However, the main interest of this investigation pertains to the vision-SLAM branch and thus seen in Section 2.2, which narrows the investigation focus towards vision-SLAM. Section 2.3 shows works that allow to further demonstrate and justify active user participation in SLAM. Finally Section 2.4 gives particular key remarks that motivate this investigation.

Chapter 3 describes the often used estimator EKF, whilst also detailing in depth a baseline monocular SLAM algorithm. Many concepts of importance in this document are seen in this sections, e.g. adding features or removing lens distortions. The EKF and its nomenclature is introduced in Section 3.1. Monocular SLAM is deeply discussed in Sections 3.2 and 3.3 using an inverse depth parametrisation exploiting the parallax effect for depth estimation.

Chapter 4 presents a first attempt to introduce active user participation within vision-SLAM. This approach is based on the EKF with user interactivity through active contours or snakes, which first require Gradient Vector Flow (GVF) forces seen in Section 4.1 and whose real-time performance is explored in Section 4.2. The foundations of the active contours can be seen in Section 4.3 and the results of applying them into the SLAM algorithm are presented in Section 4.4.

Given the results of the approach presented by using EKF, monocular SLAM and active contours, it was decided to employ a different estimation method such as particle filtering in Chapter 5. This remains a Bayesian approach but it diverges considerably from EKF, as it does not need linearisation. However, a new way to incorporate users is needed as this would allow better flexibility in long term. As a result, particle filtering is described together with some of its resampled and recursive implementations in Section 5.1. Likelihoods are required for the algorithm to work properly, which are related at the same time with user input. Therefore, a method which allows for both interactivity and likelihood is devised by means of the Hough transform in Section 5.2. Fusing this likelihood into particle filtering is seen in Section 5.3, with user interaction accounting for initial assumptions in depth in Section 5.3.3 and the results of this implementation presented in Section 5.4.

Finally, the conclusions to this investigation are presented in Chapter 6. This is proposed to be followed by the future work presented in Section 6.1.

Chapter 2

Background

Chapter 1 explored SLAM in regards to its basic principles, highlighting that dynamic interactive feature selection has not been introduced within this area of research. Meanwhile, robot platforms become more affordable as time passes, and they are further involved in industry or domestic tasks when aided by localisation and mapping. SLAM then allows robots to perform pose estimation whilst moving, yet this remains largely in reference to fixed and local features from the surroundings.

SLAM implies in the name that both localisation and mapping happen simultaneously, yet the robot first requires to map features which allow it to localise itself. This was reflected in Simultaneous Map Building and Localization (SMAL) for Autonomous Mobile Robot and Concurrent Mapping and Localisation (CML) (Leonard and Durrant-Whyte, 1991, Durrant-Whyte and Bailey, 2006). This is fundamental as a robot never really knows where initially it is when just placed in unknown environment, instead it maps features and takes them for granted as part of the environment. Therefore, this chapter begins by giving more details which allow a robot to use SLAM for self-localisation in Section 2.1, which briefly gives state of the art research focused on standard SLAM in Section 2.1.1.

Often landmarks take the form of points, which are automatically obtained using dedicated algorithms. The caveat of this is that dense representations are required, since using points are very localised limiting any descriptive information in them. Therefore, the focus shifts drastically in this literature review to the branch of vision-SLAM in Section 2.2. This is as images provide more information than other sensors and thus flexibility in feature selection. The principle of vision-SLAM remains similar as in standard SLAM, but also requires dense feature acquisition for meaningful environment representation.

Early approaches for vision-SLAM are described in Section 2.2.1, with state-of-the-art investigations presented in Section 2.2.2. This is then narrowed to monocular SLAM research in Section 2.2.3, as processing different video sources can be computationally costly. Then Section 2.2.4 discusses why user involvement within automated approaches like monocular SLAM is a good idea.

Later on some works going beyond localisation and mapping are explored in Section 2.2.5. These explore alternatives to points in vision-SLAM, which employ object recognition techniques or use other feature characteristics like silhouettes. Some of these approaches are closely related to this investigation, but they still remain dependent on previously entered data outside of operating cycle. As such, any feature that does not follow a stored pattern is not taken into account. However, these works provide glimpses for user interactivity within vision-SLAM.

The focus then shifts into tools that can be used for interactivity within SLAM, like active contours in Section 2.3.1 which is considered an assisting tool for object selection. Particle filtering is also seen as a way to introduce interaction in SLAM, thanks to flexible hypothesis generation in Section 2.3.2. Finally concluding remarks are seen in Section 2.4. Therefore, as a first step to introduce user input in SLAM its basic inner workings are explored next.

2.1 Describing an early SLAM methodology

The SLAM methodology allows a robot to ask itself where it is, which it then proceeds to estimate through landmark observations. However, what the robot obtains are positions with inherent error. This is because as the robot moves its uncertainty grows, even when the robot is equipped with sensors to measure its movement. This is as any sensor is assumed to have imperfections in its measurements, but also because of external factors, e.g. slippage in wheels. Subsequently, this uncertainty also affects feature observations.

Observing features is also assumed an imperfect process, inherent to any sensor used and thus contributes to uncertainty (Figure 2.1). However, the robot continuously registers and recalls its distance from different landmarks, allowing

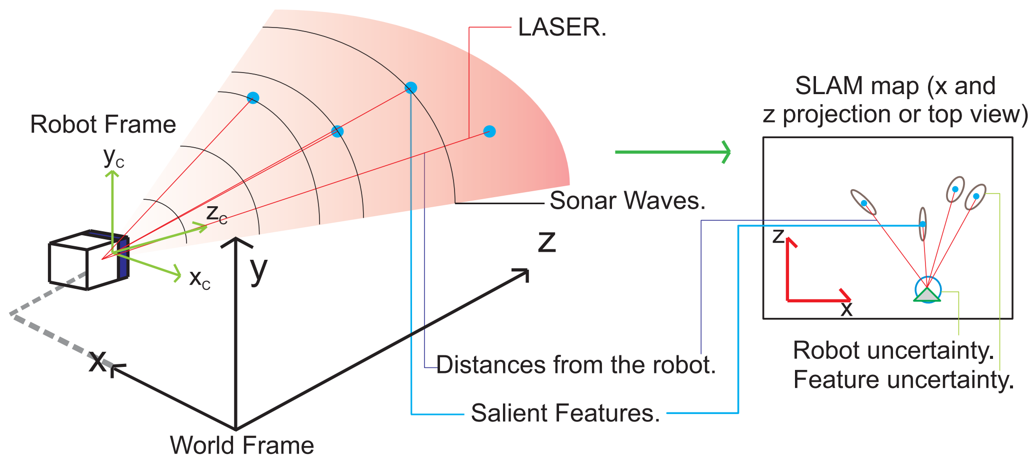


Figure 2.1: Basic SLAM Operation. A robot detects salient features from the enviroment, such as corners or high contrast points. Sonar and LASER sensors offer distance measurements towards the features, which are then registered on the SLAM map. Robot uncertainty is present because of movements without registered features, including initial assumptions at the start of the SLAM algorithm execution. This uncertainty can also increase due to model linearisations and noise, which then becomes correlated with measurement error. This is represented in the SLAM map in the form of uncertainty elipses around the robot (triangle with green contour) as well as the features (blue dots).

to perform position drift correction and uncertainty reduction. Landmarks are salient features from the surroundings that were successfully added, following a criteria in which they have to be unique enough to be associated from different observations in distinct positions. At the same time there are other sources of uncertainty, which include initial assumptions made about robot parameters or linearisation. This takes place in motion or feature appearance models, which are often involved in estimators like the EKF or its derivatives.

In initial conditions there is an assumed origin position with very little uncertainty (close to zero) and no added salient features. This robot position uncertainty remains small whilst no movement is performed, but it will immediately accumulate as soon as there is movement. The first landmarks seen by the robot are initialised with low uncertainty if they are closest, further ones will initialise depending on the current robot position uncertainty when observing them. If there are no landmarks on sight the robot can only rely on its own sensors, which consequently increases its current location uncertainty.

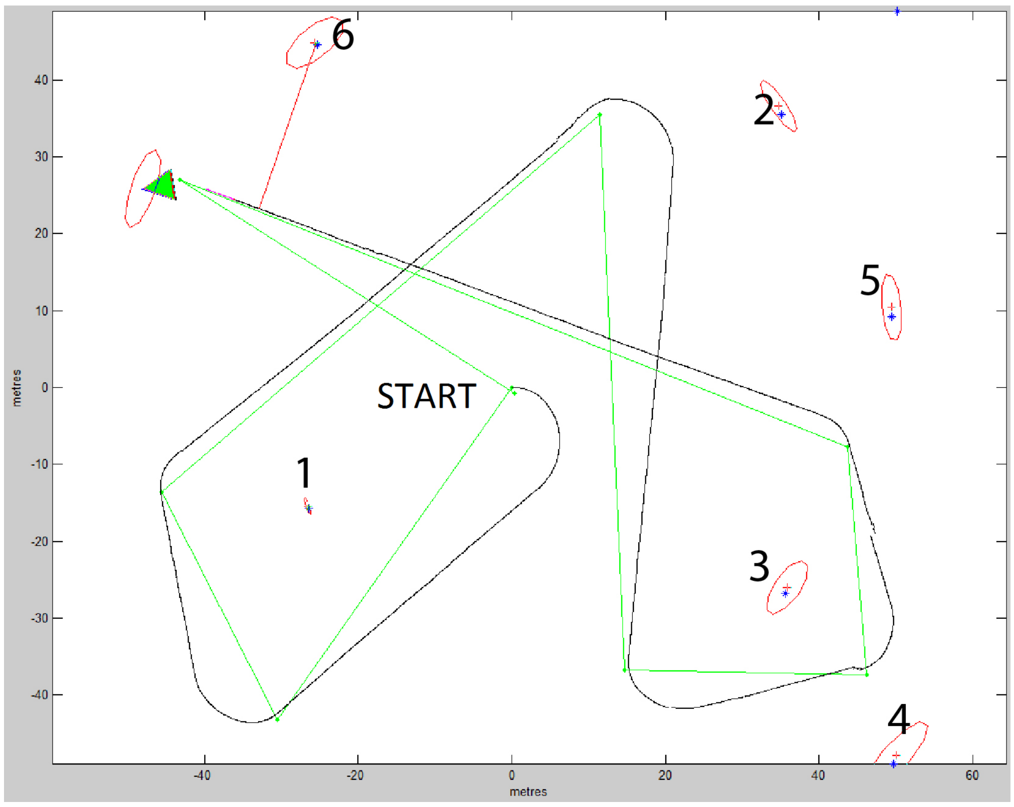
Figure 2.2 demonstrates a basic example of a SLAM algorithm[*] using EKF (Dissanayake et al., 2001): given an unknown environment, a robot equipped with sensors detects salient features from its surroundings. These are then registered as landmarks allowing the robot to reference itself through their repeated observations. The obtained measurements are assumed not precise, i.e. they are contaminated by noise. This assumption is based on the fact that sensors are not perfect, also including unforeseen events beyond said imperfections, e.g. slippage on the robot wheels. Newly added salient features are affected by robot uncertainty: if it is big, then the added landmark uncertainty will also be big. This also applies in the opposite case and hence its correlation.

One way to minimise both robot and landmark uncertainty is making use of an EKF, which uses predictions and repeated observations to give a position estimate. The EKF considers equations for both state and observation models: the former predicts the robot movement and the latter the landmarks appearance in the surroundings. The EKF stores all position parameters in states containing translations, rotations, landmark locations, etc. in a vector and their uncertainty information of each states is represented in a covariance matrix. As more observations are obtained, the uncertainties are reduced by means of a Kalman gain which later on updates both state vector and covariance matrix.

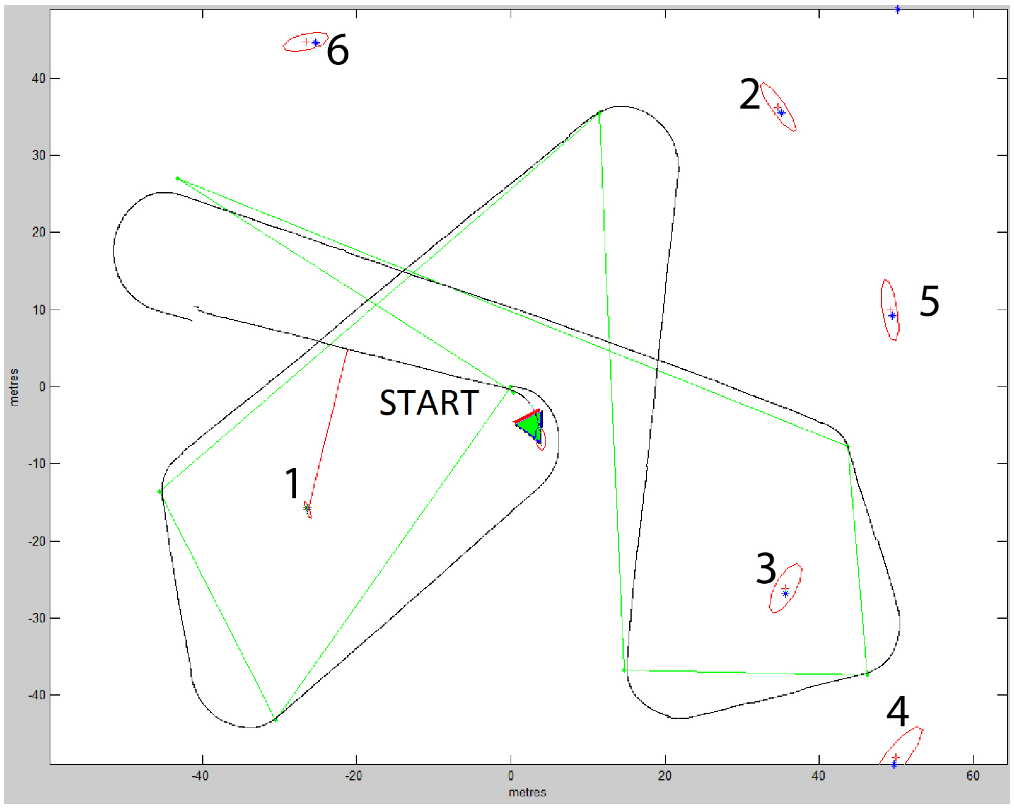
The Kalman gain changes depending on how good a prediction of the robot position is according to landmark observations: landmarks are foreseen from a predicted robot position and measurements are expected close to said predictions, any deviation in these observations will have an effect on the value of the Kalman gain. As a particular note the extended part in EKF comes from the linearisation of robot or observation (or both) models using a Jacobian matrix (Taylor series). However, this approximation and other assumptions induce uncertainty and thus overall noise in the system.

Further research improves or expands the SLAM concept beyond localisation and mapping. However, the basic idea remains the same. Therefore, some examples of recent SLAM research are described in the next section. These reflect recent SLAM development in all areas, allowing to focus later on Vision-SLAM research.

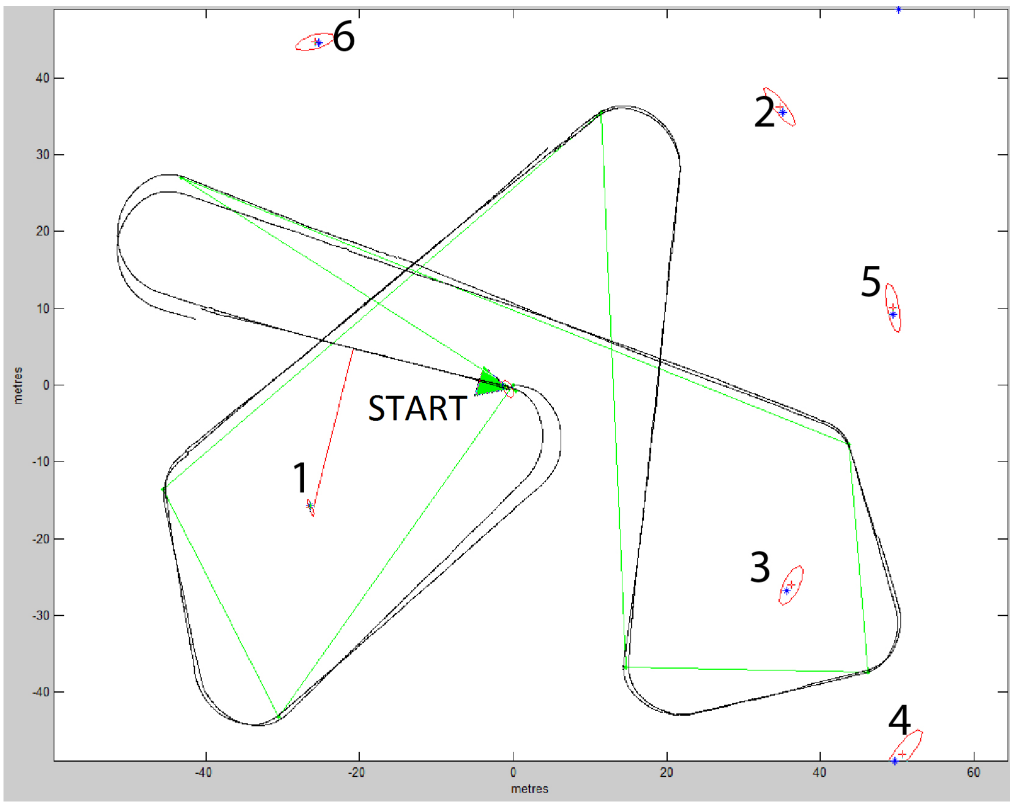
[*]Code can be obtained from *http://openslam.org/*, SLAM Package of Tim Bailey.


6
2
5
START
1
3
4
metres
metres
40
30
20
10
0
-10
-20
-30
-40
-40
-20
0
20
40
60

(a)


6
2
5
START
1
3
4
metres
metres
40
30
20
10
0
-10
-20
-30
-40
-40
-20
0
20
40
60

(b)



(c)

Figure 2.2: Simulation of a basic SLAM approach. A robot in an unknown environment (green triangle) following a predefined target path (green) discovering salient features (blue asterisks) is shown in (a). The traversed path (black) coupled with robot and landmark (x, y) position uncertainties (red ellipses) are estimated by the SLAM's internal EKF used for this simulation. Both landmarks and robot position uncertainties are correlated. This is because as the robot moves with no measurements its uncertainty increases, which then affects newly initialised landmarks. In this simulation salient features are observed in the order expressed by the number above them. The robot reduces its location uncertainty by re-observing the first landmark in (b), which was detected almost immediately and initialised with low uncertainty, therefore allowing the algorithm to correct the estimated path. The correction is seen as a discontinuity of the black path. As the location uncertainty of the robot is low, the landmarks position uncertainty is decreased as soon as the robot sees them again in (c), displaying smaller uncertainty ellipses compared to (a). The robot path can be repeated as many times as needed in order to lower all landmarks location uncertainties (EKF convergence).

2.1.1 Current SLAM Research

SLAM has been continuously researched from the basic approach described in Section 2.1. Many other applications start with the idea of obtaining a position estimate through landmark observations, improving the inner workings of this methodology. These investigations are briefly mentioned without going deep into general SLAM investigation, as the own nature of SLAM leads to considerable research branching and also because the primary area of interest in this thesis is interactivity, which is heavily involved with the area of Vision-SLAM.

The EKF is not the only estimation algorithm that can be used in SLAM. Other examples include the Unscented Kalman Filter (UKF) or particle filtering approaches. The former has similarities with the EKF but does not use Jacobian matrices for linearisation, whereas the latter uses a multi-hypothesis approach representing many different system states. Another example includes the Rao-Blackwellised Particle Filter (RWPF) which propagates a posterior using an EKF, then uses a multi-hypothesis approach to obtain estimates.

Many other works have grown from these ideas with recent examples including multi-sensor and collaborative approaches in order to improve performance, whilst also allowing to detect moving objects or work in cluttered spaces (Luo and Lai, 2014, Moratuwage et al., 2014). Also shown are arrangements of specialised sensors to detect magnetic fields, which provide refinements in pose estimation (Lee et al., 2015, Jung et al., 2015). Data association has also particular interest, since mismatching a landmark with false readings can lead to divergence (Li et al., 2014). Reliability in estimators is continuously tested, as erratic motion can lead to divergence (Carlone and Censi, 2014, Deusch et al., 2015). Applications on different environments other than indoors are considered, like underwater (Lee et al., 2014).

This small sample of research and many others are focused on improving previous SLAM investigation, as this can be aided by many other branches in engineering. However, SLAM started using points as main features for the algorithm to work and these are limited in their description and meaning. This produces research which tries to partly solve then inherent limitations of points, with many of them delivering improved results but no definitive methodology in SLAM accounting for all possible scenarios. Therefore, recent research is also being made beyond localised and dense point representations, bringing a new trend of SLAM by creating maps for metrics and topology (Reinoso et al., 2014). However, an image provides more information than other dedicated sensors allowing for more flexibility in features, albeit requiring filtering in order for an image to provide useful data to perform SLAM.

Cameras have been introduced in SLAM research, taking also the original approach of relying on feature points. However, the main difference lies in their acquisition: instead of obtaining distance readings using dedicated sensors (e.g. sonar, laser), observations are obtained in the form of two dimensional coordinates from an image. Vision-SLAM has gained popularity as sensors have become of better quality and affordability, but also because computational power has become enough to speedily process images. This prompted the use of more complex features in images for pose estimation, e.g. lines, planes or curves, whilst also offering new possibilities like augmented reality. The latter is a natural consequence of video as often graphics overlays can be used over images.

However, a key element in this thesis taken from cameras is their capability of providing visual feedback to an user, thus enabling a novel element within SLAM in the form of interactivity. Therefore, the next section focus primarily on Vision-SLAM by describing first earlier approaches, then discussing into single camera setups. This also includes user involvement as well as applications of Vision-SLAM beyond localisation and mapping.

2.2 Vision-SLAM

Compared to other traditional SLAM approaches Vision-SLAM characterises itself by using cameras, which extract salient features from the environment with or without using auxiliary sensors as in Figure 2.1. Often single camera setups are preferred, as they offer a simple, cheap and compact implementation which can be put virtually in any environment.

However, obtaining information through cameras with no aid of other sensors presents a different challenge compared to traditional SLAM. This is because whereas vision-SLAM can rely on the same estimation techniques like the EKF from its non-vision SLAM counterpart, its observation models need to account for landmark appearance in images where no depth information is present (Figure 2.3).

Despite these shortcomings, cameras offer possibilities that otherwise cannot be achieved with dedicated sensors, as the information from the latter is limited. For example, vision allows to detect complex structures or common objects within their surroundings (Gee et al., 2008, Hwang and Song, 2011). Vision also relates to real-time decision making in persons, by directly feeding information to an user which then through experience can interactively respond to the system.

As such, this thesis drastically changes focus towards vision-SLAM from here on, as it might offer insight in the novel idea of interactivity within the SLAM context. First, Section 2.2.1 presents early approaches which mostly introduce cameras as sensors in SLAM. Section 2.2.2 follows with a compilation of some general vision-SLAM work. However, there is no precise path to follow which marks an evolution over previous research. This is because as with standard SLAM further investigation is concentrated on improving over the foundations, mostly stability concerns and pose estimation improvements.

Later on, Section 2.2.3 focuses on monocular SLAM as their implementation requires almost no special setups nor arrangements, also allowing to lower computational resources for real-time operation. Section 2.2.4 differs from the previous literature, by setting a perspective in which persons are involved in automated processes in order to improve them. Finally, Section 2.2.5 shows vision-SLAM implementations which go beyond just localisation and mapping. This describes potential for future research which is not only focused on performance tweaks for SLAM, but rather implies the use of abstractions as features in this methodology.

2.2.1 Beginnings of Vision SLAM Approaches

Earlier works in vision-SLAM adapt the SLAM methodology described in Section 2.1 by concurrently extracting salient features from images of the environment, provided by one or multiple cameras (Figure 2.3). Early approaches started using a dual camera setup mounted over a robot, that whilst in movement acquires points of high contrast as features the Shi and Tomasi operator[*] (Shi and Tomasi, 1994).

[*]This operator was used as it offered good performance at the time, other operators are mentioned in Section 1.2

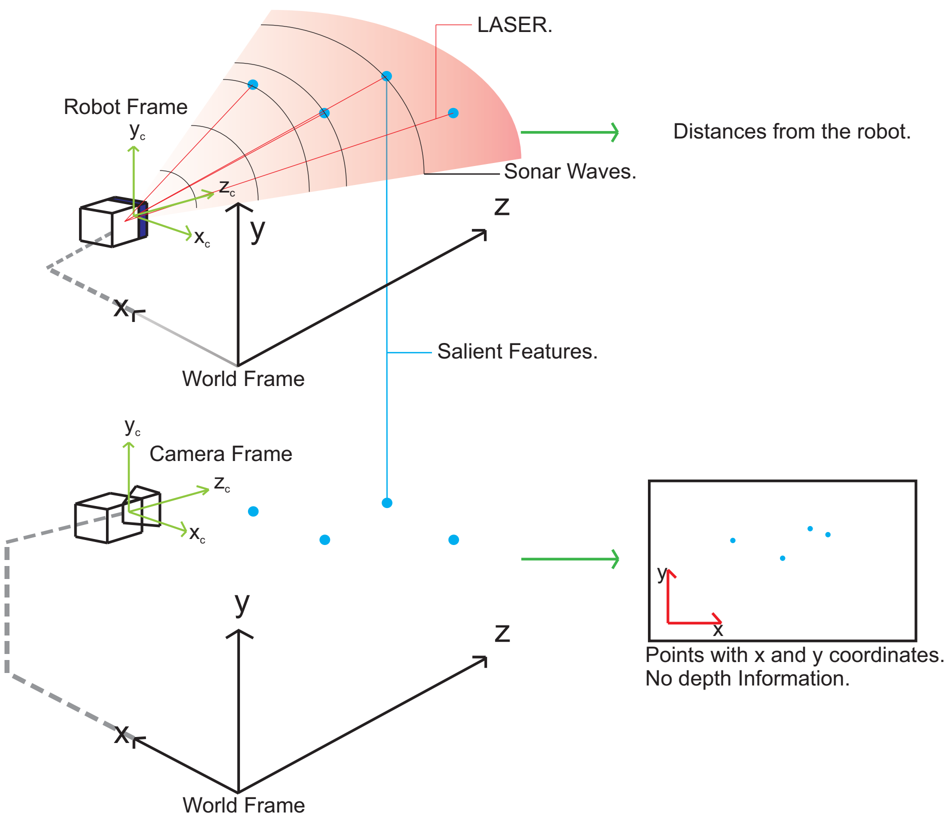


Figure 2.3: Camera and other sensors in SLAM. The upper diagram illustrates a robot performing SLAM with common sensors like sonars or laser range finders, which are used to detect salient features from the surroundings. Distance and bearing information is obtained in this way. The lower diagram shows a robot performing SLAM using only a camera. Because an image only contains two dimensional data, only coordinates can be used leaving depth to be obtained indirectly.

The right image performs stereo matching using epipolar geometry. If the feature is continuously matched then its 3 dimensional position is inferred, otherwise it is rejected (Davison and Murray, 1998). This is later on added as a landmark allowing to use EKF estimation to track it and reference the robot.

As this initial research was limited to a planar environment, no tilt changes were taken into account. Therefore, uncertainty increased when a slope was found by the robot. One solution found to this relies on an accelerometer to measure the pull of gravity, allowing the robot to infer its inclination (Davison and Kita, 2001).

This investigation also introduced 3D orientation information by using a quaternion system, which avoids the singularities presented by its Euclidean counterpart (gimbal lock). This is seen in many further SLAM implementations.

These first approaches rely on points for feature extraction, with no interactivity considered and thus not relying on user input. However, this can be explained by the need to have an algorithm with low computational cost. This is because both image sources are processed to extract image patches, which later on are matched through correlation. These operations are intensive on computational resources, with the research taking priority on using camera information to perform SLAM.

2.2.2 Further Vision SLAM Research

This section describes the research performed in vision-SLAM after the foundations seen in Section 2.2.1. The early stages of vision-SLAM research concentrated on cumulative improvements over the original investigations. However, as vision-SLAM can make use of many other areas of research further works do not follow an unique logical path as seen in the following paragraphs.

First, alternatives to EKF have been presented with estimators involving particle filtering to improve localisation. This includes an approach based on FastSLAM using a Rao-Blackwellised Particle Filter (RWPF) (Montemerlo and Thrun, 2003, Sim et al., 2006), and coupling with the Unscented Kalman Filter (UKF) with faster versions using the square root UKF (Pupilli and Calway, 2006, Holmes et al., 2008, 2009). The works which follow a similar approach to FastSLAM achieve a large number of salient features, therefore allowing dense representations.

These approaches use a considerable amount of computational resources as they use numerous pose hypotheses. The investigations using particle filtering in conjunction with UKF allow faster recovery from erratic motion, but map generation is not dense compared to FastSLAM approaches and its performance still behind of other EKF implementations.

Later on, the field of view in cameras has been considered for improvements in pose estimation. Different arrangements have been tested, including hyperboloidal mirrors (Kim et al., 2006), panoramic (Milford et al., 2006, Milford and Wyeth, 2008) and omnidirectional (Tardif et al., 2008, Andreasson et al., 2008). As local features remain limited in their context, association is a problem when obtaining readings after periods of absence. Therefore, these investigations show a increased amount of time in which localised features remain within the field of view.

In omnidirectional cameras and setups involving hyperboloidal mirrors landmarks remain visible even in full rotations, allowing robustness in association even when tested outdoors. The panoramic approach does not always retain a landmark in camera image, but this is aided by odometric information using a biologically inspired goal navigation approach. These investigations partly increase the complexity of implementation as the main interest is to keep localised features in view, as to solve problems in feature re-association when they are out of view. This reduces the need to obtain new features each time the robot turns, with reduction in the noise introduced in the system when aggregating them as landmarks.

Localised features represent a problem particularly in images, as these can only provide two dimensional features with no depth information. This increases the complexity of landmark estimation, leading to development of stereo camera algorithms with the intention of improving landmark appearance estimation. Therefore, epipolar geometry can be exploited in depth estimation (Schleicher et al., 2006), scale considerations are made in (Paz et al., 2008) and individual monocular SLAM executions are seen in (Sola et al., 2008). These approaches already benefit from a wide field of view, thanks to their stereo setup.

Epipolar geometry allows to infer depth based on the position of two cameras observing the same feature, but at the same time this requires detection of the same feature in both image sources. Considering scale is shown to reduce pose drift, with close and far feature representations used to represent structures. Decentralised schemes also allow infinity point representations, demonstrating also asynchronous usage and communication within SLAM algorithms. A side effect of using two cameras is that two image sources need to be processed, which depending on the resolution would scale computational load accordingly requiring efficient use of resources.

Further investigations integrate other sensors with cameras in vision SLAM as points are localised and limited in nature, with the objective of improving feature association indirectly yielding better pose estimation. These include laser range finders (Newman et al., 2006, Andreasson et al., 2007, Ramos et al., 2007) or three dimensional cameras which capture images with included depth information (Ohno et al., 2006). Different tasks can be assigned to each sensor whilst performing SLAM: laser range finders can build the map with cameras accounting for loop closing, with the advantage of relying on visual similarity rather than geometry. On the other hand cameras can perform SLAM whilst laser range finders are used for landmark association by creating three dimensional models, allowing to track shape and texture for improved motion estimation. Inertial measurement units in vision-SLAM are also used (Piniés et al., 2007), but their benefits reflect on reduced variations in map scaling thanks to improved forward and backward camera motion estimation.

Algorithm optimisation can be seen in these investigations, showing better feature association thanks to multi-sensor approaches. These allow features to become more resilient to false readings, but association and model complexity increase as more sensors are used. With this it can be seen that association is a notable challenge in vision-SLAM, especially when using localised features as their similarity between each other is high. Using different sensors allows to partly overcome this problem, at the cost of algorithm and implementation complexity.

Other investigations have tried to overcome the problems related to feature association, by means of using artificial landmarks. This includes RFID patches in the surroundings or simple printed fiducial makers (Kleiner et al., 2007, Hyon and Young Sam, 2009). Although the work using RFID patches uses a camera only to obtain images, its equipped antenna allows it to see the landmarks even when obstructions are present, with each one of them being associated by the information embedded on them. Fiducial markers share some of the previous advantages as well, as association becomes easier with each marker being printed distinctive from each other; they can also be put on paper and placed anywhere or even made invisible.

Many advantages can be seen for feature association with fiducial markers, as they offer meaning and context beforehand. However, this is an offline approach with user intervention limited to the information given to the fiducial markers. Improvements can be made in the form of user feature selection, as it would allow to register them in the moment of their observation rather than offline.

Due to limited computational resources and image processing complexity, these SLAM investigations tend to use points. These features perform well when the main objective of a robot is to estimate its localisation, with ongoing investigation improving their detection and matching. However, at the same time mapping is constrained by the use of localised features, as dense representations are required to offer an understandable representation of the surroundings. This represents another performance impact within their algorithms, as dense representations need to account for all the features with their respective estimation techniques. Overall this leaves little room for any user involvement and thus preferring fully automated execution. Other investigations have also found that processing multiple image sources is detrimental to performance, leading to efforts in which only one image source is processed, i.e. monocular SLAM.

The next section focuses on single camera setups only, as they are preferred in order to avoid processing twice as much image data. This allows to make propositions in which an user can be part of the algorithm, allowing to leverage some steps of feature detection to an user.

2.2.3 Monocular SLAM Research

Monocular SLAM is the idea of using only a single camera to perceive the surroundings around a robot. This approach offers simplicity of implementation, as it basically consists of a camera mounted over a robot. Stereo setups compared to monocular implementations can obtain depth through the use of epipolar geometry, but processing a single image source halves the effort and time spent in lens calibration, feature detection and matching. However, single cameras cannot directly provide feature depth from a single position (recall Figure 2.3). As such, indirect methods are required in which the camera needs different feature angles in order to provide a depth estimate.

The Rao-Blackwellised Particle Filter (RWPF) estimator is used with real-time performance in (Eade and Drummond, 2006), following the steps of FastSLAM and its stereo camera implementation (Montemerlo and Thrun, 2003, Sim et al., 2006). These approaches allow to register a considerable number of landmarks, with the ability to recover from shake in the implementations using cameras. The idea is that if the poses of a robot are already known, then the estimates from landmarks are independent from other features whose pose is not known. This is because an indirect uncertainty correlation can only occur between robot and landmark if both of their positions are uncertain. When there is no robot position uncertainty the correlation disappears. However, a map which can tell all the possible locations of a robot is required as prior.

Another investigation for monocular SLAM limited the quantity of tracked landmarks, coupled with a simplified constant velocity motion model (Davison, 2003). This is followed by another work with emphasis on fast path estimation using SIFT features (Skrypnyk and Lowe, 2004, Stasse et al., 2006). In these approaches the camera does not expect to follow a predefined path with landmarks along the way, instead grabs any available features from its surroundings. The camera is able to obtain real-time pose estimation by using sparse landmarks disregarding map representation, with the assumption that only angular and linear accelerations impulses affect camera movement. This implementation can make use of a prior, e.g. a calibrated frame with known dimensions. However, this is used to provide the internal distance units with meaning.

Faster performance is also achieved considering a total of 10 to 12 sparse features in (Davison et al., 2007). Compared to the previous work it discards SIFT image patches and instead uses high contrast points (Shi and Tomasi, 1994). This investigation also presents augmented reality thanks to its real-time performance. However, whilst the idea of SLAM in interactive scenarios is suggested, only computer graphic overlays are shown.

These investigations proved that single camera SLAM was possible even in real-time. However, as an early effort, further development of its suggested applications was limited, e.g. virtual reality. This is mainly due to computational constraints, as even landmark acquisition is constrained in order to maintain speed. Further investigation continues with algorithm tuning, speed being a priority with the main purpose of expanding its applications.

Feature depth initialisation takes interest as images can only deliver two dimensional data in (Gemeiner et al., 2007). A delayed initialisation for feature depth is presented, which uses a semi-infinite line representing all possible depth hypothesis. The line is directed towards the feature from the camera which later on is treated as a one dimensional Gaussian distribution, requiring some steps to create a depth hypothesis. Different views from the feature are gathered after moving the camera, allowing to obtain a suitable depth estimate. However, valuable orientation information is discarded, as the camera still needs to move until an acceptable estimate is found.

Delayed initialisation was addressed in the inverse depth parametrisation research (Montiel et al., 2008), followed by an improved version with robustness to changes in light, orientation and scale variations in (Chwan-Hsen et al., 2007). The parallax effect provided by feature points was used: the closer to the centre of the image from different positions the farther they are in depth. This has the advantage of linear modelling, which is ideally suited for immediate feature initialisation and EKF convergence. These investigations also suggest the use of wide angle cameras for monocular SLAM, as they possess a wider field of view which improves feature extraction. However, lens imperfections in these kind of cameras produce severe image deformations requiring radial and tangential models to correct them.

This research tries to address the inherent problem of using local features like points, as they can only provide much information from an image. Inferring data from these features becomes more complex and improved, but does not solve the need for dense representations in order to provide any meaningful representation.

So far no user involvement in the SLAM algorithm has been seen in any of the previous approaches. Mainly this is because monocular SLAM continues to follow further algorithm optimisation and applications. However, it is often believed that user interaction can be detrimental, even if it offers advantages in semantics or description. As such, the next section tries to justify user involvement within SLAM and perhaps other automated approaches.

2.2.4 Why User Involvement?

In SLAM, human interaction has been largely ignored as it is often thought that this methodology can remain completely automated, leading to minimal or non-existent user input in SLAM. This is of no surprise as reducing human input has been a trend in numerous engineering works.

Nevertheless, complete autonomy in machines might present potential problems in some scenarios: unforeseen circumstances might bring the robot to a halt or cause malfunction. Depending on the circumstances these problems can make a particular task delayed or unattainable. In the worst case scenario an unpredicted occurrence could present harm to the robot, to a person or loss of any object involved in the task. This is reflected in most of the robots possessing a total shut-down switch, with a form of user link with the robot in order to alleviate potential problems and costs for these fully autonomous devices.

Communication between robot and persons can be presented in many ways, with teleoperation being the most common. In this case the robot is just a tool and is often used for environments that are harsh or not known, its operation and task execution is limited to user skill and feedback speed between its actions and the robot. The opposite being autonomous systems, which grant independence to the robot. High-level goals are achieved this way and the person only functions as a supervisor with almost no interaction to the robot (Scholtz, 2002).

Depending on the autonomy degree in a human-robot system, the strengths of both can be maximized for a specific task taking into account their skills: ingenuity, dexterity, heuristic knowledge, intuition and 'common sense' for the human; continuous computational power for the robot (Milgram et al., 1993, Green et al., 2007).

However, systems enabling communication about goals, abilities, plans, achievements and problem solving between user and robot are more robust and better performing than those lacking the link with a person (Fong et al., 2006). Thus interactivity between robot and user is suggested for improved task execution, as the more information a person receives the more its experience can help in such tasks.

In recent years image sensors have become mainstream. Hence, computer vision allows robots to see the world. If autonomous systems would have the context and interpretations that human beings take for granted, then they might infer context and behave similar to a person (Su, 2012). However, these systems would need to analyse and understand visual data in the same way as a human. This is often the result of years of trial and error, inferring meaning through a constant sensory flow of information which includes natural vision.

Robots have increased their capabilities in perception and control, as there is an constant push towards process automation. However, high-level decisions, strategic planning and cognitive functions remain an 'achievable goal' (Milgram et al., 1993, Wang et al., 2015). Specifically for object recognition, humans segment objects better than machines (Datta et al., 2008, Borji et al., 2015). This advantage is due to years of association experience from a constant video feed through the person's eyes, being capable of discerning objects from multiple viewpoints, with different lighting conditions and even occlusions.

Particularly in SLAM, salient features from the environment that can present all the previous characteristics are preferred, yet current approaches only allow them to be very local with low-level and less robust association. This is often in the form of points which are unable to offer context or semantics, leading to limited map representation. Even with a dense number of points detailing the surroundings, further classification needs to be made adding another step. Involving a person in the algorithm would avoid this, allowing to create maps full of user given context.

Nonetheless, as technology progresses new ways to employ both robot and human skills arise. This causes a performance increase with human-robot systems surpassing perhaps that of fully autonomous robots (Milgram et al., 1993, Tsarouchi et al., 2016). An example of this would be augmented reality, enabling computerised graphics to be displayed on top of a real world image. This is an ideal platform for interactivity between robots and persons because it provides spatial cues for local and remote collaboration.

Robot interaction might also be used in collaborative workspaces, allowing to perceive positions of robots and other interacting persons (Green et al., 2007, Tsarouchi et al., 2016). Visual cues allow to reach a common ground for a human and robot, maintaining situational awareness. The robot can then communicate its internal state and intentions through graphic overlays on the real world view of the user (Collett and MacDonald, 2006). Therefore, interactivity does possess an interest within the research community, but at the same time it represents challenges which go from an user interface to a feedback mechanism for the user.

The next section will explore some SLAM works which use human interaction or extract high level information. Although scarce, they offer a glimpse for further research in this area.

2.2.5 Beyond SLAM

To the extent of our knowledge the idea of interactivity in SLAM is novel, in the sense that no other work so far has implemented user driven tools in SLAM context, nor attempted interactive user input at SLAM runtime. However, there are particular similarities with other works that help the motivation behind this investigation, together with approaches that go beyond just localisation and mapping.

Examples include sensor combinations used to find alternatives to points, i.e. avoiding localised features. This makes assumptions of environments which account for multiple geometric man made structures, allowing to search for high-level shapes where dense representations are not descriptive enough (Jeong and Lee, 2006). Cameras and infra-red sensors have been used to detect lines, making the algorithm more robust than relying only on points. A combination of lines and planes is also seen in order to achieve better camera pose (Lemaire and Lacroix, 2007, Viejo and Cazorla, 2007). A newer approach of local feature avoidance has also been researched, which relies exclusively on image gradients to perform SLAM (Engel et al., 2014).

Up to here it can be seen how abstractions can be used to obtain better estimates, as the features themselves are more 'unique'. For example, it is easier to confuse very localised features like points than to do so with lines. Usually these abstractions are complicated to distinguish in an automated form (Borji et al., 2015), whereas a person can discern them without much effort.

Further refinement led to find abstractions in the surroundings such as walls (Gee et al., 2008). Dynamic environments with moving objects are considered by repairing maps that have changed. This works by using long term matching techniques involving detection of intersections, walls, floors and ceilings (Konolige and Bowman, 2009). Depending on the detection of certain complex structures a robot can be assigned to perform labelling on walls, floors and ceilings. This assigns limited semantics on automated map generation (Flint et al., 2010). SLAM can be performed by looking for common indoor objects including: lamps, corners and doors. These are detected by a camera looking upwards (Hwang and Song, 2011).

These works show a tendency to detect commonly found indoors or outdoors objects, as they remain the same in essence with slight variations of form and shape. An algorithm can be given this information prior to perform SLAM in order to recognise them, or if user interaction was considered an individual might do so.

More complex research introduces B-Splines to SLAM, producing an improved representation of the map, by making better use of the measurements obtained with a laser range finder as often 95% of the readings are discarded (Pedraza et al., 2007, 2009). A similar approach is seen with the use of Bézier curves. These are used for stereo matching instead of points, but are limited to obtain curves from the ground plane image, i.e. from riverine areas or where surfaces lack textures as points detectors rely on them. Edge detection is used to produce the most visible silhouettes and later on curve fitting them in order to perform SLAM (Rao et al., 2012).

In these approaches no user interaction is given, as there are already assumptions about how the world might be and which objects there are in it. Nevertheless, there is the reasoning that dense representations are not always suited nor are the best performers.

Database object recognition has been used in order to reduce dense feature acquisition. Early approaches demonstrate a better landmark association. However, this method requires to build an offline database containing any expected objects of interest for association (Ahn et al., 2006). Sonar sensors coupled with cameras have been used: the former provides point and line features and the latter recognises planar objects from a previously constructed database (Choi et al., 2006).

A practical implementation shows a robot receiving input from a person to locate a particular product within a store, recognising it from previously stored images in a database (Gross et al., 2008). Other applications involve gaining descriptive information about an object of interest using database matching. An aiding robot can make use of these clues, e.g. a glass can be detected and the robot informed of it. This delivers more information inherent to the object, e.g. an object's material can be informed to the robot so it knows how to grasp it and to which site it must transported (Civera et al., 2011).

Pre-programmed labelling of indoor places through image matching also represents a huge advantage, as it allows a robot to execute specific tasks according to its location in a building. It also permits user input to perform predefined tasks in such places, with the robot knowing where it has to go (Ranganathan and Lim, 2011). Visually impaired persons can benefit from semantics provided by clouds of SIFT features. A set of features can be extracted and saved onto a database prior to SLAM operation. A camera can be set onto a person and the algorithm will extract features from the environment. For example if the gathered cloud of image patches match the entry of a dining table in the database, then feedback is heard helping the user to recognize its surroundings (Ali and Nordin, 2010).

From these examples it is possible to see that descriptive or semantic information is of great help for both the robot and persons. However, offline database creation must be done prior to SLAM execution. Even so these applications are limited to recognise only the objects or places that are stored in the database, restraining its flexibility. User interaction would allow for online user input, with information given to an object for further usage.

Augmented reality is also directly related with interactivity as the surroundings are expanded in information, using image overlays which may help a user in decision making. This concept has been applied in SLAM, creating games that offer different camera perspectives referenced to discovered planes (Chekhlov et al., 2007). However, this approach does not make an user part of the SLAM algorithm, instead it relegates the person to other applications which do not involve SLAM improvements.

One of the closest approaches to the research presented in this thesis refers to a system involving graphic overlays. Research shows an interactive annotation algorithm with three types of selection: punctual, planar or oval. These selections are incorporated into a FastSLAM algorithm through different models, allowing camera pose estimation (Montemerlo and Thrun, 2003, Reitmayr et al., 2007). Remarks are made for remote collaboration in unknown environments with an expert adding notes to support other collaborators. This approach signals towards the importance of state in features, yet discards any other possible input from the user in order to form part of the SLAM algorithm itself, instead relying on FastSLAM. Augmented reality takes estimated camera positions for perspective changes, changing the annotations perspectives as well.

This thesis presents an interactive approach, in the sense that there is missing active user interaction within the SLAM methodology. This is motivated by real-time user input in SLAM leading to flexibility in object selection, brief feature state description and at the same time because this introduces a new branch in SLAM exploiting interactivity benefits. This is particularly important for mapping objectives, as autonomous algorithms do not take into account objects of interest for a person. This makes the assumption that an user or other robots might not require fine grain detail on a map, but rather they prefer to focus on important objects or places. This information could also be shared in cooperative environments, which would speed up certain tasks in other robots.

The next section proposes the use of either active contours or particle filtering for feature abstraction. In both cases, the user can set a prior that eventually allows a particular object of interest to be tracked, which is used at the same time for mapping with user given semantics.

2.3 Enabling User Interactivity with Cameras

Most of the research in Section 2.2 has shown a complete preference towards fully automated approaches, but also gave a glimpse of recent interest in including more abstract user input. Hence, this research builds on these glimpses by introducing a novel approach, combining active user input with tracking into the vision-SLAM paradigm.

A first inspiration for a methodology combining human interaction with tracking was found in a research using fractal dimensions, which measure the variation between pixels over an image. The work uses active contours (snakes) which deform to track a texture with either user help or predefined parameters (Smith, 2010). The user can pinpoint a location over the image and plant a seed, creating a snake that eventually contains the selected texture. Although limited to textures the approach illustrates an interactive way between snakes and a person, allowing to obtain useful tracking information.

Active contours allow for interactivity as the user can set an initial position for them over an object of interest, evolving and keeping track of it whilst the camera is in motion. This causes position changes in the snake, which can be later interpreted as measurements affected by noise. All this information can be entered in an EKF vision-SLAM algorithm.

Particle filtering also offers a way to introduce interactivity into SLAM, due to its flexibility in hypothesis generation and voting. A defined criteria can be used to obtain votes based on interactivity, allowing to replicate the most supported hypothesis.

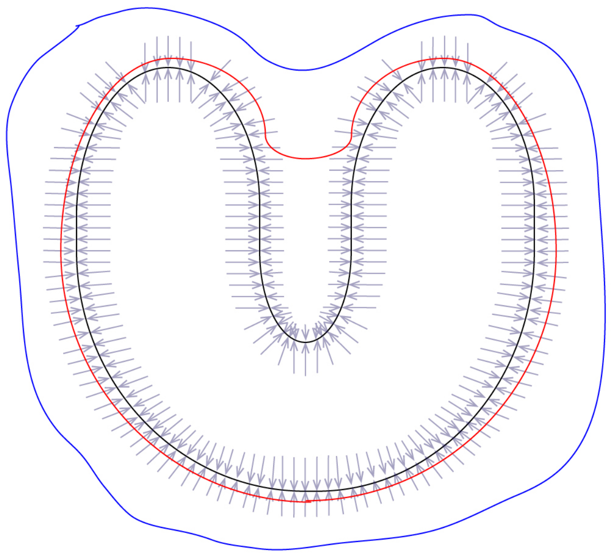
Therefore, Section 2.3.1 explores research pertaining to active contours and objects. However, works in which snakes can perform object tracking are fewer than those using active contours for image segmentation purposes. Later on Section 2.3.2 explores particle filtering in SLAM giving small hints to interactivity.

2.3.1 Active Contours (Snakes)

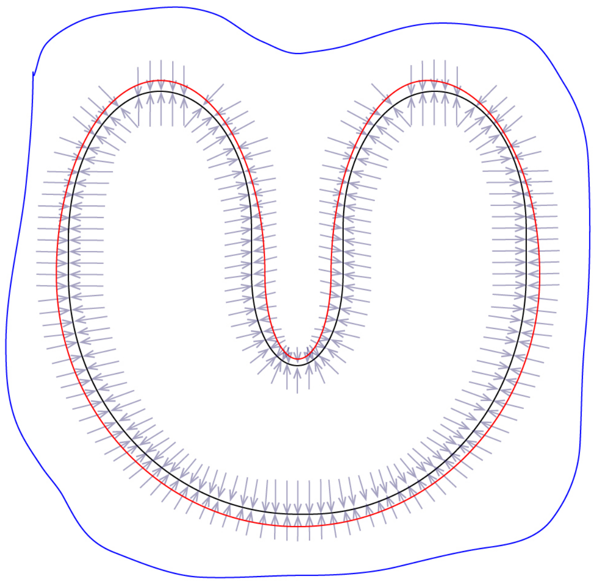
Active contours make use of interpolated curves, commonly found in other computer vision problems. The basic behaviour of a snake is as follows (Kass and Witkin, 1987): an active contour is initialised near a region of interest in a filtered (binary) image, which contains only discernible silhouettes. Attractive forces are generated from this image allowing the snake to attach itself to them, gradually taking its shape. A sketch of this can be seen in Figure 2.4, which compares an early (baseline) approach against a more efficient implementation of force extraction such as Gradient Vector Flow (GVF).

Snakes have the property of attaching themselves to a desired contour, which produces a local minima as shown in Figure 2.5. However, there is a need for a mechanism that initialises them near an object's contour. In the case of a person this is considered a power assist, as in the hands of an expert it allows to initialise the snake onto an object of interest (Kass and Witkin, 1987). Automated approaches can be used to initialise the snake, but they need to analyse the whole image looking for a desired local minima or object silhouette.

Tracking motion has been demonstrated using snakes since its inception (Kass and Witkin, 1987). However, occlusion or rapid changes in speed will cause the active contour to lose track of the target. Because of this the elements of the snake are proposed to include mass in order to minimise this effect. Later on, Kalman filtering was introduced in active contours (Terzopoulos and Szeliski, 1992). This



(a)



(b)

Figure 2.4: Sketches of different image generated forces. Shown in (a) and (b) are forces or vector fields (grey arrows) generated by a picture contour (black line) attracting an hypothetical active contour (blue line). This snake is initialised outside the figure and does not have to follow a particular shape. The baseline generated forces in (a) do not allow the active contour to fully follow the concavity after reaching an steady state (red line), as the forces generated around the concavity are parallel to each other causing the snake to be pulled in opposite sides. Forces generated using GVF appear in (b) which allow the active contour to fully attach itself to the silhouette of the figure, as the forces are not parallel and thus pull the snake inside the concavity.

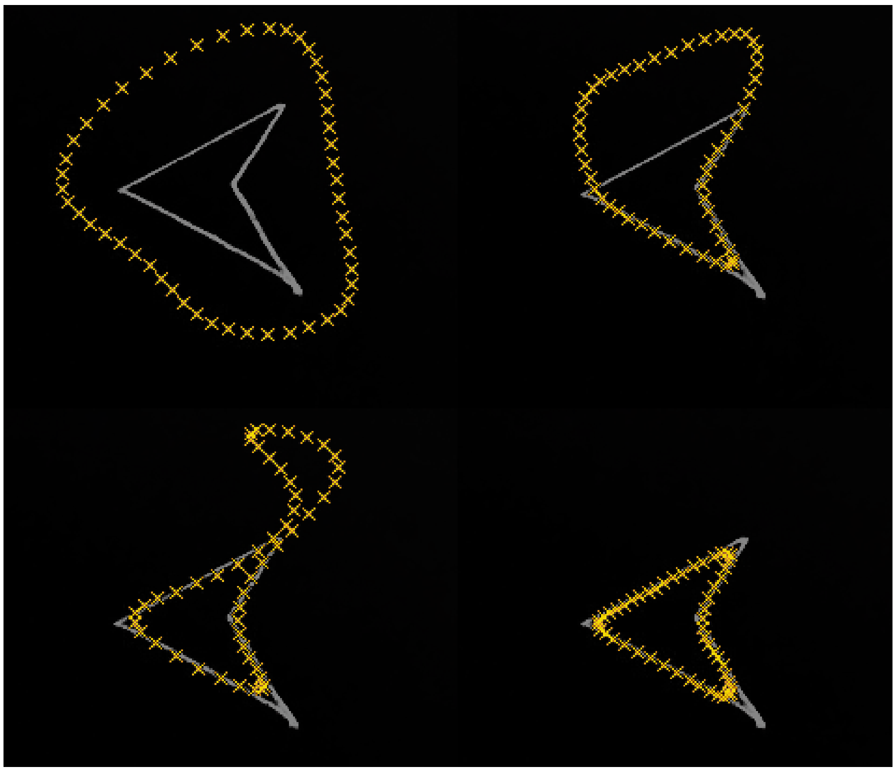


Figure 2.5: Active contour demonstration. The orange crosses are discretised points of the snake curve which are sometimes called snaxels. Image forces attract these snaxels with each one of them impacting the form of the snake in each iteration, which does not follow an uniform attraction towards the shape. After steady state has been reached, i.e. after all the snaxels have found local minima, the snake resembles the form of the silhouette in the figure.

proposed two models for snake tracking: the first one has fast performance but does not account for occlusions, with the second one following the idea of mass in the elements of the active contour, by means of using an inertial term but possessing less performance than the first approach.

Optical flow allows to detect brightness changes in moving objects, providing description of its motion which can be used to drive an active contour (Horn and Schunck, 1981, Peterfreund, 1997). This approach is able to perform tracking prediction, but it requires to process first a batch of images in order to provide estimates for an object of interest. These tracking capabilities are improved using a Kalman filter (Peterfreund, 1999). However, as the Kalman filter is only able to follow unimodal distributions this approach is still affected in its tracking by occlusion or clutter (Cham and Rehg, 1999, Drummond and Cipolla, 2002).

It must be noted that around this time Gradient Vector Flow Forces (GVF) were presented in (Xu and Prince, 1998b,a). These allow a snake to posses better concavity deformation and insensitivity to initialisation. However, the main drawback of GVF is its high computational cost per image and thus it is not widely used. This changed when another research proposed them instead of baseline forces for tracking (Lam and Lee, 2004). The conclusions obtained were that GVF increased algorithm efficiency avoiding distractions produced by other contours, but computational limitations required generating forces only around the object of interest. Another approach for avoiding distractions from outside contours in the snake involves constrained evolution (Sundaramoorthi et al., 2006). This uses Sobolev spaces for internal snake energy minimisation, allowing it to deform maintaining gradual but not sudden shape changes even in noise corrupted images.

These investigations suggest the idea of using active contours for tracking purposes in SLAM. However, particle filtering is seen as another alternative to introduce interactivity in this thesis thanks to flexible hypothesis generation. Therefore, the works that rely mainly on cameras and particle filtering are seen next. This is focused on research containing cameras as here they are considered an important feedback mechanism for interactivity.

2.3.2 Tracking with Particle Filtering

Other estimation techniques are more flexible for approximating complex parameters like camera pose or feature appearance compared to the EKF, such as particle filtering or sequential Monte Carlo methods (Gordon et al., 1993). These use a multi-hypothesis approach in which each particle can represent a different feature or robot pose, with voting deciding which ones remain and which ones are discarded (Särkkä, 2013). As such, particle filtering is a widely used approach in many areas. In SLAM it is often seen when the performance of estimators like EKF is not satisfactory enough, or when the chosen estimator can be improved by incorporating it into the algorithm.

One of the first implementations of particle filtering in SLAM considered only sonar sensors and later cameras (Montemerlo and Thrun, 2003, Eade and Drummond, 2006). This demonstrated advantages by using a hypothesis approach to infer location, which is more suitable when aggregating a considerable amount of landmarks. This is because in particle filtering there is need to update a whole matrix as in EKF, which contains all uncertainties of the landmarks.

Erratic motion is often the cause of divergence in estimators like EKF (Pupilli and Calway, 2006, Mirabdollah and Mertsching, 2012). In this case the particle filter is used to relocate the camera when experiencing erratic motion, stopping mapping in order to first correct camera pose. These investigations are of particular interest as the particle filter allows multiple hypotheses, which represent different camera locations as in (Pupilli and Calway, 2005). To obtain a valid position voting is performed, in which the best hypothesis is able to perceive all registered landmarks (Figure 2.6).

Other works have improved over feature stability or association (Gil et al., 2006, Tomono, 2007, Lee et al., 2007). These consider several camera pose hypotheses according to multiple views, epipolar geometry or image sequences. However, this research uses particle filtering mainly to accommodate dense feature acquisition but benefiting only pose estimation. Other works have employed a similar approach for loop closing in maps (Elinas et al., 2006, Pradeep et al., 2009). This allows the robot to return towards a predefined place, then particle filtering gives hypotheses for loop closure which can be used in large area maps or mini-maps that can be stitched together.

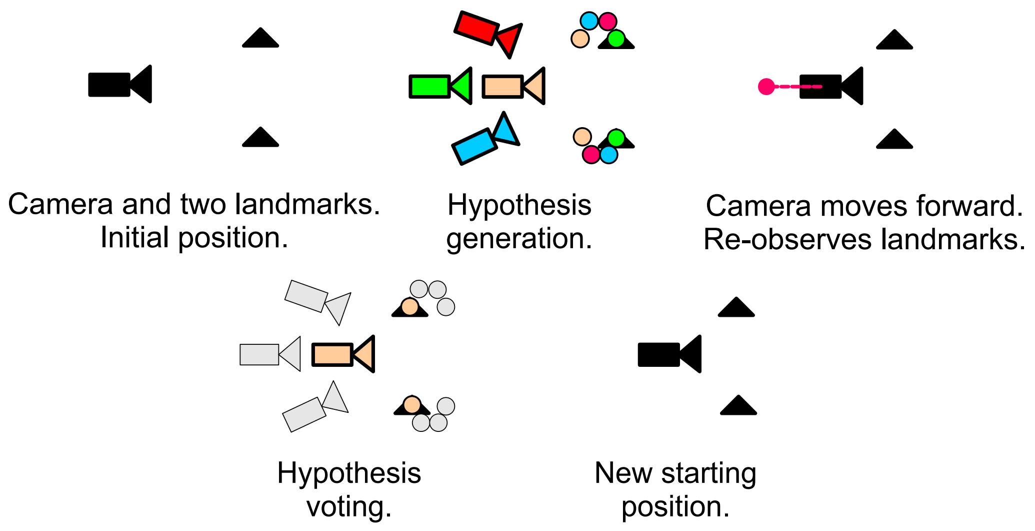


Figure 2.6: Simplified particle filtering camera principle based on (Pupilli and Calway, 2005). First there is an initial position where the camera starts, features are assumed available in the surroundings. From the initial position several hypothesis are generated, representing a possible camera position after movement as well as their respective possible observations. The camera finally moves and then obtains landmark measurements which then will be matched, with each of the predicted observations from the previously generated hypotheses. Only the hypothesis with the most votes is considered for the next iteration, generating more hypotheses from it. Note that in this example an unimodal distribution is considered for the camera position. However, this often does not follow a Gauss distribution and rather it takes a more complex form.

Visual cues have been used in SLAM to aid mapping whilst using particle filtering (Kundu et al., 2011, Skoglar and Törnqvist, 2012, Lowry et al., 2013). These systems take references from moving objects, known road networks or even places which can be the same place but present environmental changes. Multi sensor fusion approaches in SLAM have been researched involving particle filtering (Bleser and Stricker, 2008, Schroeter and Gross, 2008, Moemeni and Tatham, 2014, Silveira Vidal et al., 2015). These approaches show flexibility in sensor fusion, including inertial measurement units, sonar and cameras which are also capable of capturing depth. However, these approaches are still related to landmark association using particle filtering in dense feature acquisition scenarios.

Much of this investigation does not consider particle filtering besides improvements in camera or feature estimation. However, particle filtering is a very flexible approach. This is because it relies on hypotheses which can be voted on according to any criteria assigned to them, thus its flexibility on both selection and scoring. This kind of voting can be used for interactive scenarios in SLAM, as an user can point a landmark and modify its parameters whilst the camera is in motion. This is not limited to localised features (points) and instead more abstract landmarks can be used, as assuming that they can be voted they are suitable for pose estimation.

2.4 Concluding Remarks

Despite being implied that both localisation and mapping happens simultaneously in SLAM, what the robot does first is to map in order to reference itself. Indeed even the name SLAM was different in early approaches being SMAL, Simultaneous Map building And Localisation or CML, Concurrent Mapping and Localisation (Leonard and Durrant-Whyte, 1991, Durrant-Whyte and Bailey, 2006). This is very important as the robot never really knows where it is from the beginning, its initial position is just assumed to have little uncertainty. From there on, the robot performs mapping by observing adjacent features and references itself with them.

The general literature of SLAM revealed research concentrated on better landmark position estimation, data association, feature depth modelling, quantity of features and performance improvements at the cost of functionality. However, it also shown a lack of user interaction and real-time object representation in SLAM algorithms. This can be explained by a continuous push towards automated approaches, in which theoretically information can be stored and manipulated in big quantities.

There are works that go beyond just SLAM, i.e. those not directly trying to offer an alternative to current SLAM solutions. These include approaches involving feature abstraction: lines, planes, ceilings or walls. This is done in order to avoid localised features that offer weak feature association. This is further taken forward with curved walls or riverine areas using curve fitting techniques. Offline database recognition is also pushed forward as having prior information of a feature helps in matching as well.

However, it was shown that fully autonomous algorithms might lead to problems associated with computing resources, often limiting real-time operation. Also worth of mention are foreign situations which are outside of algorithm programming, which might lead to robot malfunction or complete task interruption. Therefore, there is a need for an hybrid approach involving autonomous operation and high-level decision making. One way to achieve this is by introducing user interactivity in SLAM, which might also lead into artificial intelligence algorithms research for SLAM, focused onto objects in order to avoid dense mapping altogether.

Therefore, this thesis attempts to involve an user in a more active role within SLAM. However, first a methodology is needed to incorporate user input. This research proposes active contours coupled with curve fitting techniques, allowing contour evolution to start in a nearby shape selected by an user. This would allow to exploit a snake's deforming capabilities in order to track silhouettes, which would be produced by objects of interest. Particle filtering is also proposed in which a person actively modifies the properties of an object of interest, with particles as hypothesis of the camera pose. This would allow to see how good an user choice of object parameters is, based on camera movements reflected in perspective changes in the object. These propositions take form by several key factors:

- In many works feature extraction is limited by automatic approaches, often in the form of points. Dense representations are required if detail is needed, but this increases also the computational power required (Stasse et al., 2006).
- Systems enabling goals communication and algorithm information are more

robust and better performing than those that do not (Fong et al., 2006). Often there are persons evaluating the SLAM performance in a robot or protecting the hardware, but a more active role outside virtual reality scenarios is not yet present allowing for improvement.

- Persons often segment objects better than machines (Datta et al., 2008). This ability comes from experience and contact with a variety of objects through the years, which allow to offload feature selection to an operator. This saves computational resources and it is not limited to offline database construction.
- Whereas there is not an investigation discussing user interaction at algorithm level, there is an investigation that really considers user input, providing semantics in the form of annotations in (Reitmayr et al., 2007). However, it does not incorporate an user into the SLAM algorithm and rather includes the annotations as an afterthought. This is akin to other augmented reality approaches were graphics overlays are put over the image in real-time, but this shows than an user can give valuable information for the robot and perhaps other collaborative users as well.

The approach presented here first elaborates over a simple implementation using shape silhouettes, in order to observe the feasibility of user interaction into SLAM. However, first the monocular SLAM foundations are described in the next chapter, as these are often assumed known in much of the related investigation.

Chapter 3

Bayesian SLAM and Cameras

As a whole, the aim of this thesis is to introduce interactivity into SLAM. It is then proposed first to explore in more detail the inner workings of this methodology, more precisely setting the focus on vision-SLAM. This is as images offer a good feedback mechanism for interactivity. At the same time, the links between feature acquisition, images and the EKF are often assumed known and hence overlooked in many investigations.

Therefore, this chapter begins by recalling the idea from Section 2.1 in which a mobile robot is capable of perceiving its surroundings through sensed landmarks. A first step taken by the robot is to rely on its odometry or any other sensors which allow modelling, with the intention of obtaining a predicted future state or robot pose; this is possible given that there is a time interval, a motion model and a previous robot state or position.

This is a process assumed susceptible to error, e.g. slippage in wheels can affect the odometry. Therefore, the robot also looks for features in the surroundings, in order to find a second reference for motion estimation based on environment observation. However, if the robot moves and no features are found the uncertainty in prediction accumulates. This is as its previous state already possesses an assumed error, from which a prediction is also obtained. This yields increased uncertainty based on the previous robot pose.

When a new landmark is found the current uncertainty in robot pose affects this observation, as it is uncertain about its position and the newly found feature references from the current robot pose. As a consequence, correlation occurs between the uncertainties in landmark and robot pose (Smith et al., 1986).

At the same time, when using dedicated sensors for observing landmarks there is an error in the measurements as they are assumed not ideal. In some cases the obtained reading does not directly relate to a position. Therefore, a model is used to obtain a desired result from the observed quantity. This linearisation can also be presented in the motion model. In both cases, if it is less than ideal the linearisation can introduce further uncertainty into the system.

An elegant way to cope with this uncertainty correlation is by means of Bayesian or probabilistic techniques, such as the Kalman Filter or particle filtering. In the case of SLAM where model linearisation is involved, the Extended Kalman Filter (EKF) can be used. Much investigation has been made in SLAM and vision-SLAM parting from EKF, but these often assume the reader knowledgeable in the complex processes involved for pose prediction and feature acquisition.

Therefore, this chapter gives detail of the EKF vision-SLAM methodology, considering models for predicting feature appearance using inverse depth parametrisation (Civera et al., 2008). This is done with the intention of finding links for an interactive approach within SLAM:

- A brief introduction to the EKF is presented, in which the general idea of using a state vector and covariance matrix is explored. This includes prediction and update steps involving features extracted from images.

- An often overlooked compilation of all the methodology required to perform vision-SLAM, including several steps taken in image feature acquisition: the lens pin-hole model, accounting for image lens deformations, camera motion model and landmark prediction. Also considered is feature registration based on camera pose into the state vector and covariance matrix of the EKF.

- The use of inverse depth parametrisation for three dimensional feature representation, which allows to reduce non-linearities affecting the performance of the EKF (Civera et al., 2008). However, many principles explored in this chapter can be used in other estimation techniques, such as particle filtering.

These foundations allow to explore in which form interactivity can be introduced, in a way that allows data to be obtained from user input and used for

SLAM. The presented contribution includes the mathematical foundations behind monocular SLAM, which are presented in a detailed manner.

Therefore, Section 3.1 provides a brief overview of the EKF algorithm, describing both the additive and non-additive noise formulations in Sections 3.1.1 and 3.1.2 respectively. These include the general equations, which can be applied as an initial framework for estimation and sensor fusion in non-linear systems.

The EKF in SLAM has two main stages: adding features into the state and covariance, as well as robot and feature prediction with update after observations. The first stage is explored in Section 3.2 which involves feature acquisition in Section 3.2.1, considerations on lens deformations in Section 3.2.2 and finally, adding features to EKF's state vector and covariance matrix in Sections 3.2.3 and 3.2.4.

The second stage can be seen in Section 3.3 involving camera state and covariance prediction in Section 3.3.1, from which feature appearance position and uncertainty predictions using the inverse depth parametrisation model are made in Sections 3.3.2 and 3.3.3. In the last part of the section, feature observations are obtained and verified to be from the same sample in Section 3.3.4. The deviations from their respective predictions allow to perform EKF update, affecting both state and covariance matrices as seen in Section 3.3.5. Finally an implementation of this approach is shown in Section 3.4, with concluding remarks described in Section 3.5.

3.1 The Extended Kalman Filter

The Kalman Filter is an estimator capable of dynamically producing estimates from different sensor sources, which are assumed to produce a certain amount of error in their readings. The resulting estimate is better than only using the noisy measurements from the sensors (Särkkä, 2013, Zarchan and Musoff, 2009).

This section briefly explores the two steps involved in the EKF, which are similar for the Kalman Filter but with included linearisation. In the vision-SLAM context, the first step obtains predictions of both camera and landmarks positions (state variables) and the corresponding uncertainties. A second step delivers estimates, which update the predictions using noisy landmark positions from the camera (noisy observations).

The usefulness of a Kalman Filter relies in its capability to dynamically weight a prediction and an observation. This is done according to their uncertainty, e.g. if the camera prediction has a sizeable uncertainty, the filter delivers an estimate which relies more on the landmark observations than in the camera motion model. Similarly if the landmarks observations are too noisy, the resulting estimates favour the prediction made by the camera motion model.

3.1.1 Additive Noise Formulation

An EKF with assumed additive noise uses a state-space model, which represents state (camera and landmarks positions) and measurements (landmark observations) in the following recursive form (Särkkä, 2013):

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{q}_{k-1} \tag{3.1}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{r}_k \tag{3.2}$$

where the sub-index k denotes a current iteration, $\mathbf{x}_k$ is the state vector containing the variables of interest, $\mathbf{z_k}$ are the measurements. $\mathbf{q}_{k-1}$ and $\mathbf{r}_k$ are zero mean Gaussian $\mathbf{N}(\cdot,\cdot)$ state and measurement noises $\mathbf{N}(\mathbf{0},\mathbf{Q}_{k-1})$ and $\mathbf{N}(\mathbf{0},\mathbf{R}_k)$. Noise covariances are contained for state and measurement in the matrices $\mathbf{Q}_{k-1} = \mathbf{E}(\mathbf{q}_{k-1}\mathbf{q}_{k-1}^\top)$ and $\mathbf{R}_k = \mathbf{E}(\mathbf{r}_k\mathbf{r}_k^\top)$ in which $\mathbf{E}(\cdot)$ denotes expectation. The state model and measurement model are represented by $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ respectively.

The probability density of the state $\mathbf{x}_k$ given all the measurements $\mathbf{z}_{1:k}$ is approximated in the EKF, by using a Gaussian distribution with mean $\mathbf{m}_k$ and covariance $\mathbf{P}_k$:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \backsimeq \mathbf{N}(\mathbf{x}_k|\mathbf{m}_k,\mathbf{P}_k) \tag{3.3}$$

Prediction of $\mathbf{m}_k$ and $\mathbf{P}_k$ is performed using

$$\mathbf{m}_k^- = \mathbf{f}(\mathbf{m}_{k-1}) \tag{3.4}$$

$$\mathbf{P_k^-} = \mathbf{F_x}(\mathbf{m}_{k-1})\mathbf{P}_{k-1}\mathbf{F_x}(\mathbf{m}_{k-1})_x^\top + \mathbf{Q}_{k-1} \tag{3.5}$$

where $\mathbf{F_x}$ is obtained by evaluating the Jacobian of $\mathbf{f}(\mathbf{x})$ with $\mathbf{x} = \mathbf{m}_{k-1}$

$$[\mathbf{F_x}(\mathbf{m}_{k-1})]_{i,j} = \left.\frac{\partial f_i(\mathbf{x})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}}$$

The update of $\mathbf{m}_k$ and $\mathbf{P}_k$ requires the error (difference) $\mathbf{v}_k$ between the observations $\mathbf{z}_k$ and their predictions $\mathbf{h}(\mathbf{m}_k^-)$, also requires the innovation covariance $\mathbf{S}_k$ and the Kalman gain $\mathbf{K}_k$:

$$\mathbf{v}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{m}_k^-) \tag{3.6}$$

$$\mathbf{S}_k = \mathbf{H_x}(\mathbf{m}_k^-)\mathbf{P}_\mathbf{k}^-\mathbf{H_x}^\top(\mathbf{m}_k^-) + \mathbf{R}_k \tag{3.7}$$

$$\mathbf{K}_k = \mathbf{P}_\mathbf{k}^-\mathbf{H_x}(\mathbf{m}_k^-)^\top\mathbf{S}_k^{-1} \tag{3.8}$$

where $\mathbf{H_x}$ is obtained evaluating the Jacobian of $\mathbf{h}(\mathbf{x})$ with $\mathbf{x} = \mathbf{m}_{k-1}$

$$[\mathbf{H_x}(\mathbf{m})]_{i,j} = \left.\frac{\partial h_i(\mathbf{x})}{\partial h_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}}$$

Overall $\mathbf{P}_k^-$ indicates the uncertainty of the state prediction, thus if this covariance matrix is big then the state prediction is assumed to have high uncertainty. This is reflected in an increase of the Kalman gain $\mathbf{K}_k$, making the filter trust more the obtained measurements. On the other hand the elements of $\mathbf{S}_k$ indicate the uncertainty in the measurements, thus if this innovation covariance matrix is big the confidence over the observations will be low. Hence, this renders also the Kalman gain low causing the state predictions to be trusted more. Finally the values of $\mathbf{m}_k$ and $\mathbf{P}_k$ can be updated using Equations (3.6), (3.7) and (3.8):

$$\mathbf{m}_k = \mathbf{m}_k^- + \mathbf{K}_k\mathbf{v}_k$$
$$\mathbf{P}_k = \mathbf{P}_\mathbf{k}^- - \mathbf{K}_k\mathbf{S}_k\mathbf{K}_k^\top$$

3.1.2 Non-Additive Noise Formulation

SLAM often uses the EKF with non-additive noise formulation, including its inverse depth parametrisation approach (Civera et al., 2008). This is a further generalisation of the EKF which can be obtained using a non-additive noise formulation, accounting also for signal based noises (Iickho Song, 2002). In this scenario the EKF model changes to:

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{q}_{k-1}) \tag{3.9}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{r}_k) \tag{3.10}$$

where $\mathbf{q}_{k-1}$ and $\mathbf{r}_k$ are again also zero mean Gaussian state and measurement noises $\mathbf{N}(\mathbf{0}, \mathbf{Q}_{k-1})$ and $\mathbf{N}(\mathbf{0}, \mathbf{R}_k)$. The state and measurement models are therefore represented in the same way with $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$.

The prediction step changes accordingly for $\mathbf{m}_k$ and $\mathbf{P}_k$ to

$$\mathbf{m}_k^- = \mathbf{f}(\mathbf{m}_{k-1}, \mathbf{0}) \tag{3.11}$$

$$\mathbf{P}_{\mathbf{k}}^- = \mathbf{F}_\mathbf{x}(\mathbf{m}_{k-1})\mathbf{P}_{k-1}\mathbf{F}_\mathbf{x}(\mathbf{m}_{k-1})_x^\top + \mathbf{F}_\mathbf{q}(\mathbf{m}_{k-1})\mathbf{Q}_{k-1}\mathbf{F}_\mathbf{q}(\mathbf{m}_{k-1})^\top \tag{3.12}$$

where $\mathbf{F}_\mathbf{x}$ and $\mathbf{F}_\mathbf{q}$ are obtained by evaluating the Jacobian of $\mathbf{f}(\mathbf{x}, \mathbf{q})$ with $\mathbf{x} = \mathbf{m}_{k-1}$ and $\mathbf{q} = \mathbf{0}$

$$[\mathbf{F}_\mathbf{x}(\mathbf{m}_{k-1})]_{i,j} = \left.\frac{\partial f_i(\mathbf{x}, \mathbf{q})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}, \mathbf{q}=\mathbf{0}} \tag{3.13}$$

$$\left[\mathbf{F}_\mathbf{q}(\mathbf{m}_{k-1})\right]_{i,j} = \left.\frac{\partial f_i(\mathbf{x}, \mathbf{q})}{\partial q_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}, \mathbf{q}=\mathbf{0}} \tag{3.14}$$

Similarly to an EKF with additive noise formulation, the update of $\mathbf{m}_k$ and $\mathbf{P}_k$ requires the error $\mathbf{v}_k$ between measurements $\mathbf{z}_k$ and their predicted observations $\mathbf{h}(\mathbf{m}_k^-, \mathbf{0})$, needing as well innovation covariance $\mathbf{S}_k$ and the Kalman gain $\mathbf{K}_k$ but accounting for non additive noise:

$$\mathbf{v}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{m}_k^-, \mathbf{0}) \tag{3.15}$$

$$\mathbf{S}_k = \mathbf{H}_\mathbf{x}(\mathbf{m}_k^-)\mathbf{P}_\mathbf{k}^-\mathbf{H}_\mathbf{x}{}^\top(\mathbf{m}_k^-) + \mathbf{H}_\mathbf{r}(\mathbf{m}_k^-)\mathbf{R}_k\mathbf{H}_\mathbf{r}{}^\top(\mathbf{m}_k^-) \tag{3.16}$$

$$\mathbf{K}_k = \mathbf{P}_\mathbf{k}^-\mathbf{H}_\mathbf{x}(\mathbf{m}_k^-)^\top\mathbf{S}_k^{-1} \tag{3.17}$$

where $\mathbf{H}_\mathbf{x}$ and $\mathbf{H}_\mathbf{r}$ are obtained by evaluating the Jacobian of $\mathbf{h}(\mathbf{x}, \mathbf{r})$ with $\mathbf{x} = \mathbf{m}_{k-1}$

and $\mathbf{r} = 0$

$$[\mathbf{H_x}(\mathbf{m}_{k-1})]_{i,j} = \left.\frac{\partial h_i(\mathbf{x}, \mathbf{r})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}, \mathbf{r}=\mathbf{0}} \tag{3.18}$$

$$[\mathbf{H_r}(\mathbf{m}_{k-1})]_{i,j} = \left.\frac{\partial h_i(\mathbf{x}, \mathbf{r})}{\partial r_j}\right|_{\mathbf{x}=\mathbf{m}_{k-1}, \mathbf{r}=\mathbf{0}} \tag{3.19}$$

Finally the values of $\mathbf{m}_k$ and $\mathbf{P}_k$ can be updated using Equations (3.15), (3.16) and (3.17):

$$\mathbf{m}_k = \mathbf{m}_k^- + \mathbf{K}_k \mathbf{v}_k \tag{3.20}$$

$$\mathbf{P}_k = \mathbf{P}_{\mathbf{k}}^- - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top \tag{3.21}$$

More information about the derivation of these equations, as well as further discussion of estimation algorithms can be found in (Särkkä, 2013). A more detailed and pragmatic approach can be found in (Zarchan and Musoff, 2009).

The following sections then detail a vision-SLAM algorithm with using inverse depth parametrisation, whilst relating it to the EKF equations presented in this section. These also describe some links in vision-SLAM which are independent of the selected estimation method, be either EKF or particle filtering and which can give insight for an interactive approach in SLAM.

3.2 Inverse Depth Monocular SLAM: Adding Features to State and Covariance

SLAM is a structured methodology in which each of the stages represent an important part within the whole process. For monocular SLAM, the basic setup includes a camera connected to a computer, Figure 3.1. The environment in which the camera is placed is assumed to have available salient features, i.e. not textureless surfaces.

Monocular SLAM relies on performing localisation and mapping using only a single camera as a sensor, as well as the principles expressed in Section 3.1. The inverse depth parametrisation aids landmark estimation by using 6 parameters, instead of 3 from an Euclidean representation. This exploits the parallax effect,

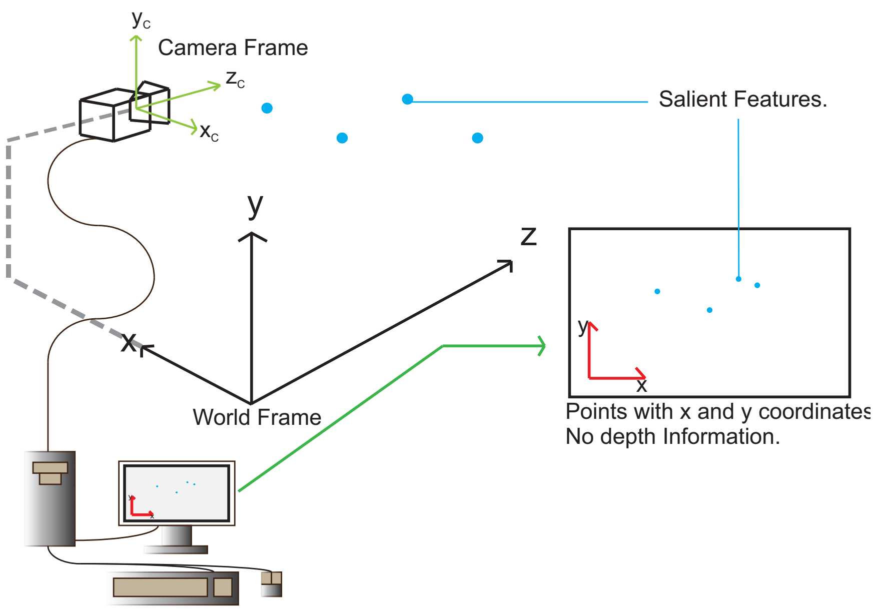


Figure 3.1: Monocular SLAM setup. A single camera is used to observe the surroundings obtaining salient features, the latter become landmarks when added into the EKF state vector and covariance. Depth information is not directly available from images, as such only two dimensional coordinates are obtained.

in which a farther feature will appear to move less from the centre compared to a closer one. Hence, helping in non linearities and allowing for fast initialisation (Civera et al., 2008).

The EKF methodology is used in SLAM by assuming a probability density for both camera and landmarks, depending on feature measurements as in Equation (3.3). As such, landmark acquisition is the first important step in monocular SLAM. Camera and landmark positions are assumed to have a mean, representing an estimate of their location and a covariance, describing the uncertainty in said estimate. When adding features a correlation between camera and landmark uncertainty occurs, as feature observation is affected by the camera uncertainty. Once a landmark has been added it remains static as the feature is assumed not to move.

All of the uncertainties are then stored in the covariance matrix, which initially only has values for all camera states and later on for the acquired landmarks. It must be noted that landmarks can be deleted, from the state and covariance if they present low observability or by going out of range. In such case new features

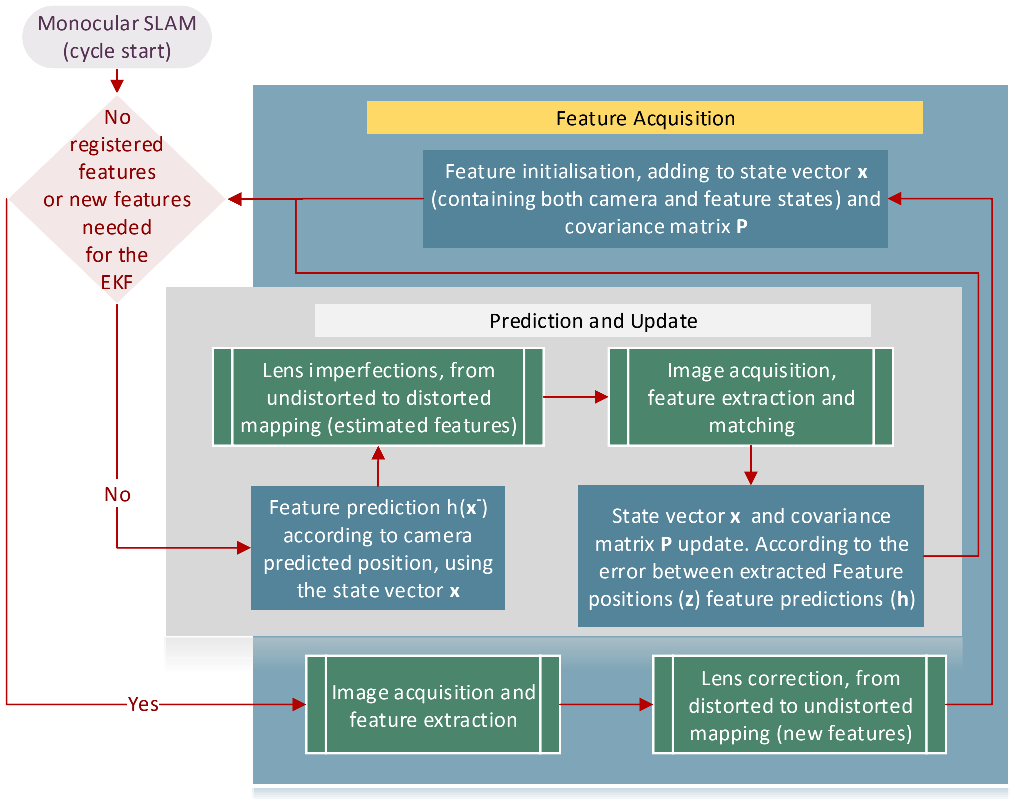

Monocular SLAM (cycle start)
No registered features or new features needed for the EKF
No
Yes
Feature Acquisition
Feature initialisation, adding to state vector **x** (containing both camera and feature states) and covariance matrix **P**
Prediction and Update
Lens imperfections, from undistorted to distorted mapping (estimated features)
Image acquisition, feature extraction and matching
Feature prediction h(**x**⁻) according to camera predicted position, using the state vector **x**
State vector **x** and covariance matrix **P** update. According to the error between extracted Feature positions (**z**) feature predictions (**h**)
Image acquisition and feature extraction
Lens correction, from distorted to undistorted mapping (new features)

Figure 3.2: Monocular SLAM simplified diagram with feature acquisition group. Blue rectangles signal EKF steps while green ones represent stages inherent to vision-SLAM. The feature acquisition group shows how features are added as needed, after lens deformations have been removed. Correction of lens imperfections is needed in order to add salient features, as their initial 3D positions do not involve those deformations. Afterwards prediction and update in the EKF's state and covariance is performed.

are sought, maintaining a predefined amount of landmarks in the EKF.

The following sections focus on the aspects involved for feature acquisition: From preparing an image in order to use automated feature detectors, accounting for image deformations produced by lens imperfections, and their aggregation as landmarks in the EKF's state vector and covariance matrix. These steps are summarised in the simplified monocular representation seen in Figure 3.2.

3.2.1 Feature Acquisition

Vision research has always relied on visual cues obtained from images, which are often distinguishable from other elements in it. Early examples demonstrate this by detecting changes in brightness to describe object motion or through extraction of high contrast points (Horn and Schunck, 1981, Shi and Tomasi, 1994). A good part of vision-SLAM takes this concept further by adding a condition of repeatability, i.e. the same visual cues must be detected even after changes in position or orientation. Hence, the more repeatable a feature is the better the camera referencing becomes.

The algorithm of monocular SLAM with inverse depth parametrisation makes use of local features in mapping and matching tasks. The former allows to perform SLAM as the obtained features serve for map creation, the latter matches an added landmark with an observation from a different camera pose to perform motion estimation (Civera et al., 2008).

Often feature acquisition is made using automated algorithms such as SIFT, SURF or FAST, which deliver points with resilience to variations in scale, orientation and illumination (Skrypnyk and Lowe, 2004, Bay et al., 2008, E. Rosten and Drummond, 2010). As such, feature detection algorithms possess great importance within vision-SLAM, as the more reliable they are the better referencing is achieved.

Automated feature detectors are continuously investigated for SLAM and many other vision applications, examples include BRISK, ORB, KAZE and AKAZE (Leutenegger et al., 2011, Rublee et al., 2011, Alcantarilla et al., 2012, F. Alcantarilla et al., 2013). Of particular interest in this thesis is the AKAZE salient feature detector, as it is a new and reliable approach which is already integrated in the commonly used vision processing package OpenCV[*]. The latter offers a foundation for fast execution of vision algorithms, possessing an interface coded in the programming language C++, which is also used in this investigation due to its speed.

The first step for feature acquisition consists of image simplification, as an image provided by a camera consists of many intensity values. These often go from

[*]Open Source Computer Vision is a package of libraries dedicated to vision algorithms, supporting a variety of languages and platforms (OpenCV, n.d.)

0 to 255 in three channels: red (R), green (G) and blue (B). Therefore a transformation from RGB to greyscale intensities is often applied over an image using the formula:

$$\text{Greyscale intensity} = R * 0.299 + G * 0.587 + B * 0.114 \tag{3.22}$$

which produces a greyscale image, with only 1 channel containing intensity values. This image is the input for the chosen feature detector, which then analyses it in order to find features with invariant characteristics. The result of this is simply a point coordinate, which if observed with the same salient feature detector from a pose, orientation or lighting variation should deliver the same point but within a new image.

The process to acquire landmarks for SLAM often follows this procedure: first a random region within an image (already greyscale) is selected, considering margins from its total horizontal and vertical resolution. From there, the selected salient feature detector obtains an invariant point, if it does not find one then another random region is selected in the same image. Often points are extracted from object corners, high contrast or distinctive parts of an image. However, in the case of walls with plain or smooth colour there will be not salient feature detection. As such it is important for the image not to be too uniform.

Image deformations must be accounted after a feature is extracted. This is because lens imperfections are present in wide angle optics, which are preferred in SLAM for their big field of view. Therefore, the next section addresses this by using radial models.

3.2.2 From Distorted to Undistorted Mapping and Vice Versa

Vision tracking applications benefit from wide fields of view, because any object of interest remains more time in image when camera displacements occur. This is as association or the ability to recognise an object again has a direct effect in tracking performance. Because of this, vision SLAM has also shown improvements yielding better camera orientation estimates (Schleicher et al., 2006).

A bigger field of view can be accomplished in two ways: 1. Increasing the image sensor area; 2. Reducing the focal length of the lens. The former allows to have longer focal lengths that still cover a wide area and are less difficult to

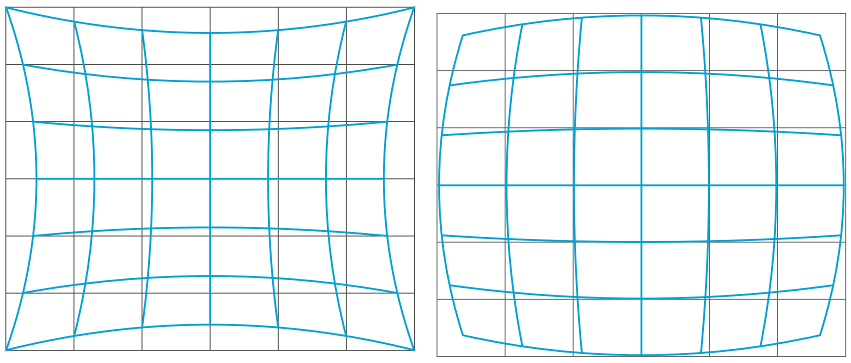


Figure 3.3: Image pincushion and barrel imperfections. Each camera lens is different, thus also its produced image. Most common corrections are performed for pin-cushion (a) or for barrel (b) distortions. Lens imperfections can be corrected by either adding more lenses (optical correction) or by software post processing.

manufacture, whereas the latter permits to create small cameras like the ones in mobile phones. However, both options present advantages and disadvantages compared to each other. Larger sensors allow for less image noise, but bigger optics are required to fill the entire image frame making them more expensive. Shorter focal lengths offer more field of view in smaller sensors (at the cost of more image noise), but optics are very difficult to create at shorter focal lengths without producing a deformed image, often requiring many optical elements to partially correct it.

Despite this, cameras with wide angle lenses are found in webcams with enough picture quality. These are commonly used in vision-SLAM thanks to their recent affordability. However, the image is still affected by distortion, which tends to present itself as barrel, pincushion, or a mixture of both, as seen on Figure 3.3.

Correcting distortions from coordinates $\mathbf{h_d} = [X_{cam}, Y_{cam}]^\top$ or the deviations from rectilinear projections is important, as vision-SLAM does not account for these, often this is left as aggregated steps for feature acquisition and prediction in the SLAM algorithm. In both of these cases the Brown-Conrady model is used to deliver undistorted coordinates $\mathbf{h_u} = \left[h_{u_x}, h_{u_y}\right]^\top$ as it has good invertible properties*, i.e. it is possible to remove lens deformations for incorporating a salient feature into SLAM, as well as adding imperfections to said feature in order to

*This particular step is inverted in Civera's Monocular SLAM code, which can be obtained from *http://openslam.org/*

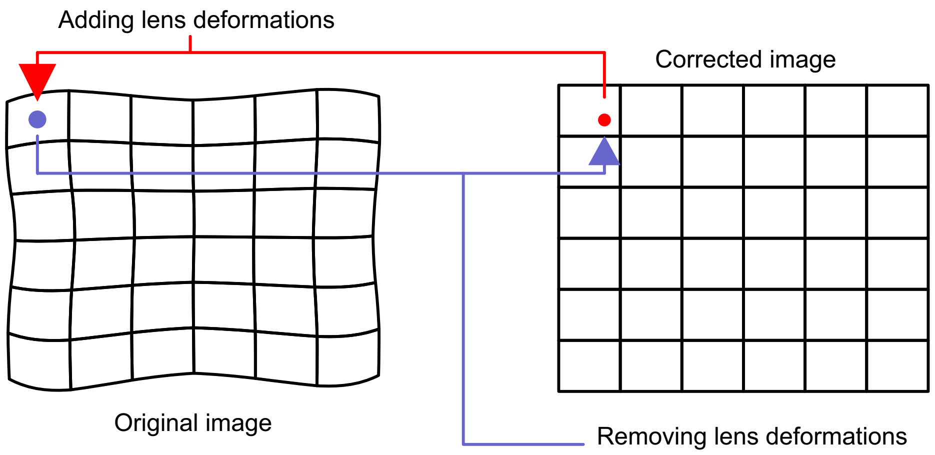


Figure 3.4: An example of removing and adding image distortion in features. A deformed mesh can be seen on the left hand side, showing a mixture of barrel and pincushion deformations commonly found on wide angle lenses. The right hand side depicts a corrected image using a Brown-Conrady model. The blue point in the original image gets displaced as part of this, seen as the red dot in the corrected image. The Brown-Conrady model allows to move between from the original and the corrected representations, particularly used to add and predict landmarks.

predict the appearance of it over an image (Civera et al., 2008).

Radial distortions can be considered using more than one coefficient, which allows to correct more complex image deformations. However, numerical approximations are needed for good results with two coefficients, whereas a model with one coefficient will allow for a direct analytical solution (Brown, 1966). An example of removing and adding deformations can be seen in Figure 3.4.

Predicting Features, Adding Distortion

Inverse depth parametrisation produces estimates for landmarks in 6 parameters using EKF. The algorithm uses a transformation which projects these landmarks onto a two dimensional representation, allowing to perform landmark position prediction over a new image. However, this representation does not account for image imperfections produced by wide angle lenses.

The undistorted coordinates of the predicted landmark $\mathbf{h_u} = \left[h_{u_x}, h_{u_y}\right]^\top$ are first transformed to normalised undistorted coordinates $\mathbf{n_u} = \left[n_{u_x}, n_{u_y}\right]^\top$ using the following conversion:

$$n_{u_x} = \frac{(h_{u_x} - X_c)}{f_x} \qquad n_{u_y} = \frac{(h_{u_y} - Y_c)}{f_y} \tag{3.23}$$

where the principal points X_c, Y_c together with the focal lengths f_x, f_y must be obtained using camera calibration software, e.g. using a camera calibration toolbox (Matlab).

Therefore, for a normalised pixel mapping going from undistorted $\mathbf{n_u}$ to a distorted $\mathbf{n_d} = \left[n_{d_x}, n_{d_y}\right]^\top$ representation (adding image imperfections produced by wide angle lenses) the following formulae are used:

$$n_{d_x} = n_{u_x}(1 + K_1 r_d^2 + K_2 r_d^4) \qquad n_{d_y} = n_{u_y}(1 + K_1 r_d^2 + K_2 r_d^4) \tag{3.24}$$

with r_d defined for this mapping as:

$$r_d = \sqrt{n_{u_x}^2 + n_{u_y}^2} \tag{3.25}$$

and the coefficients K_1, K_2 are obtained also with camera calibration software.

Transforming the normalised distorted coordinates $\mathbf{n_d} = \left[n_{d_x}, n_{d_y}\right]^\top$ obtained from Equation (3.24) to distorted coordinates $\mathbf{h_d} = [X_{cam}, Y_{cam}]^\top$, is used to match a predicted landmark with a salient feature over a new camera image. For this case the following formulae apply:

$$X_{cam} = n_{d_x} f_x + X_c \qquad Y_{cam} = n_{d_y} f_y + Y_c \tag{3.26}$$

where X_{cam} and Y_{cam} are pixel coordinates straight from the camera image.

Adding Features, Removing Distortion

In the SLAM process the features are usually added first in order to perform prediction, Figure 3.2. Here this step was deliberately left afterwards, as adding distortion is a more direct method than removing it.

Extracted features $\mathbf{h_d} = [X_{cam}, Y_{cam}]^\top$ possess distorted coordinates as they are directly extracted from a new camera image. Similar to Section 3.2.2 the extracted features need to be first transformed into normalised distorted coordinates

$\mathbf{n_d} = \left[n_{d_x}, n_{d_y}\right]^\top$, which is done using the following conversion:

$$n_{d_x} = \frac{(X_{cam} - X_c)}{f_x} \qquad n_{d_y} = \frac{(Y_{cam} - Y_c)}{f_y} \tag{3.27}$$

For an inverse mapping from a distorted to an undistorted representation (removing image imperfections produced by wide angle lenses), Equation (3.24) becomes:

$$n_{u_x} = \frac{n_{d_x}}{1 + K_1 r_u^2 + K_2 r_u^4} \qquad n_{u_y} = \frac{n_{d_y}}{1 + K_1 r_u^2 + K_2 r_u^4} \tag{3.28}$$

with r_u defined using (Zhang, 1999):

$$r_d = r_u(1 + K_1 r_u^2 + K_2 r_u^4) \tag{3.29}$$

Compared to a one coefficient model which has a direct solution (Devernay and Faugeras, 1995), Equation (3.29) comes from a two coefficient model. Therefore in order to solve for r_u a numerical solution is required such as Newton-Rhapson[*]. Once r_u has been found it can be plugged onto (3.28) in order to obtain the normalised undistorted coordinates $\mathbf{n_u} = \left[n_{u_x}, n_{u_y}\right]^\top$, which will eventually lead to undistorted coordinates $\mathbf{h_u} = \left[h_{u_x}, h_{u_y}\right]^\top$ using the following conversion:

$$h_{u_x} = n_{u_x} f_x + X_c \qquad h_{u_y} = n_{u_y} f_y + Y_c \tag{3.30}$$

where the principal points X_c, Y_c together with the focal lengths f_x, f_y are the same as obtained from the camera calibration software. Once the feature is free from any deformation it can be initialised in the SLAM algorithm, eventually allowing for prediction.

3.2.3 Adding Features to State

After a feature has been extracted and has been accounted for image deformations, it must be entered into the state and covariance matrix of the EKF in the

[*]Civera's code mentions that 10 iterations are sufficient to find a proper r_u value.

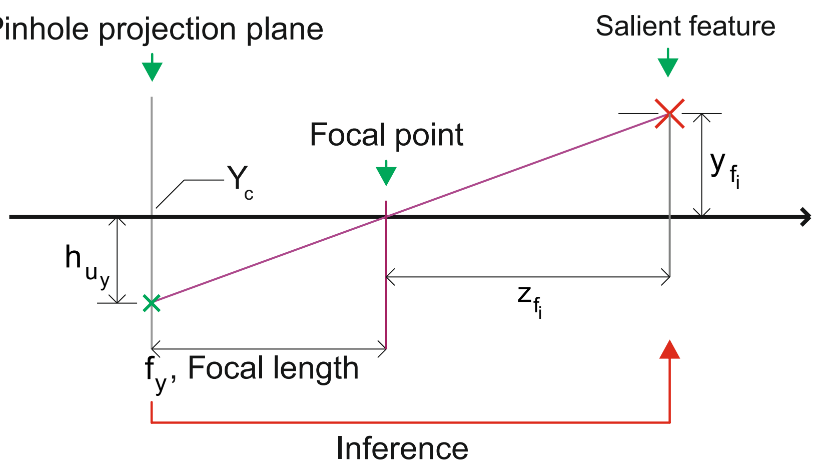


Figure 3.5: The pinhole camera model allows to infer three dimensional positions using an idealised camera model, by knowing parameters such as the focal lengths f_x, f_y and the principal points X_c and Y_c. In this example the pinhole camera model is seen considering only the YZ plane for simplification. A salient feature f_i (big red cross on the right) is seen by a hypothetical pinhole camera. Because of the model this feature is projected in a inverted way, towards the pinhole projection plane (small green cross on the left). This pin-hole projection of the salient feature is assumed with no lens deformations, which require applying first Equations (3.28) and (3.30) with a salient feature obtained from a camera image. Camera parameters will affect this projection as its height y_{f_i} and distance z_{f_i} will correspond to an inverted height h_{u_y}, considering the focal length f_y in the pinhole projection plane.

vision-SLAM algorithm. For the case of monocular SLAM this is an indirect process, since a three dimensional salient feature is projected onto a two dimensional camera image. This projection is assumed free of lens imperfections and is used to initialise a new landmark with 6 parameters, which first are obtained from a three dimensional representation inferred from the previous two dimensional projection. This inference can be performed using an idealised pin-hole camera model as seen in Figure 3.5.

For simplicity of notation $\mathbf{x}$ will represent both the state vector and mean of the EKF in all subsequent sections, i.e. $\mathbf{x}_k = \mathbf{m}_k$. This is because pragmatically the mean is the estimate of the state vector using the EKF assumption seen in Equation (3.3).

Therefore an extracted salient feature is assumed observed by an ideal pinhole camera, which is then projected onto the pinhole projection plane. As an opposite case the EKF makes use of an already added landmark for prediction, from the pinhole projection plane towards a camera frame coordinate system as in Figure 3.6. These transformations are given by:

$$\begin{aligned} \frac{x_{f_i}}{z_{f_i}} &= -\frac{h_{u_x}}{f_x} \\ \frac{y_{f_i}}{z_{f_i}} &= -\frac{h_{u_y}}{f_y} \end{aligned} \tag{3.31}$$

where h_{u_x}, h_{u_y} are the undistorted coordinates from the salient feature obtained through Equations (3.28) and (3.30). The coordinates x_{f_i}, y_{f_i} and z_{f_i} express the feature position in three dimensions in front of the idealised pinhole camera, which is also named camera frame. Note that the negative sign replicates the effect of a real pinhole camera, i.e. the projection is upside down (an entire image would look inverted). Removing the sign corrects this at the cost of becoming physically impossible to replicate, nevertheless it simplifies the model theoretically.

Given a pinhole model inference it is possible to initialise a new feature f_i. For simplification purposes the sign is removed in Equation (3.31) and z_{f_i} is set to 1 as the feature depth is not yet known, with the EKF assumed to eventually produce a better estimate of it thanks to inverse depth parametrisation. This leads to the initial feature coordinates in camera frame as seen in Figure 3.6:

$$\mathbf{h}^c = \begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} = \begin{bmatrix} \frac{h_{u_x} - X_c}{f_x} \\ \frac{h_{u_y} - Y_c}{f_y} \\ 1 \end{bmatrix} \tag{3.32}$$

Note the subtraction of both principal points X_c and Y_c. This is needed as it centres the pinhole projection plane, since image coordinates are often only positive.

After the new salient feature $\mathbf{h}^c$ as been expressed in terms of the camera frame with (3.32), it needs to be transformed into world frame coordinates. Compared to camera frame terms this helps to keep uncertainties down (Civera et al., 2010). In order to transform the feature, information from the camera state vector is

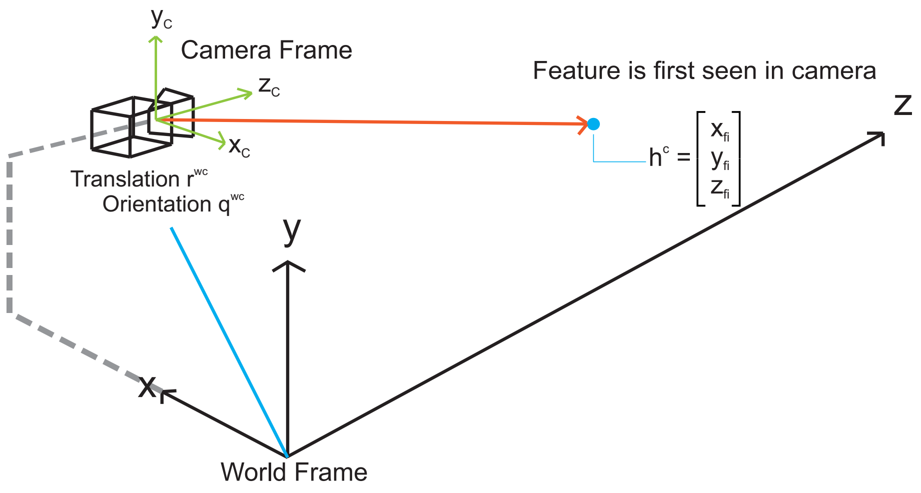

y_c
Camera Frame
z_c
x_c
Translation r^{wc}
Orientation q^{wc}
Feature is first seen in camera
$h^c = \begin{bmatrix} x_{fi} \\ y_{fi} \\ z_{fi} \end{bmatrix}$
Z
y
X
World Frame

Figure 3.6: Inverse Depth SLAM Feature Initialisation. In this scenario a camera with translation $\mathbf{r}^{WC}$ and orientation $\mathbf{q}^{WC}$ with respect to the world frame observes for the first time a salient feature, which is set in terms of the camera frame according to Equation (3.32).

needed:

$$\mathbf{x}_V = \begin{bmatrix} \mathbf{r}^{WC} \\ \mathbf{q}^{WC} \\ \mathbf{v}^{W} \\ \omega^{C} \end{bmatrix} \tag{3.33}$$

where $\mathbf{r}^{WC}$ is the estimated camera optical centre position w.r.t. the world frame (coordinates x, y, z in Figure 3.6), $\mathbf{q}^{WC}$ the estimated orientation quaternion w.r.t. the world frame, $\mathbf{v}^{W}$ is the estimated linear velocity w.r.t. the world frame and ω^{C} represents the estimated angular velocity w.r.t the camera frame.

Using the orientation quaternion from the camera state it is possible to generate a rotation matrix $\mathbf{ROT}_{WC}$ from the camera frame $\mathbf{h}^{C}$ to the world frame $\mathbf{h}^{W}$. This is done in order to remove the camera rotation, which might not be aligned with the world frame (Diebel, 2006):

$$\mathbf{ROT}_{WC} = \begin{bmatrix} q_r^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_r q_z) & 2(q_z q_x + q_r q_y) \\ 2(q_x q_y + q_r q_z) & q_r^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_r q_x) \\ 2(q_z q_x - q_r q_y) & 2(q_y q_z + q_r q_x) & q_r^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{3.34}$$

where q_r, q_x, q_y, q_z are the quaternion components of $\mathbf{q}^{wc}$, the current camera orientation estimation. Thus using Equations (3.32) and (3.34):

$$\mathbf{h}^{w} = \mathbf{ROT}_{wc}\mathbf{h}^{c} = \mathbf{ROT}_{wc}\begin{bmatrix} \frac{h_{u_x} - X_c}{f_x} \\ \frac{h_{u_y} - Y_c}{f_y} \\ 1 \end{bmatrix} \tag{3.35}$$

The last step consists of changing the feature parametrisation to inverse depth, which consists of 6 states:

$$\mathbf{y}_{f_i} = \begin{bmatrix} x_{f_i} & y_{f_i} & z_{f_i} & \theta_{f_i} & \phi_{f_i} & \rho_{f_i} \end{bmatrix} \tag{3.36}$$

which models a 3D point $\mathbf{x_i}$ in camera frame

$$\mathbf{x}_i = \begin{bmatrix} X_{f_i} \\ Y_{f_i} \\ Z_{f_i} \end{bmatrix} = \begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} + \frac{1}{\rho_{f_i}}\mathbf{m}(\theta_{f_i}, \phi_{f_i}), \tag{3.37}$$
$$\mathbf{m} = \begin{bmatrix} \cos\phi_{f_i} \cdot \sin\theta_{f_i} & -\sin\phi_{f_i} & \cos\phi_{f_i} \cdot \cos\theta_{f_i} \end{bmatrix}^{\top},$$

where $\mathbf{y}_{f_i}$ encodes the position 'ray' where the feature was first seen, by the camera's EKF estimated optical centre position x_i y_i z_i or $\mathbf{r}^{wc}$ in Equation (3.33). The azimuth θ_{f_i} and elevation ϕ_{f_i} are calculated using Equation (3.35) according to (Civera et al., 2008)*:

$$\begin{bmatrix} \theta_{f_i} \\ \phi_{f_i} \end{bmatrix} = \begin{bmatrix} \arctan(h_x^w, h_z^w) \\ \arctan(-h_y^w, \sqrt{(h_x^w)^2 + (h_z^w)^2}) \end{bmatrix} \tag{3.38}$$

Finally, depth is estimated by the EKF through its inverse $\rho_{f_i} = 1/d_{f_i}$, with ρ_{f_i} set empirically to a value ρ_0 (Civera et al., 2008). A full representation is illustrated in Figure 3.7.

Feature states $\mathbf{y}_{f_i}$ in Equation (3.36) are added into the state vector after the camera states shown in Equation (3.33), so that it resembles the following full state vector:

$$\mathbf{x}_{k-1} = \begin{bmatrix} \mathbf{r}_{k-1}^{wc} & \mathbf{q}_{k-1}^{wc} & \mathbf{v}_{k-1}^{w} & \omega_{k-1}^{c} & \mathbf{y}_{f_0} & \cdots & \mathbf{y}_{f_i} \end{bmatrix}^{\top} \tag{3.39}$$

*Note that azimuth θ_{f_i} and elevation ϕ_{f_i} are swapped in (Montiel et al., 2008).

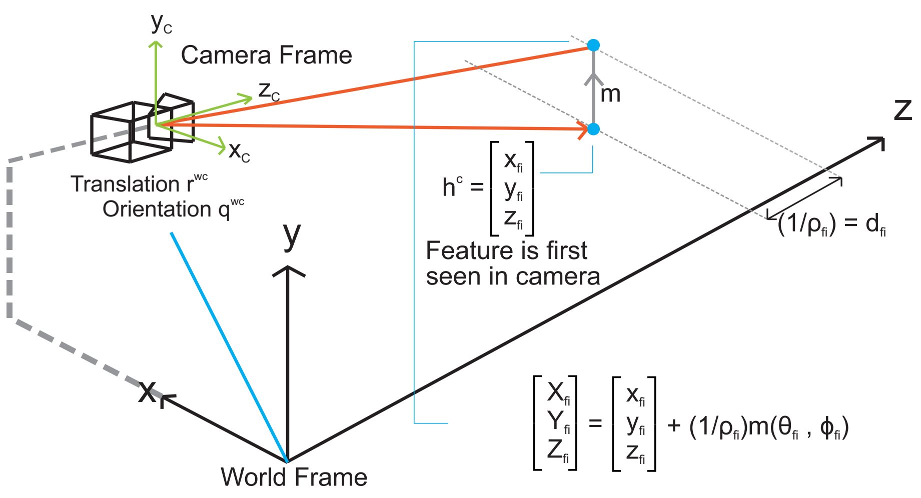

y_c
Camera Frame
z_c
x_c
m
Translation r^{wc}
Orientation q^{wc}
$h^c = \begin{bmatrix} x_{fi} \\ y_{fi} \\ z_{fi} \end{bmatrix}$
Feature is first seen in camera
z
$(1/\rho_{fi}) = d_{fi}$
y
x
World Frame
$\begin{bmatrix} X_{fi} \\ Y_{fi} \\ Z_{fi} \end{bmatrix} = \begin{bmatrix} x_{fi} \\ y_{fi} \\ z_{fi} \end{bmatrix} + (1/\rho_{fi})m(\theta_{fi}, \phi_{fi})$

Figure 3.7: Feature inverse depth parametrisation. A feature is observed for the first time in the camera frame according to Equation (3.32). Later on the EKF provides better estimates of depth through its inverse $\rho_{\mathrm{f_i}} = 1/\mathrm{d_{f_i}}$, changing the magnitude of the directional ray $\mathbf{m}(\theta_{\mathrm{f_i}}, \phi_{\mathrm{f_i}})$. Hence the feature appearance in the image changes, using to the parallax effect and resulting in Equation (3.37).

It is important to remember that this vector is an estimate from either the first or a previous iteration just before the update step in the EKF, hence the subindex $_{k-1}$ in $\mathbf{x}_{k-1}$ which recalls Equations (3.9) and (3.10). Many features can be added as long as computational power allows. Afterwards it is needed to assign uncertainty to the features by adding them into the covariance matrix. This is done as the EKF idea remains a Gaussian approximation (recall Equation (3.3)), therefore requiring both mean and covariance to perform prediction and update respectively.

3.2.4 Adding Features to the Covariance

Once a feature has been added to the state vector of the EKF, Equation (3.39), the covariance matrix $\mathbf{P}_{k-1}$, the measurement covariance matrix $\mathbf{R}_{\mathrm{f_i}}$ and the inverse depth feature uncertainty $\sigma_{\rho_{\mathrm{f_i}}}$ are used to initialise the uncertainties of the new feature. The covariance matrix $\mathbf{R}_{\mathrm{f_i}}$ considers the observation uncertainty when a feature is observed, considering its image projection with coordinates $\mathrm{h_{u_x}}$ and $\mathrm{h_{u_y}}$

(recall Figure 3.6):

$$\mathbf{R}_{\mathrm{f_i}} = \begin{bmatrix} \sigma^2_{\mathrm{h_{ux}}} & 0 \\ 0 & \sigma^2_{\mathrm{h_{uy}}} \end{bmatrix} \tag{3.40}$$

where $\sigma^2_{\mathrm{h_{ux}}}$ and $\sigma^2_{\mathrm{h_{uy}}}$ are the variations in measurements from the image feature. The initial value for the uncertainty in the inverse depth of the feature $\sigma_{\rho_{\mathrm{f_i}}}$ is obtained through experimentation (Civera et al., 2008). Given these considerations the covariance matrix for the EKF after feature initialisation becomes:

$$\mathbf{P}^{\mathrm{new}}_{\mathrm{k-1}} = \mathbf{J} \begin{bmatrix} \mathbf{P}_{\mathrm{k-1}} & 0 & 0 \\ 0 & \mathbf{R}_{\mathrm{f_i}} & 0 \\ 0 & 0 & \sigma^2_{\rho_{\mathrm{f_i}}} \end{bmatrix} \mathbf{J}^{\top},$$

with the Jacobian $\mathbf{J}$ defined as:

$$\mathbf{J} = \left[\begin{array}{ccccc|cc} & & \mathrm{I} & & & 0 & 0 \\ \hline \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{r}^{\mathrm{wc}}} & \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{q}^{\mathrm{wc}}} & 0 & \cdots & 0 & \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{h_d}} & \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \rho_{\mathrm{f_i}}} \end{array}\right]. \tag{3.41}$$

It is possible to obtain the Jacobian $\mathbf{J}$ in Equation (3.41) considering the chain rule. For the first two partial derivatives Equations (3.33) and (3.36) are used:

$$\frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{r}^{\mathrm{wc}}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}; \qquad \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{q}^{\mathrm{wc}}} = \frac{\partial \mathbf{y}_{\mathrm{f_i}}}{\partial \mathbf{h}^{\mathrm{w}}} \cdot \frac{\partial \mathbf{h}^{\mathrm{w}}}{\partial \mathbf{q}^{\mathrm{wc}}}$$

where $\partial \mathbf{y}_{f_i}/\partial \mathbf{h}^w$ has mostly zeros except for the azimuth and elevation:

$$\frac{\partial \mathbf{y}_{f_i}}{\partial \mathbf{h}^w} = \begin{bmatrix} \frac{\partial x_{f_i}}{\partial h^w_x} & \frac{\partial x_{f_i}}{\partial h^w_y} & \frac{\partial x_{f_i}}{\partial h^w_z} \\ \frac{\partial y_{f_i}}{\partial h^w_x} & \frac{\partial y_{f_i}}{\partial h^w_y} & \frac{\partial y_{f_i}}{\partial h^w_z} \\ \frac{\partial z_{f_i}}{\partial h^w_x} & \frac{\partial z_{f_i}}{\partial h^w_y} & \frac{\partial z_{f_i}}{\partial h^w_z} \\ \frac{\partial \theta_{f_i}}{\partial h^w_x} & \frac{\partial \theta_{f_i}}{\partial h^w_y} & \frac{\partial \theta_{f_i}}{\partial h^w_z} \\ \frac{\partial \phi_{f_i}}{\partial h^w_x} & \frac{\partial \phi_{f_i}}{\partial h^w_y} & \frac{\partial \phi_{f_i}}{\partial h^w_z} \\ \frac{\partial \rho_{f_i}}{\partial h^w_x} & \frac{\partial \rho_{f_i}}{\partial h^w_y} & \frac{\partial \rho_{f_i}}{\partial h^w_z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{\partial \theta_{f_i}}{\partial h^w_x} & \frac{\partial \theta_{f_i}}{\partial h^w_y} & \frac{\partial \theta_{f_i}}{\partial h^w_z} \\ \frac{\partial \phi_{f_i}}{\partial h^w_x} & \frac{\partial \phi_{f_i}}{\partial h^w_y} & \frac{\partial \phi_{f_i}}{\partial h^w_z} \\ 0 & 0 & 0 \end{bmatrix} \qquad (3.42)$$

Therefore

$$\frac{\partial \theta}{\partial \mathbf{h}^w} = \begin{bmatrix} \frac{h^w_z}{(h^w_x)^2+(h^w_z)^2} \\ 0 \\ \frac{-h^w_x}{(h^w_x)^2+(h^w_z)^2} \end{bmatrix}^\top ; \quad \frac{\partial \phi}{\partial \mathbf{h}^w} = \begin{bmatrix} \frac{h^w_x h^w_y}{(h^w_x)^2+(h^w_y)^2+(h^w_z)^2 \sqrt{(h^w_x)^2+(h^w_y)^2+(h^w_z)^2}} \\ -\frac{\sqrt{(h^w_x)^2+(h^w_z)^2}}{(h^w_x)^2+(h^w_y)^2+(h^w_z)^2} \\ \frac{h^w_z h^w_y}{(h^w_x)^2+(h^w_y)^2+(h^w_z)^2 \sqrt{(h^w_x)^2+(h^w_y)^2+(h^w_z)^2}} \end{bmatrix}^\top \qquad (3.43)$$

For $\partial \mathbf{h}^w/\partial \mathbf{q}^{wc}$, Equation (3.35) can be used as the rotation matrix $\mathbf{ROT}_{wc}$ in Equation (3.34) depends on $\mathbf{q}^{wc}$, thus the partial derivative $\partial \mathbf{h}^w/\partial \mathbf{q}^{wc}$ would be equal to $\partial \mathbf{h}^w/\partial \mathbf{q}^{wc} = \partial(\mathbf{ROT}_{wc})/\partial \mathbf{q}^{wc} \times \mathbf{h}^c$ from Equation (3.32), where the components of $\partial(\mathbf{ROT}_{wc})/\partial \mathbf{q}^{wc}$ are given by (Diebel, 2006)*:

$$\frac{\partial(\mathbf{ROT}_{wc})}{\partial q_r} = 2 \begin{bmatrix} q_r & q_z & -q_y \\ -q_z & q_r & q_x \\ q_y & -q_x & q_r \end{bmatrix}^\top , \quad \frac{\partial(\mathbf{ROT}_{wc})}{\partial q_x} = 2 \begin{bmatrix} q_x & q_y & q_z \\ q_y & -q_x & q_r \\ q_z & -q_r & -q_x \end{bmatrix}^\top ,$$

$$\frac{\partial(\mathbf{ROT}_{wc})}{\partial q_y} = 2 \begin{bmatrix} -q_y & q_x & -q_r \\ q_x & q_y & q_z \\ q_r & q_z & -q_y \end{bmatrix}^\top , \quad \frac{\partial(\mathbf{ROT}_{wc})}{\partial q_z} = 2 \begin{bmatrix} -q_z & q_r & q_x \\ -q_r & -q_z & q_y \\ q_x & q_y & q_z \end{bmatrix}^\top .$$

*Note that in this citation they are transposed, but that is only because the rotation matrix considered there is from world to camera, not from camera to world.

Thus

$$\frac{\partial \mathbf{h}^w}{\partial \mathbf{q}^{wc}} = \left[\frac{\partial(\mathbf{ROT}_{wc})}{\partial q_r} \cdot \mathbf{h}^c \quad \frac{\partial(\mathbf{ROT}_{wc})}{\partial q_x} \cdot \mathbf{h}^c \quad \frac{\partial(\mathbf{ROT}_{wc})}{\partial q_y} \cdot \mathbf{h}^c \quad \frac{\partial(\mathbf{ROT}_{wc})}{\partial q_z} \cdot \mathbf{h}^c \right]. \tag{3.45}$$

Continuing with $\partial \mathbf{y}_{f_i}/\partial \mathbf{h_d}$ in (3.41) the chain rule is also used:

$$\frac{\partial \mathbf{y}_{f_i}}{\partial \mathbf{h_d}} = \frac{\partial \mathbf{y}_{f_i}}{\partial \mathbf{h}^w} \cdot \frac{\partial \mathbf{h}^w}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h_d}} \tag{3.46}$$

where $\partial \mathbf{y}_{f_i}/\partial \mathbf{h}^w$ is given by Equation (3.42). The relationship between a point $\partial \mathbf{h}^w/\partial \mathbf{h}^c$ in both camera frame and world frame is given by $\mathbf{ROT}_{wc}$. The partial derivative $\partial \mathbf{h}^c/\partial \mathbf{h_u}$ can be extracted from Equation (3.32):

$$\frac{\partial \mathbf{h}^c}{\partial \mathbf{h_u}} = \begin{bmatrix} \frac{1}{f_x} & 0 \\ 0 & \frac{1}{f_y} \\ 0 & 0 \end{bmatrix} \tag{3.47}$$

For $\partial \mathbf{h_u}/\partial \mathbf{h_d}$ Equations (3.24), (3.23), (3.26) are required:

$$\frac{\partial \mathbf{h_u}}{\partial \mathbf{h_d}} = \frac{\partial \mathbf{h_u}}{\partial \mathbf{n_u}} \cdot \frac{\partial \mathbf{n_u}}{\partial \mathbf{n_d}} \cdot \frac{\partial \mathbf{n_d}}{\partial \mathbf{h_d}}$$

which can also be rewritten as:

$$\frac{\partial \mathbf{h_u}}{\partial \mathbf{h_d}} = \frac{\partial \mathbf{h_u}}{\partial \mathbf{n_u}} \cdot \left(\frac{\partial \mathbf{n_d}}{\partial \mathbf{n_u}}\right)^{-1} \cdot \frac{\partial \mathbf{n_d}}{\partial \mathbf{h_d}}$$

Therefore $\partial \mathbf{h_u}/\partial \mathbf{n_u}$ and $\partial \mathbf{n_d}/\partial \mathbf{h_d}$ are given by:

$$\frac{\partial \mathbf{h_u}}{\partial \mathbf{n_u}} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}; \qquad \frac{\partial \mathbf{n_d}}{\partial \mathbf{h_d}} = \begin{bmatrix} \frac{1}{f_x} & 0 \\ 0 & \frac{1}{f_y} \end{bmatrix} \tag{3.48}$$

and for $\partial \mathbf{n_d}/\partial \mathbf{n_u}$:

$$\frac{\partial \mathbf{n_d}}{\partial \mathbf{n_u}} = \begin{bmatrix} \frac{\partial n_{d_x}}{\partial n_{u_x}} & \frac{\partial n_{d_x}}{\partial n_{u_y}} \\ \frac{\partial n_{d_y}}{\partial n_{u_x}} & \frac{\partial n_{d_y}}{\partial n_{u_y}} \end{bmatrix} \tag{3.49}$$

with*

$$\frac{\partial n_{d_x}}{\partial n_{u_x}} = 1 + 3K_1 n_{u_x}^2 + K_1 n_{u_y}^2 + 5K_2 n_{u_x}^4 + 6K_2 n_{u_x}^2 n_{u_y}^2 + K_2 n_{u_y}^4$$
$$\frac{\partial n_{d_x}}{\partial n_{u_y}} = 2K_1 n_{u_x} n_{u_y} + 4K_2 n_{u_x}^3 n_{u_y} + 4K_2 n_{u_x} n_{u_y}^3$$
$$\frac{\partial n_{d_y}}{\partial n_{u_x}} = 2K_1 n_{u_x} n_{u_y} + 4K_2 n_{u_x}^3 n_{u_y} + 4K_2 n_{u_x} n_{u_y}^3$$
$$\frac{\partial n_{d_y}}{\partial n_{u_y}} = 1 + 3K_1 n_{u_y}^2 + K_1 n_{u_x}^2 + 5K_2 n_{u_y}^4 + 6K_2 n_{u_x}^2 n_{u_y}^2 + K_2 n_{u_x}^4$$

Finally $\partial \mathbf{y}_{f_i}/\partial \rho_{f_i}$ is simply 1. All these partial derivatives can be set in Equation (3.41), allowing to form the Jacobian now includes the new feature. This will eventually multiply the covariance matrix $\hat{\mathbf{P}}_{k-1}$, introducing the feature as well. The next section pertains to prediction step, which now can be performed given that a new feature has been included in both the EKF state and covariance matrix. This prediction expects an observation within uncertainty limits, in which the difference between these values is used for the update step in the EKF.

3.3 Inverse Depth Monocular SLAM: Prediction and Update

After a state model with registered landmarks has been created like in Equation (3.39), with the latter added to the covariance as described in Section 3.2.4, it is possible to predict both state and covariance as in Equations (3.11) and (3.12). This eventually allows for EKF update through feature re-observation.

The state model only allows to predict camera motion and is considered affected with non-additive noise, as this accounts for noise present within the model as described in Section 3.1.2. A newly predicted camera position will change the appearance of the registered landmarks $\mathbf{h}(\mathbf{m}_k^-, \mathbf{0})$ using an observation model. In case of not having registered features in the EKF, the camera position uncertainty will simply increase with each time step k.

Using all this information the Kalman update can be performed, which occurs

*Civera's code uses a different Jacobian for $\partial \mathbf{n_d}/\partial \mathbf{n_u}$.

after the error from the predicted landmarks and the observations is obtained, Equation (3.15). The observed features are not perfect as the observation model is also contaminated by non-additive noise, thus affecting landmark observations and subsequently the innovation covariance matrix **S** in Equation (3.16).

The innovation covariance matrix contains the uncertainty in the measurements: if it has big values the confidence over the observations will be low, if it has lower values the opposite case applies. Both the innovation and covariance matrices provide weighting in the form of the Kalman gain **K** in Equation (3.17). This gain indicates in which proportion to trust both the model and observations. If it is low the camera motion model prediction will be trusted more, if it is high the observations will be given increased importance. As a result the mean, i.e. camera and landmarks position estimates and covariance matrix are updated with Equations (3.20) and (3.21).

As such, the next sections focus on the aspects involved for the prediction and update steps: Predicting three dimensional landmark positions, adding lens imperfections in their bi-dimensional projections, matching new features with predictions, and finally state and covariance update. These steps are summarised in the simplified monocular representation seen in Figure 3.8.

3.3.1 Predict Camera State and Covariance

Once new features have been added to both the state and covariance of the EKF, it is possible to perform camera and feature appearance prediction. The camera state evolution follows the constant velocity model. This is used as it gives a simplified camera motion taking into account acceleration impulses:

$$\mathbf{x}_{v_k}^- = \begin{bmatrix} \mathbf{r}_k^{wc^-} \\ \mathbf{q}_k^{wc^-} \\ \mathbf{v}_k^{w^-} \\ \omega_k^{c^-} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{k-1}^{wc} + (\mathbf{v}_{k-1}^{w} + \mathbf{V}^{w})\Delta t \\ \mathbf{q}((\omega_{k-1}^{c} + \Omega^{c})\Delta t) \times \mathbf{q}_{k-1}^{wc} \\ \mathbf{v}_{k-1}^{w} + \mathbf{V}^{w} \\ \omega_{k-1}^{c} + \Omega^{c} \end{bmatrix} \tag{3.50}$$

where $\mathbf{r}^{wc}$, $\mathbf{q}^{wc}$, $\mathbf{v}^{w}$ and ω^{c} share the same definitions as in Equation (3.33); $\mathbf{q}((\omega_{k-1}^{c} + \Omega^{c})\Delta t)$ is a quaternion[*] defined by $(\omega_{k-1}^{c} + \Omega^{c})\Delta t$. Finally $\mathbf{V}^{w}$ are the lin-

[*]Quaternion multiplication is not commutative, the importance of this will be seen later.

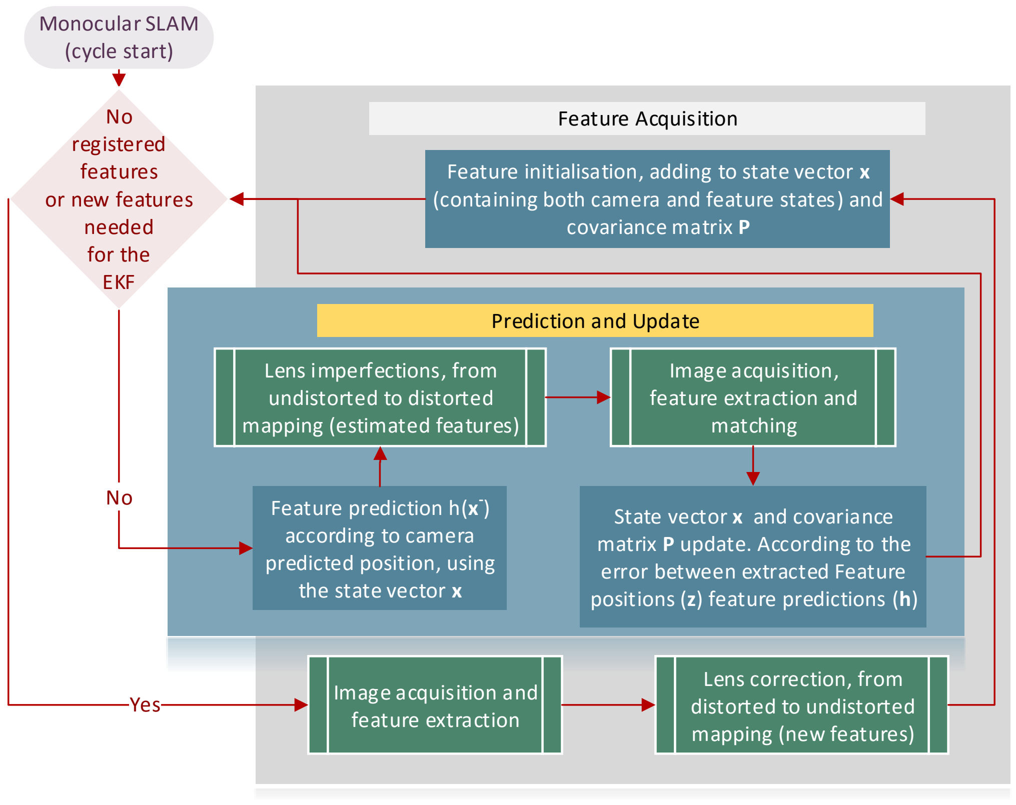

Monocular SLAM (cycle start)
No registered features or new features needed for the EKF
Feature Acquisition
Feature initialisation, adding to state vector **x** (containing both camera and feature states) and covariance matrix **P**
Prediction and Update
Lens imperfections, from undistorted to distorted mapping (estimated features)
Image acquisition, feature extraction and matching
Feature prediction h(**x**ˆ) according to camera predicted position, using the state vector **x**
State vector **x** and covariance matrix **P** update. According to the error between extracted Feature positions (**z**) feature predictions (**h**)
No
Yes
Image acquisition and feature extraction
Lens correction, from distorted to undistorted mapping (new features)

Figure 3.8: Monocular SLAM simplified diagram with remarked prediction and update group. Blue rectangles signal EKF steps while green ones represent stages inherent to vision-SLAM. The prediction and update group shows how stored features used, allowing their state and covariance prediction using current camera position. Adding lens imperfections is needed for feature observation, as their predicted 3D positions do not involve those deformations. Update is done in the EKF after observations have been obtained.

ear velocity impulses $\mathbf{a}^w\Delta t$, produced by linear accelerations $\mathbf{a}^w$ w.r.t. the world frame and Ω^c are the angular velocity impulses $\alpha^c\Delta t$ produced by angular accelerations α^c w.r.t. the camera frame. Both linear and angular velocity impulses are assumed to be zero mean and Gaussian, which can be seen as uncertainty affecting the camera whilst in movement. However, for this model the non-additive noise formulation is considered for which the state evolution resembles that of Equation (3.11):

$$\mathbf{x}_{v_k}^- = \mathbf{f}(\mathbf{x}_{v_{k-1}}, \mathbf{0}) = \begin{bmatrix} \mathbf{r}_k^{wc^-} \\ \mathbf{q}_k^{wc^-} \\ \mathbf{v}_k^{w^-} \\ \omega_k^{c^-} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{k-1}^{wc} + \mathbf{v}_{k-1}^{w}\Delta t \\ \mathbf{q}(\omega_{k-1}^{c}\Delta t) \times \mathbf{q}_{k-1}^{wc} \\ \mathbf{v}_{k-1}^{w} \\ \omega_{k-1}^{c} \end{bmatrix} \tag{3.51}$$

The process noise and its covariance are given by:

$$\mathbf{q}_{k-1} = \begin{bmatrix} \mathbf{N}(0, (\mathbf{a}^w\Delta t)^2) \\ \mathbf{N}(0, (\alpha^c\Delta t)^2) \end{bmatrix}; \qquad \mathbf{Q}_{k-1} = \begin{bmatrix} (\mathbf{a}^w\Delta t)^2 & 0 \\ 0 & (\alpha^c\Delta t)^2 \end{bmatrix} \tag{3.52}$$

Then the Jacobian of the process with respect to the noise states is given by Equation (3.14) as:

$$\mathbf{F}_\mathbf{q} = \frac{\partial \mathbf{x}_{v_k}^-}{\partial \mathbf{q}_{k-1}} = \left.\frac{\partial f_i(\mathbf{x}, \mathbf{q})}{\partial q_j}\right|_{\mathbf{x}=\mathbf{x}_{v_{k-1}}, \mathbf{q}=\mathbf{0}} = \begin{bmatrix} \frac{\partial \mathbf{r}_k^{wc^-}}{\partial \mathbf{v}_{k-1}^w} & \frac{\partial \mathbf{r}_k^{wc^-}}{\partial \omega_{k-1}^c} \\ \frac{\partial \mathbf{q}_k^{wc^-}}{\partial \mathbf{v}_{k-1}^w} & \frac{\partial \mathbf{q}_k^{wc^-}}{\partial \omega_{k-1}^c} \\ \frac{\partial \mathbf{v}_k^{w^-}}{\partial \mathbf{v}_{k-1}^w} & \frac{\partial \mathbf{v}_k^{w^-}}{\partial \omega_{k-1}^c} \\ \frac{\partial \omega_k^{c^-}}{\partial \mathbf{v}_{k-1}^w} & \frac{\partial \omega_k^{c^-}}{\partial \omega_{k-1}^c} \end{bmatrix} \tag{3.53}$$

where

$$\frac{\partial \mathbf{v}_k^{w^-}}{\partial \mathbf{v}_{k-1}^w} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \frac{\partial \omega_k^{c^-}}{\partial \omega_{k-1}^c} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \frac{\partial \mathbf{r}_k^{wc^-}}{\partial \mathbf{v}_{k-1}^w} = \begin{bmatrix} \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix} \tag{3.54}$$

For the partial derivative $\partial \mathbf{q}_k^{wc^-} / \partial \omega_{k-1}^c$ the following quaternion multiplication formulae are required (Diebel, 2006):

$$\begin{aligned} \mathbf{qp} &= \mathbf{q}_m(\mathbf{q}, \mathbf{p}) = Q(\mathbf{q})\mathbf{p} = \bar{Q}(\mathbf{p})\mathbf{q} \\ \mathbf{pq} &= \mathbf{q}_m(\mathbf{p}, \mathbf{q}) = Q(\mathbf{p})\mathbf{q} = \bar{Q}(\mathbf{q})\mathbf{p} \end{aligned} \tag{3.55}$$

where $\mathbf{q}$ and $\mathbf{p}$ are two quaternions, with Q being a matrix function and $\bar{Q}$ its conjugate whose values depend on a quaternion as well:

$$Q(\mathbf{q}) = \begin{bmatrix} q_r & -q_x & -q_y & -q_z \\ q_x & q_r & q_z & -q_y \\ q_y & -q_z & q_r & q_x \\ q_z & q_y & -q_x & q_r \end{bmatrix}; \qquad \bar{Q}(\mathbf{q}) = \begin{bmatrix} q_r & -q_x & -q_y & -q_z \\ q_x & q_r & -q_z & q_y \\ q_y & q_z & q_r & -q_x \\ q_z & -q_y & q_x & q_r \end{bmatrix}$$

Given this, the partial derivatives of a quaternion multiplication can be defined as (Diebel, 2006):

$$\frac{\partial \mathbf{q}_m(\mathbf{q}, \mathbf{p})}{\partial \mathbf{q}} = \bar{Q}(\mathbf{p}) \tag{3.56a}$$

$$\frac{\partial \mathbf{q}_m(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}} = Q(\mathbf{q}) \tag{3.56b}$$

Following the state update in Equation (3.51), it is possible to observe that $\mathbf{q}(\omega_{k-1}^c \Delta t) \times \mathbf{q}_{k-1}^{wc}$ is a quaternion multiplication and it can also be written as $\mathbf{q}_{k-1}^{wc} \times \mathbf{q}(\omega_{k-1}^c \Delta t)$*. For the former case Equation (3.56a) becomes $\bar{Q}(\mathbf{q}_{k-1}^{wc})$ and for the latter is $\bar{Q}(\mathbf{q}(\omega_{k-1}^c \Delta t))$†. Thus in order to solve $\partial \mathbf{q}_k^{wc^-} / \partial \omega_{k-1}^c$ the chain rule and Equation (3.56a) can be used:

$$\frac{\partial \mathbf{q}_k^{wc^-}}{\partial \omega_{k-1}^c} = \frac{\partial (\mathbf{q}(\omega_{k-1}^c \Delta t) \times \mathbf{q}_{k-1}^{wc})}{\partial \mathbf{q}(\omega_{k-1}^c \Delta t)} \cdot \frac{\partial \mathbf{q}(\omega_{k-1}^c \Delta t)}{\partial \omega_{k-1}^c} = \bar{Q}(\mathbf{q}_{k-1}^{wc}) \frac{\partial \mathbf{q}(\omega_{k-1}^c \Delta t)}{\partial \omega_{k-1}^c} \tag{3.57}$$

with $\mathbf{q}(\omega_{k-1}^c \Delta t)$ the quaternion obtained from the angular velocity times Δt,

*Quaternion multiplication is not commutative. However, any order can be used as long as it is maintained through all the state evolution steps. Note that this requires using the proper equations for the needed partial derivatives.

†Civera's code uses $\mathbf{q}_k^{wc} \times \mathbf{q}(\omega_k^c \Delta t)$, but Equation (3.56a) is used for $\bar{Q}(\mathbf{q}_k^{wc})$.

where:

$$\mathbf{q} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) \times \left(\frac{\omega^c_{x_{k-1}}\Delta t}{\theta}\right) \\ \sin\left(\frac{\theta}{2}\right) \times \left(\frac{\omega^c_{y_{k-1}}\Delta t}{\theta}\right) \\ \sin\left(\frac{\theta}{2}\right) \times \left(\frac{\omega^c_{z_{k-1}}\Delta t}{\theta}\right) \end{bmatrix}; \quad \theta = \sqrt{(\omega^c_{x_{k-1}}\Delta t)^2 + (\omega^c_{y_{k-1}}\Delta t)^2 + (\omega^c_{z_{k-1}}\Delta t)^2}$$

The Jacobian of the next state with respect to the current state or transition matrix is obtained, as seen in Equation (3.13):

$$\mathbf{F}_k = \frac{\partial \mathbf{x}^-_{v_k}}{\partial \mathbf{x}_{v_{k-1}}} = \left.\frac{\partial f_i(\mathbf{x},\mathbf{q})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_{v_{k-1}},\mathbf{q}=\mathbf{0}} = \begin{bmatrix} \frac{\partial \mathbf{r}^{wc^-}_k}{\partial \mathbf{r}^{wc}_{k-1}} & \frac{\partial \mathbf{r}^{wc^-}_k}{\partial \mathbf{q}^{wc}_{k-1}} & \frac{\partial \mathbf{r}^{wc^-}_k}{\partial \mathbf{v}^{w}_{k-1}} & \frac{\partial \mathbf{r}^{wc^-}_k}{\partial \omega^{c}_{k-1}} \\ \frac{\partial \mathbf{q}^{wc^-}_k}{\partial \mathbf{r}^{wc}_{k-1}} & \frac{\partial \mathbf{q}^{wc^-}_k}{\partial \mathbf{q}^{wc}_{k-1}} & \frac{\partial \mathbf{q}^{wc^-}_k}{\partial \mathbf{v}^{w}_{k-1}} & \frac{\partial \mathbf{q}^{wc^-}_k}{\partial \omega^{c}_{k-1}} \\ \frac{\partial \mathbf{v}^{w^-}_k}{\partial \mathbf{r}^{wc}_{k-1}} & \frac{\partial \mathbf{v}^{w^-}_k}{\partial \mathbf{q}^{wc}_{k-1}} & \frac{\partial \mathbf{v}^{w^-}_k}{\partial \mathbf{v}^{w}_{k-1}} & \frac{\partial \mathbf{v}^{w^-}_k}{\partial \omega^{c}_{k-1}} \\ \frac{\partial \omega^{c^-}_k}{\partial \mathbf{r}^{wc}_{k-1}} & \frac{\partial \omega^{c^-}_k}{\partial \mathbf{q}^{wc}_{k-1}} & \frac{\partial \omega^{c^-}_k}{\partial \mathbf{v}^{w}_{k-1}} & \frac{\partial \omega^{c^-}_k}{\partial \omega^{c}_{k-1}} \end{bmatrix} \tag{3.58}$$

in which the terms are given by Equations (3.54), coupled with:

$$\frac{\partial \mathbf{r}^{wc^-}_k}{\partial \mathbf{r}^{wc}_{k-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

However, Equation (3.56b) is used for $\partial \mathbf{q}^{wc^-}_k / \partial \mathbf{q}^{wc}_{k-1}$:

$$\frac{\partial \mathbf{q}^{wc^-}_k}{\partial \mathbf{q}^{wc}_{k-1}} = \frac{\partial(\mathbf{q}(\omega^c_{k-1}\Delta t) \times \mathbf{q}^{wc}_{k-1})}{\partial \mathbf{q}^{wc}_{k-1}} = Q(\mathbf{q}(\omega^c_{k-1}\Delta t)) \tag{3.59}$$

Finally the camera covariance $\mathbf{P}_{v_{k-1}}$ is predicted in a similar way as Equation (3.12) using Equations[*] (3.52), (3.53), and (3.58):

$$\mathbf{P}^{-}_{\mathrm{v}_k} = \mathbf{F}_k \mathbf{P}_{\mathrm{v}_{k-1}} \mathbf{F}_k^{\top} + \mathbf{L}_k \mathbf{Q}_k \mathbf{L}_k^{\top} \tag{3.60}$$

Note that there is no state evolution for the features, as they are supposed to remain in the same place where they were first observed. Therefore the entire covariance matrix $\mathbf{P}^{-}_{k}$ is updated as follows:

$$\left[\mathbf{P}^{-}_{k}\right]_{m,n} = \begin{bmatrix} \mathbf{P}^{-}_{\mathrm{v}_k} & \mathbf{F}_k \left[\mathbf{P}_{k-1}\right]_{0\cdots 12, 13\cdots n} \\ \left[\mathbf{P}_{k-1}\right]_{13\cdots m, 0\cdots 12} \mathbf{F}_k^{\top} & \left[\mathbf{P}_{k-1}\right]_{13\cdots m, 13\cdots n} \end{bmatrix} \tag{3.61}$$

This resulting covariance reflects the dynamics of feature uncertainty after the camera covariance prediction. Hence, the camera uncertainty $\mathbf{P}^{-}_{\mathrm{v}_k}$ remains the same as obtained in Equation (3.60). The uncertainty between the features $[\mathbf{P}_{k-1}]_{13\cdots m,13\cdots n}$ also remains unaltered, as they are not influenced by camera movements but only by themselves. However, $\mathbf{F}_k\,[\mathbf{P}_{k-1}]_{0\cdots 12,13\cdots n}$ and $[\mathbf{P}_{k-1}]_{13\cdots m,0\cdots 12}\,\mathbf{F}_k^{\top}$ show modification influenced by a predicted change in camera pose, as these represent the uncertainty between the camera and features.

3.3.2 Predict Camera Measurements

Given the camera state prediction $\mathbf{x}^{-}_{\mathrm{v}_k}$ in Equation (3.51), the update step of the EKF requires to obtain observations $\mathbf{h}(\mathbf{x}^{-}_{\mathrm{v}_k}, \mathbf{0})$ as required in Equation (3.15). For an observation with inverse depth parametrisation (recall Figure 3.7), the formula is (Civera et al., 2008):

$$\mathbf{h}^{c} = \mathbf{h}^{c}_{\rho} = \mathbf{ROT}_{cw} \left(\rho_{f_i} \left(\begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} - \mathbf{r}^{wc} \right) + \mathbf{m}(\theta_{f_i}, \phi_{f_i}) \right) \tag{3.62}$$

where $\mathbf{ROT}_{cw}$ is the rotation matrix that brings a point from world frame to camera frame. This is similar to Equation (3.34) but transposed[†], $\mathbf{m}(\theta_{f_i}, \phi_{f_i})$ is given

[*]Where $\mathbf{Q}$ for practical purposes remains only with the subindex k, i.e. $\mathbf{Q}_k$.

[†]This is an orthogonal matrix and it has the property that both its inverse and transpose are the same. In Civera's code both transpose and inverse are used indistinctly. However, it is worth mentioning that the inverse of a matrix presents a higher computational load than its transpose.

by:

$$\mathbf{m}(\theta_{f_i}, \phi_{f_i}) = \begin{bmatrix} \cos(\phi_{f_i})\sin(\theta_{f_i}) \\ -\sin(\phi_{f_i}) \\ \cos(\phi_{f_i})\cos(\theta_{f_i}) \end{bmatrix} \tag{3.63}$$

with the feature states x_{f_i}, y_{f_i}, z_{f_i}, θ_{f_i}, ϕ_{f_i} and ρ_{f_i} obtained from Equation (3.36). Because $\mathbf{h}^c$ represents the feature in camera frame, it is necessary to map it into a 2D prediction, as an image only has 2 dimensions:

$$\mathbf{h_u} = \begin{bmatrix} h_{u_x} \\ h_{u_y} \end{bmatrix} = \begin{bmatrix} f_x \dfrac{h^c_x}{h^c_z} + X_c \\ f_y \dfrac{h^c_y}{h^c_z} + Y_c \end{bmatrix} \tag{3.64}$$

Note the addition of the principal points X_c and Y_c. As opposed to Equation (3.32), these are added as an image has positive coordinates.

After Equation (3.64) has been obtained it is necessary to follow the steps to go from an undistorted projection $\mathbf{h_u}$ to a distorted projection $\mathbf{h_d}$ as described in Section 3.2.2. Note that once the pixel coordinates X_{cam} and Y_{cam} are obtained, a check must be performed in order to assert they are within the camera picture. If they are not, the features are simply marked as not possible to be observed and therefore skipped from the EKF update. The following step considers building the innovation covariance matrix for this prediction based on both the predicted camera covariance, the current feature uncertainty and the noise present in observations.

3.3.3 Feature Innovation Covariance Matrix

Once the features have been predicted it is necessary to obtain their current uncertainty, based on that obtained from the covariance prediction step. However, the uncertainty inherent in feature observation needs to be accounted for too. This is because noise is assumed to affect its extraction from the image. This can be done through the innovation covariance matrix for each feature independently*,

*This allows to use a method seen further on to check whether or not an extracted feature belongs to the same sample.

which is akin to Equation (3.16):

$$\mathbf{S}_{k_{f_i}} = \mathbf{H}_{x_{f_i}} \mathbf{P}_k^{-} \mathbf{H}_{x_{f_i}}^{\top} + \mathbf{R}_{k_{f_i}} \tag{3.65}$$

Note that compared to Equation (3.16) there is no Jacobian $\mathbf{H}_{r_{f_i}}$ since there is no model for the observations, i.e. coordinates are acquired from an image 'as is'. Hence, this becomes similar to an additive noise formulation. In this case the measurement covariance matrix $\mathbf{R}_{k_{f_i}}$ resembles that of Equation (3.40), with $\mathbf{P}_k^{-}$ given by Equation (3.61) and $\mathbf{H}_{x_{f_i}}$ defined by:

$$\mathbf{H}_{x_{f_i}} = \begin{bmatrix} \frac{\partial \mathbf{h_d}}{\partial \mathbf{x}_v} & 0 & \cdots & 0 & \frac{\partial \mathbf{h_d}}{\partial \mathbf{y}_{f_i}} \end{bmatrix} \tag{3.66}$$

where $0 \cdots 0$ represents all the other features besides the one currently chosen, in order to obtain its innovation covariance matrix. Given this, $\partial \mathbf{h_d}/\partial \mathbf{x}_v$ can be written as:

$$\begin{aligned}
\frac{\partial \mathbf{h_d}}{\partial \mathbf{x}_v} &= \begin{bmatrix} \frac{\partial \mathbf{h_d}}{\partial \mathbf{r}^{wc}} & \frac{\partial \mathbf{h_d}}{\partial \mathbf{q}^{wc}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial \mathbf{h_d}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{r}^{wc}} & \frac{\partial \mathbf{h_d}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{q}^{wc}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial \mathbf{h_d}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{r}^{wc}} & \frac{\partial \mathbf{h_d}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{q}^{wc}} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial \mathbf{h_d}}{\partial \mathbf{n_d}} \cdot \frac{\partial \mathbf{n_d}}{\partial \mathbf{n_u}} \cdot \frac{\partial \mathbf{n_u}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{r}^{wc}} & \frac{\partial \mathbf{h_d}}{\partial \mathbf{n_d}} \cdot \frac{\partial \mathbf{n_d}}{\partial \mathbf{n_u}} \cdot \frac{\partial \mathbf{n_u}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^c} \cdot \frac{\partial \mathbf{h}^c}{\partial \mathbf{q}^{wc}} \end{bmatrix}
\end{aligned}$$

where $\partial \mathbf{n_d}/\partial \mathbf{n_u}$ is given by Equation (3.49), later on $\partial \mathbf{h_d}/\partial \mathbf{n_d}$ and $\partial \mathbf{n_u}/\partial \mathbf{h_u}$ are given by:

$$\frac{\partial \mathbf{h_d}}{\partial \mathbf{n_d}} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix}; \qquad \frac{\partial \mathbf{n_u}}{\partial \mathbf{h_u}} = \begin{bmatrix} \frac{1}{f_x} & 0 \\ 0 & \frac{1}{f_y} \end{bmatrix} \tag{3.67}$$

Next $\partial \mathbf{h_u}/\partial \mathbf{h}^c$ can be obtained from Equation (3.64) using first Equation (3.62) to find $\mathbf{h}^c$, thus yielding:

$$\frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^c} = \begin{bmatrix} \frac{f_x}{h_z} & 0 \\ 0 & \frac{f_y}{h_z} \\ -f_x \frac{h_x}{h_z^2} & -f_y \frac{h_y}{h_z^2} \end{bmatrix}$$

The next partial derivative $\partial\mathbf{h}^{c}/\partial\mathbf{r}^{wc}$ is then obtained from Equation (3.62) as:

$$\frac{\partial\mathbf{h}^{c}}{\partial\mathbf{r}^{wc}} = -\mathbf{ROT}_{cw}(\rho_{i})$$

Lastly the partial derivative $\partial\mathbf{h}^{c}/\partial\mathbf{q}^{wc}$ follows a similar approach to Equation (3.45), however, insight now comes from Equation (3.62), where

$$\mathbf{h}^{c} = \mathbf{ROT}_{cw}\mathbf{h}^{w} \tag{3.68}$$

with $\mathbf{h}^{w}$ as:

$$\mathbf{h}^{w} = \rho_{f_i}\left(\begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} - \mathbf{r}^{wc}\right) + \mathbf{m}(\theta_{f_i}, \phi_{f_i}) \tag{3.69}$$

Hence the partial derivatives of the rotational matrix $\mathbf{ROT}_{cw}$ with respect to the quaternion $\mathbf{q}^{wc}$ are given by (Diebel, 2006):

$$\frac{\partial(\mathbf{ROT}_{cw})}{\partial q_r} = 2\begin{bmatrix} q_r & q_z & -q_y \\ -q_z & q_r & q_x \\ q_y & -q_x & q_r \end{bmatrix}; \qquad \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_x} = 2\begin{bmatrix} q_x & q_y & q_z \\ q_y & -q_x & q_r \\ q_z & -q_r & -q_x \end{bmatrix}$$

$$\frac{\partial(\mathbf{ROT}_{cw})}{\partial q_y} = 2\begin{bmatrix} -q_y & q_x & -q_r \\ q_x & q_y & q_z \\ q_r & q_z & -q_y \end{bmatrix}; \qquad \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_z} = 2\begin{bmatrix} -q_z & q_r & q_x \\ -q_r & -q_z & q_y \\ q_x & q_y & q_z \end{bmatrix}$$

which leads to the partial derivative of interest*:

$$\frac{\partial\mathbf{h}^{c}}{\partial\mathbf{q}^{wc}} = \begin{bmatrix} \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_r}\cdot\mathbf{h}^{w} & \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_x}\cdot\mathbf{h}^{w} & \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_y}\cdot\mathbf{h}^{w} & \frac{\partial(\mathbf{ROT}_{cw})}{\partial q_z}\cdot\mathbf{h}^{w} \end{bmatrix}$$

*Civera's approach to this was similar to Equation (3.45), however instead of using the partial derivatives from Equation (3.70), the ones from Equation (3.44) are used but with a conjugate quaternion, i.e. $(\partial(\mathbf{ROT}_{wc})/\partial\bar{\mathbf{q}}^{wc})\cdot\mathbf{h}^{w}\cdot(\partial\bar{\mathbf{q}}^{wc}/\partial\mathbf{q}^{wc})$.

The partial derivative $\partial \mathbf{h_d} / \partial \mathbf{y}_{\mathrm{f_i}}$ can be obtained in the following way:

$$\begin{aligned}\frac{\partial \mathbf{h_d}}{\partial \mathbf{y}_{\mathrm{f_i}}} &= \left[\frac{\partial \mathbf{h_d}}{\partial \mathbf{h}^{\mathrm{c}}} \cdot \frac{\partial \mathbf{h}^{\mathrm{c}}}{\partial \mathbf{y}_{\mathrm{f_i}}}\right] \\ &= \left[\frac{\partial \mathbf{h_d}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^{\mathrm{c}}} \cdot \frac{\partial \mathbf{h}^{\mathrm{c}}}{\partial \mathbf{y}_{\mathrm{f_i}}}\right] \\ &= \left[\frac{\partial \mathbf{h_d}}{\partial \mathbf{n_d}} \cdot \frac{\partial \mathbf{n_d}}{\partial \mathbf{n_u}} \cdot \frac{\partial \mathbf{n_u}}{\partial \mathbf{h_u}} \cdot \frac{\partial \mathbf{h_u}}{\partial \mathbf{h}^{\mathrm{c}}} \cdot \frac{\partial \mathbf{h}^{\mathrm{c}}}{\partial \mathbf{y}_{\mathrm{f_i}}}\right]\end{aligned}$$

where all the partial derivatives are known except for $\partial \mathbf{h}^{\mathrm{c}} / \partial \mathbf{y}_{\mathrm{f_i}}$, this partial derivative can be obtained directly from Equation (3.62). Once $\mathbf{S}_{\mathrm{k_{f_i}}}$ has been obtained it can be used to match a feature in a newly extracted image. If a salient feature is well outside of this uncertainty it simply does not belong to the same sample and therefore the reading for this feature is discarded.

3.3.4 Obtain Observations

After the innovation covariance matrix $\mathbf{S}_{\mathrm{k_{f_i}}}$ in Equation (3.65) has been obtained for each of the features contained in the state vector from Equation (3.39), the next step is to obtain its corresponding observations $\mathbf{h}^{\mathrm{c}}$. This is no different than the process explained in Section 3.2.2 although matching must be performed. This is to ensure that the observation corresponds to the same landmark which was initialised from the same salient feature.

When using feature detectors a common approach is to obtain *descriptors* from the salient features, which work as identifiers that allow to perform brute-force matching. However, the innovation covariance matrix $\mathbf{S}_{\mathrm{k_{f_i}}}$ together with the Mahalanobis distance (Winter, 2010) can also be used to detect if a newly extracted salient feature is part of the same sample:

$$\text{Mahalanobis Distance} = \sqrt{\mathbf{v}_{\mathrm{k}}^{\mathrm{T}} \mathbf{S}_{\mathrm{k_{f_i}}}^{-1} \mathbf{v}_{\mathrm{k}}}$$

where $\mathbf{v}_{\mathrm{k}}$ is the error between the estimate $\mathbf{h}^{\mathrm{c}}$ given by $\mathbf{h}(\mathbf{m}_{\mathrm{k}}^{-}, \mathbf{0})$ in Equation (3.62) and the observation $\mathbf{z}_{\mathrm{f_i}}$ as in Equation (3.15). For the latter this is given as the x and y coordinates of the newly obtained salient feature. If the Mahalanobis distance fails to pass a certain threshold for this feature, this is skipped in the subsequent update of the EKF algorithm.

3.3.5 State and Covariance Update

The last part of the algorithm before a new cycle starts is to update both state $\mathbf{x}_k$ and covariance $\mathbf{P}_k$, given all the predictions $\mathbf{h}$ from Section 3.3.2 and the feature measurements $\mathbf{z}$ obtained in Section 3.3.4. This is akin to using Equations (3.15), (3.17), (3.20) and (3.21):

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^\top \mathbf{S}_k^{-1} \tag{3.71}$$

$$\mathbf{x}_k = \mathbf{x}_k^- + \mathbf{K}_k(\mathbf{z} - \mathbf{h}) \tag{3.72}$$

$$\mathbf{P}_k = \mathbf{P}_\mathbf{k}^- - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top \tag{3.73}$$

where $\mathbf{P}_k^-$ is given by Equation (3.61), $\mathbf{x}_k^-$ is the predicted state which consists of the camera states from Equation (3.51) and the already stored features in Equation (3.39), as they are assumed to remain static:

$$\mathbf{x}_k^- = \begin{bmatrix} \mathbf{r}_k^{wc^-} & \mathbf{q}_k^{wc^-} & \mathbf{v}_k^{w^-} & \omega_k^{c^-} & \mathbf{y}_{f_0} & \cdots & \mathbf{y}_{f_i} \end{bmatrix}^\top \tag{3.74}$$

$\mathbf{S}_k$ are the stacked matrices obtained from each feature in Equation (3.65), $\mathbf{H}_k$ are also stacked matrices from Equation (3.66). The stacking is done in order to include all features for the update. It is worth mentioning that because the camera orientation is kept as a quaternion, its magnitude must be always 1, therefore normalisation after each state update is recommended.

With this the whole algorithm of monocular SLAM with inverse depth parametrisation has been detailed. Next, results are seen for a C++ implementation of SLAM, using both EKF and inverse depth parametrisation.

3.4 Implementation Results

This section explores the feasibility of the SLAM algorithm, using both EKF and inverse depth parametrisation (Montiel et al., 2008). The algorithm was set to run on a computer with an Intel 4790K processor, aided by a GPU R9 290X from AMD and the Logitech C920 camera. The processor has the task of detecting features using the AKAZE feature detector (F. Alcantarilla et al., 2013), which grabs a total of 15 features randomly selected across the image. The GPU is used to accelerate image processing tasks like greyscale conversion. Each EKF cycle is constrained to be of 0.08 seconds.

The implementation works in real-time, meaning that a live or a pre-recorded video can be used as source instead of frame by frame processing. It is worth mentioning that this does not consider meaningful units for mapping, as the main interest is to ensure stability within the implementation. Therefore, the EKF algorithm works using its own units. The results of SLAM using inverse depth parametrisation are shown next.

3.4.1 Baseline EKF SLAM With Inverse Depth Parametrisation

In order to evaluate the performance of SLAM with inverse depth parametrisation it was decided to use a source video recorded hand-held. However, when hand-held video was used stability issues arose in many runs after runtime reached the minute mark or less. Therefore, it was chosen to perform the experiments using a tripod and performing camera swinging left and right motions, for about 90 degrees as smoothly as possible. This was performed outdoors within an urban area, as this allows to have enough unique, close and far features for good matching.

Detected features are added into both state vector and covariance matrix, then later they are mapped showing elongated ellipses since their initial depth uncertainty is big (Figures 3.9, 3.10 and 3.11). Note that the generated map contains numbers that are not representative of any unit, as the algorithm works with its own internal units with the intention of observing algorithm stability. The map shows from a bottom view perspective what the camera sees at the front, with features remaining for the most part static. However, the ellipse shape representation will shrink down to a point when the feature that represents reduces its uncertainty after camera motion.

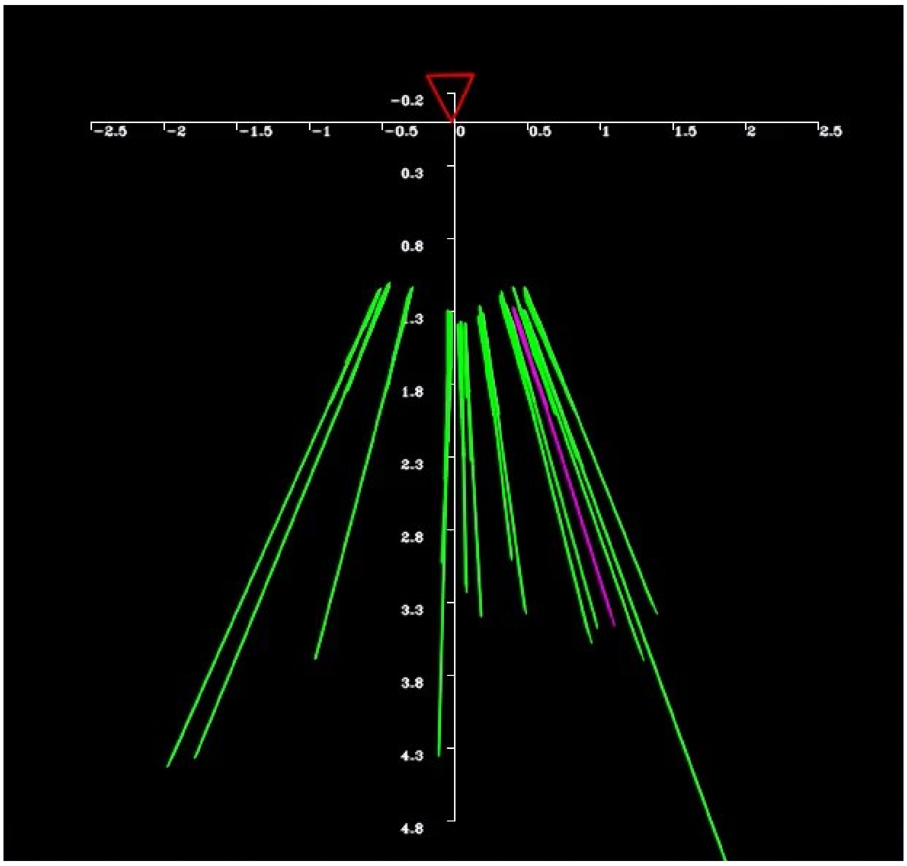
Using a tripod to constrain motion did show stability improvements with the camera estimate, allowing to see orientation changes in each swing. However, this still caused the algorithm to eventually stop working due to matrix inversion problems in Equation (3.71), with the longest run stopping at around 550 seconds in runtime. This is believed to be caused in part by the tripod constrained motion, as the algorithm tries to assume also translation movements which produce little change in feature appearance but keep increasing uncertainty within the algorithm with no recovery. Note that this scenario is unique and a ground of truth is not available for comparison with the same hardware configuration. However, this algorithm implementation follows that of (Montiel et al., 2008), serving as a baseline for further chapters in this thesis.

The camera translation and orientation values can be seen in Figures 3.12 and 3.13. A conversion from the quaternion states to angle degrees is shown for readability purposes in Figure 3.14, this shows the left and right swinging motion applied to the camera. However, it also shows incorrectly estimated values for roll and pitch. This is despite the features having low uncertainty in their states, as this does not mean that the camera ones will have them too. Eventually after adding and removing enough landmarks, the EKF algorithm fails to properly update both state and covariance matrix, leading to uncertainty increase until failure (Figures 3.15 and 3.16).

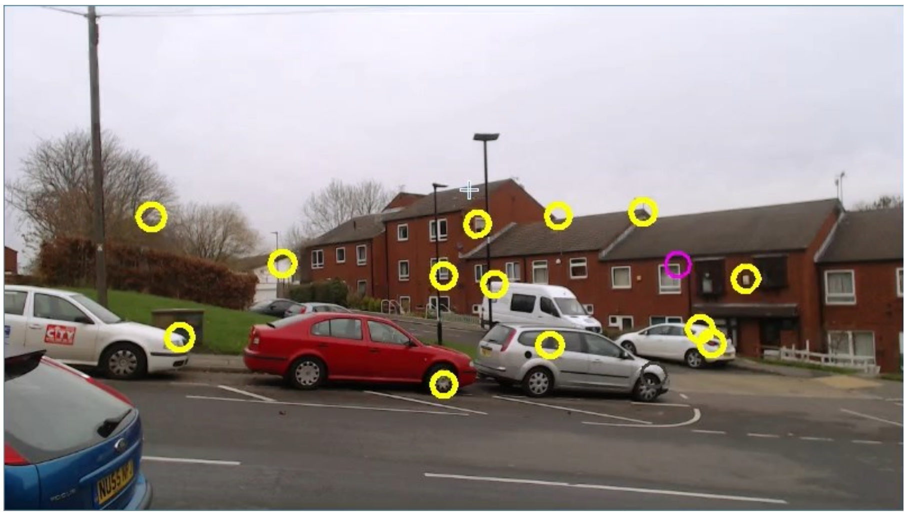
With this it has been seen that the performance of EKF SLAM, even with improvements such as inverse depth parametrisation is prone to stability issues. The next section discusses some key elements regarding the feasibility of a baseline implementation of SLAM with inverse depth parametrisation.

3.4.2 Monocular SLAM with Inverse Depth Parametrisation Feasibility

The results in Section 3.4.1 did not show stable performance in the algorithm, as many problems were encountered in the implementation of the monocular SLAM algorithm with inverse depth parametrisation (Civera et al., 2008). Many of these problems were associated with undesired results in the algorithm, often slowing down leading to program stop within less than five minutes of operation. These can be traced to a particular and crucial step within the algorithm in Section 3.3.5,

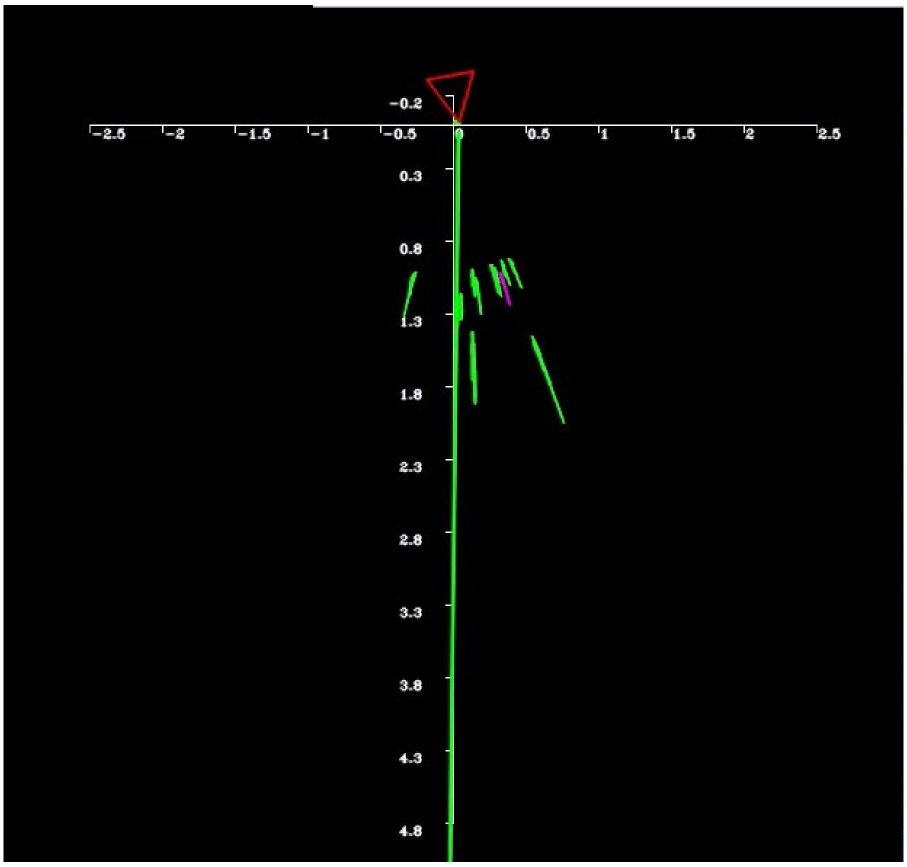


(a)

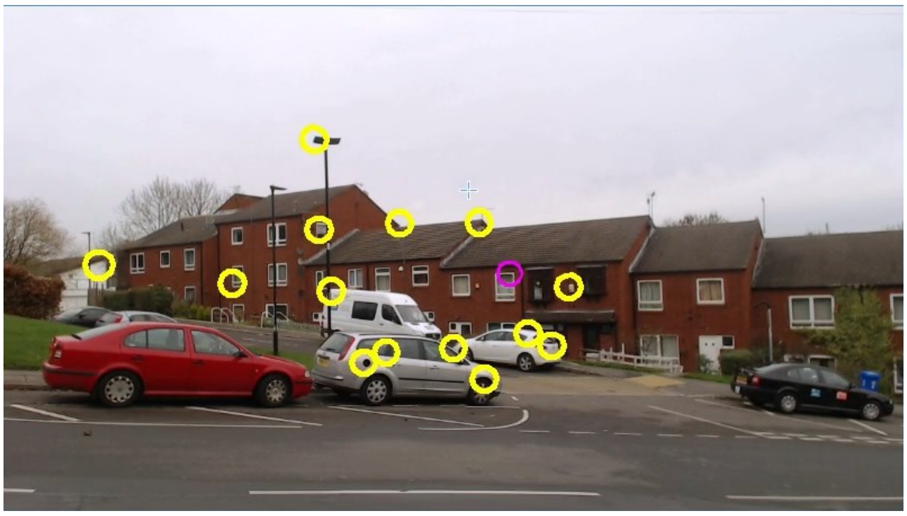


(b)

Figure 3.9: EKF SLAM with inverse depth parametrisation initialisation. Features that have been initialised and mapped are shown in (a) with long elongated ellipses (green), as depth uncertainty is initially big and only reduced after camera movements (red triangle). The algorithm grabs and keeps track of features in (b) (yellow circles). Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.

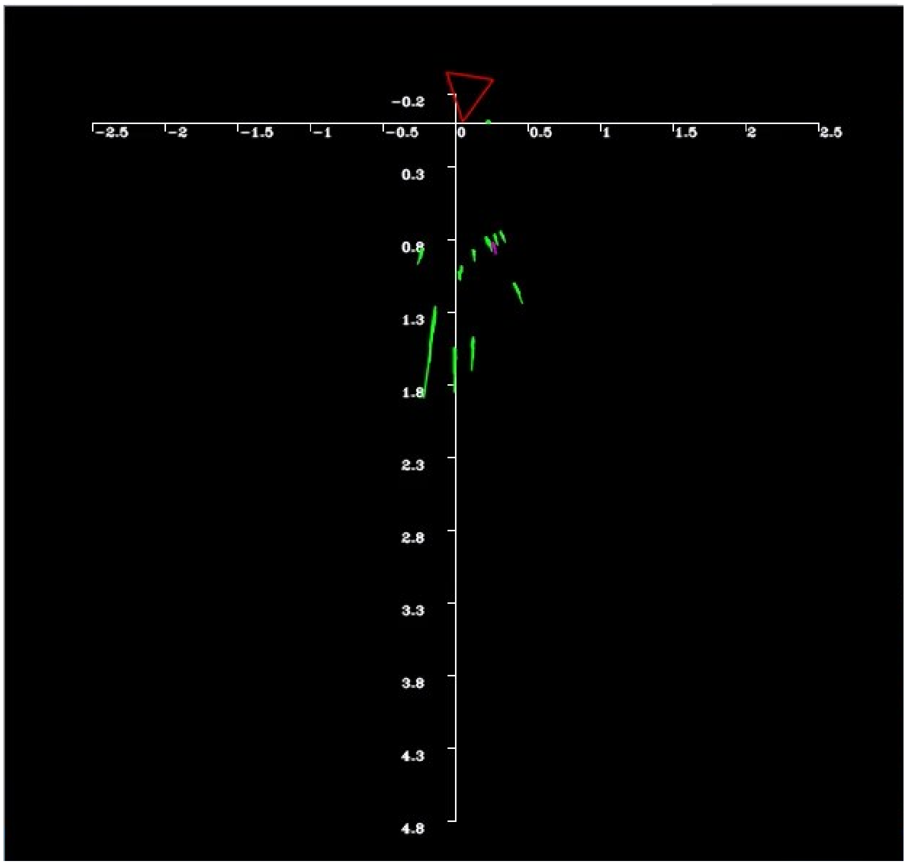


(a)

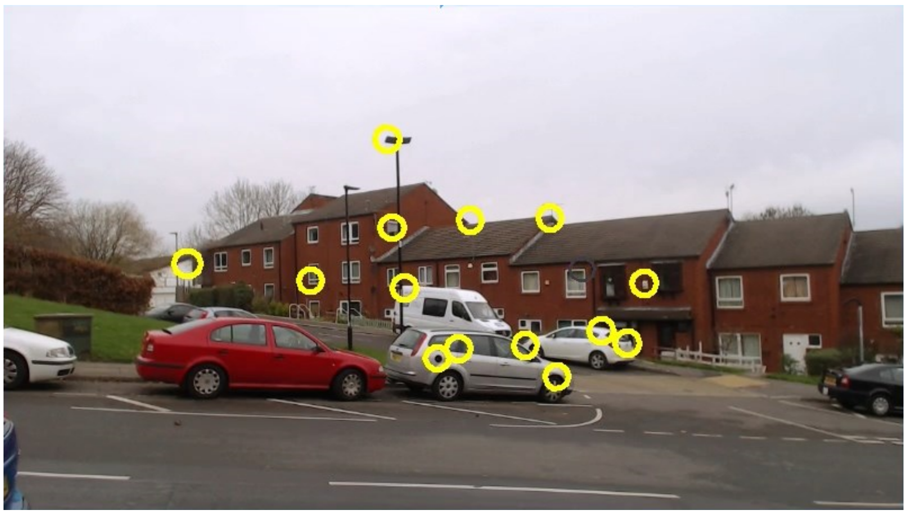


(b)

Figure 3.10: EKF SLAM with inverse depth parametrisation, feature initialisation error. As the camera moves sideways (a) (red triangle) the algorithm reduces depth uncertainty in the features (green). Those that are not discarded remain mostly static in their mapped position except for their depth. Features that failed to be predicted are discarded and replaced by a new one obtained from (b) (yellow circles). However, when feature initialisation fails it often displays the behaviour of a very elongated ellipse (a). Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.



(a)



(b)

Figure 3.11: EKF SLAM with inverse depth parametrisation, low uncertainty features. After enough EKF cycles have passed with the camera moving sideways (red triangle). If the features have not been removed previously, these experience reduced depth uncertainty (green) (a). However, this does not mean that the camera uncertainty is also low. The mapped features correspond to the ones shown in (b) (yellow circles). Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.

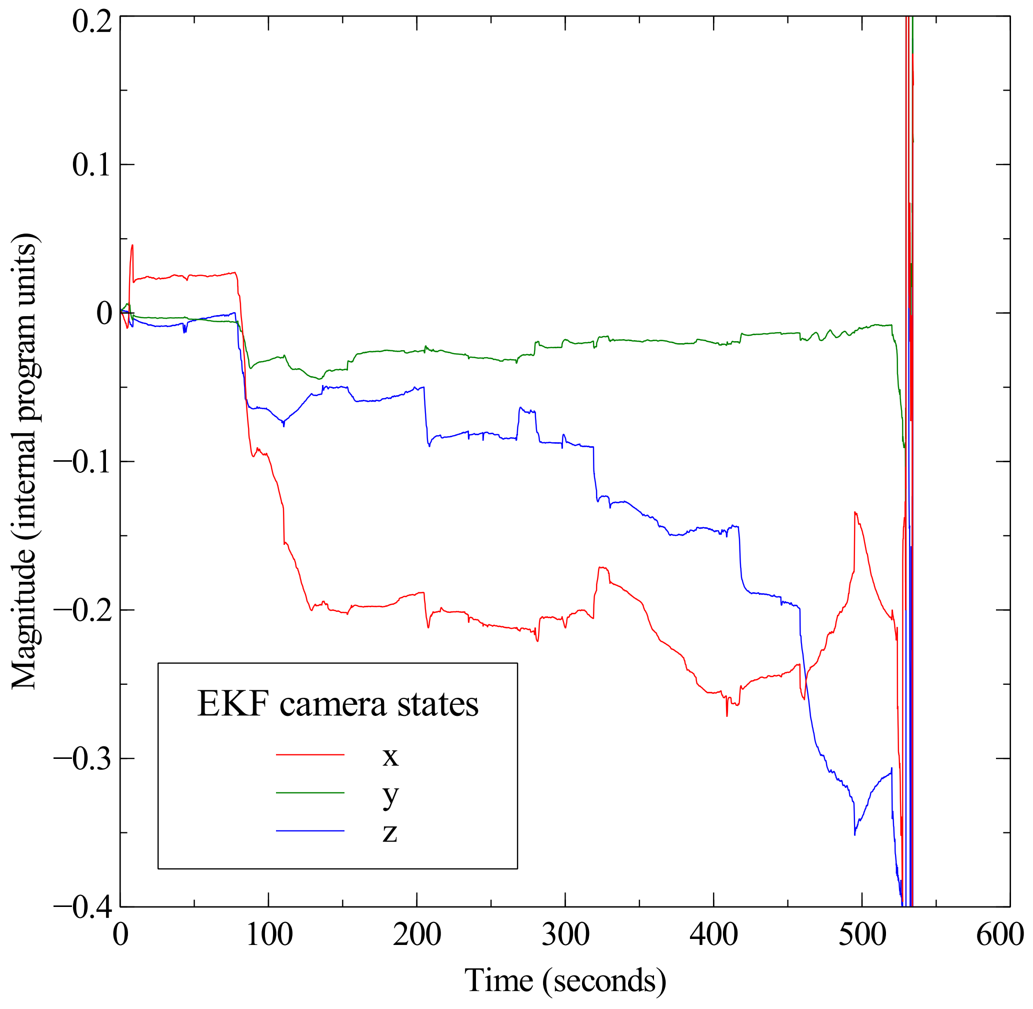


Figure 3.12: Inverse depth parametrisation EKF camera translation states. Around 530s the algorithm fails to properly update the EKF state vector, hence the overshoot in values. Magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

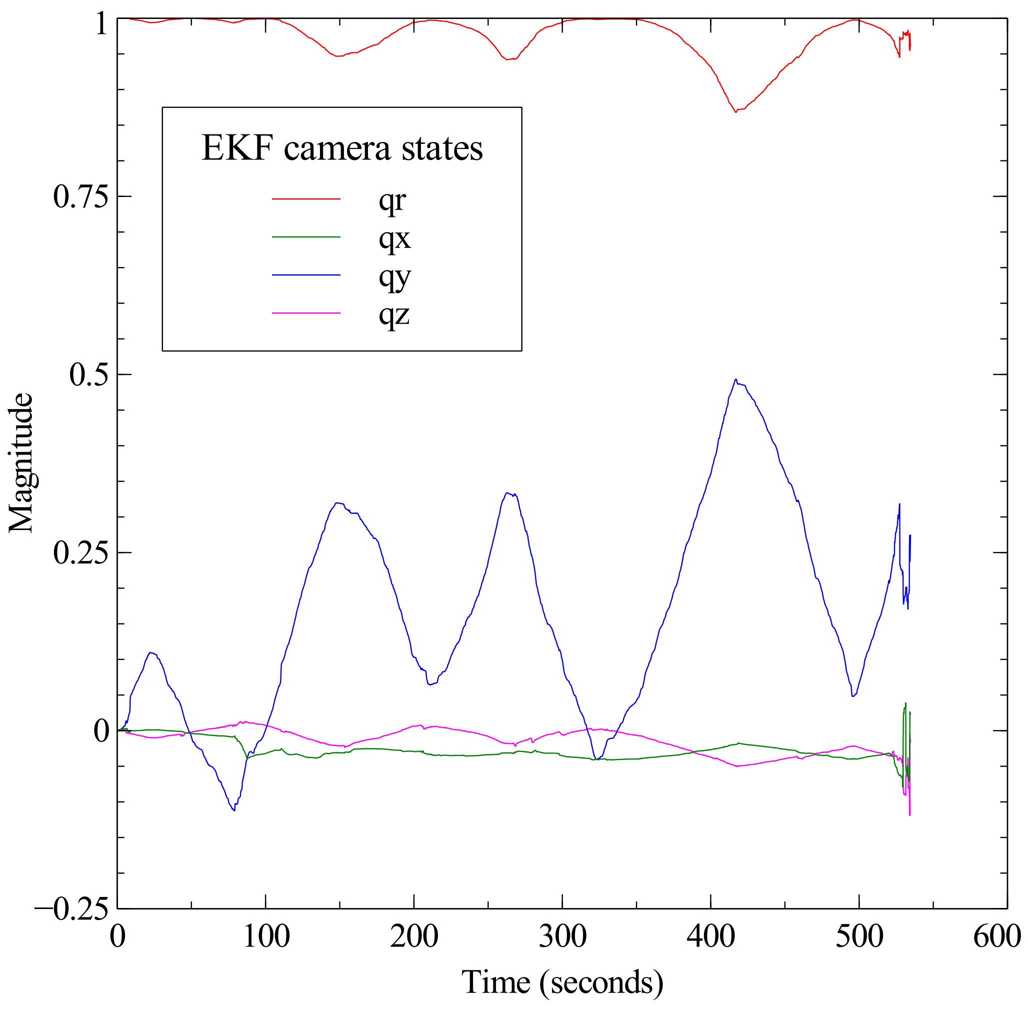


Figure 3.13: Inverse depth parametrisation EKF quaternion states. Around 530s the algorithm fails to properly update the EKF state vector, hence the truncation in the values.

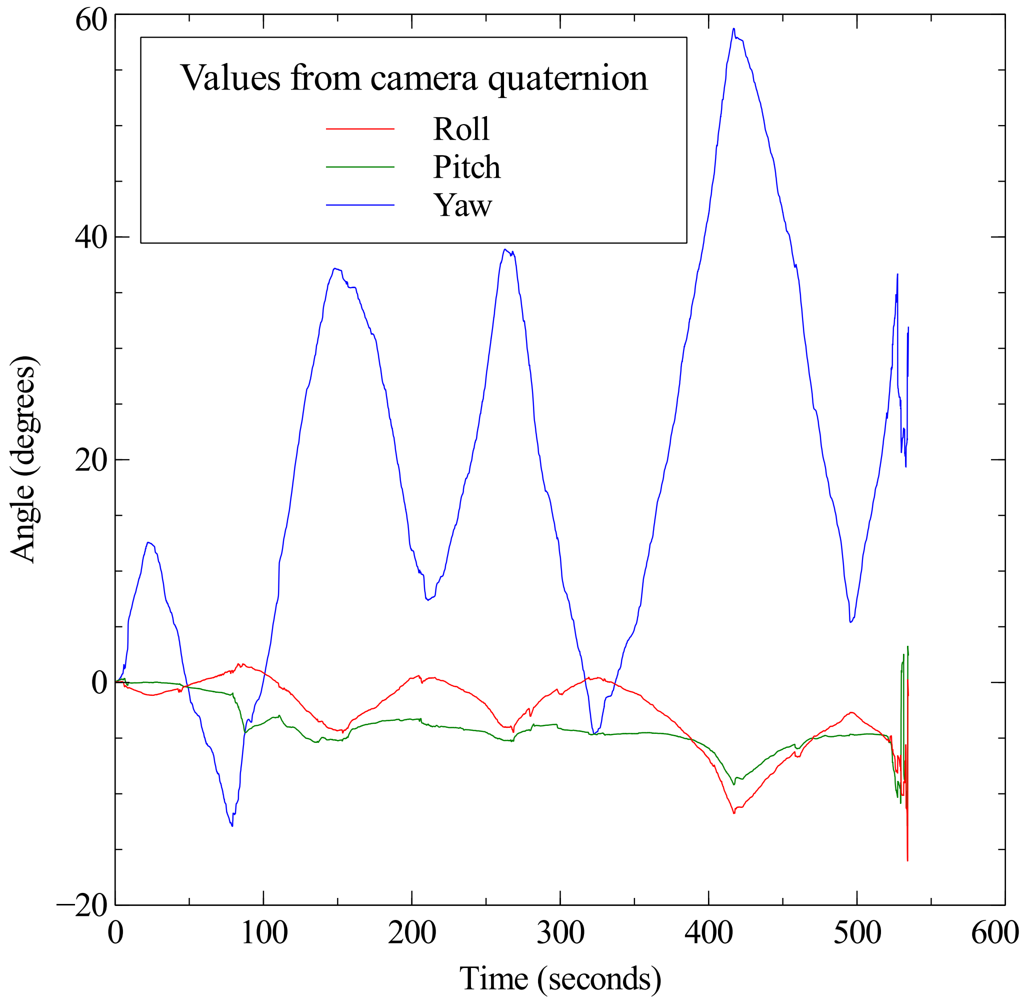


Figure 3.14: Inverse depth parametrisation roll pitch and yaw from EKF quaternion states. The values of the yaw angles follows that of the left and right swinging motion used, with the camera mounted over a tripod. However, pitch and roll are given some values as well which are incorrect estimations. These values have been obtained from the quaternion states in Figure 3.13, according to (Berner, 2008).

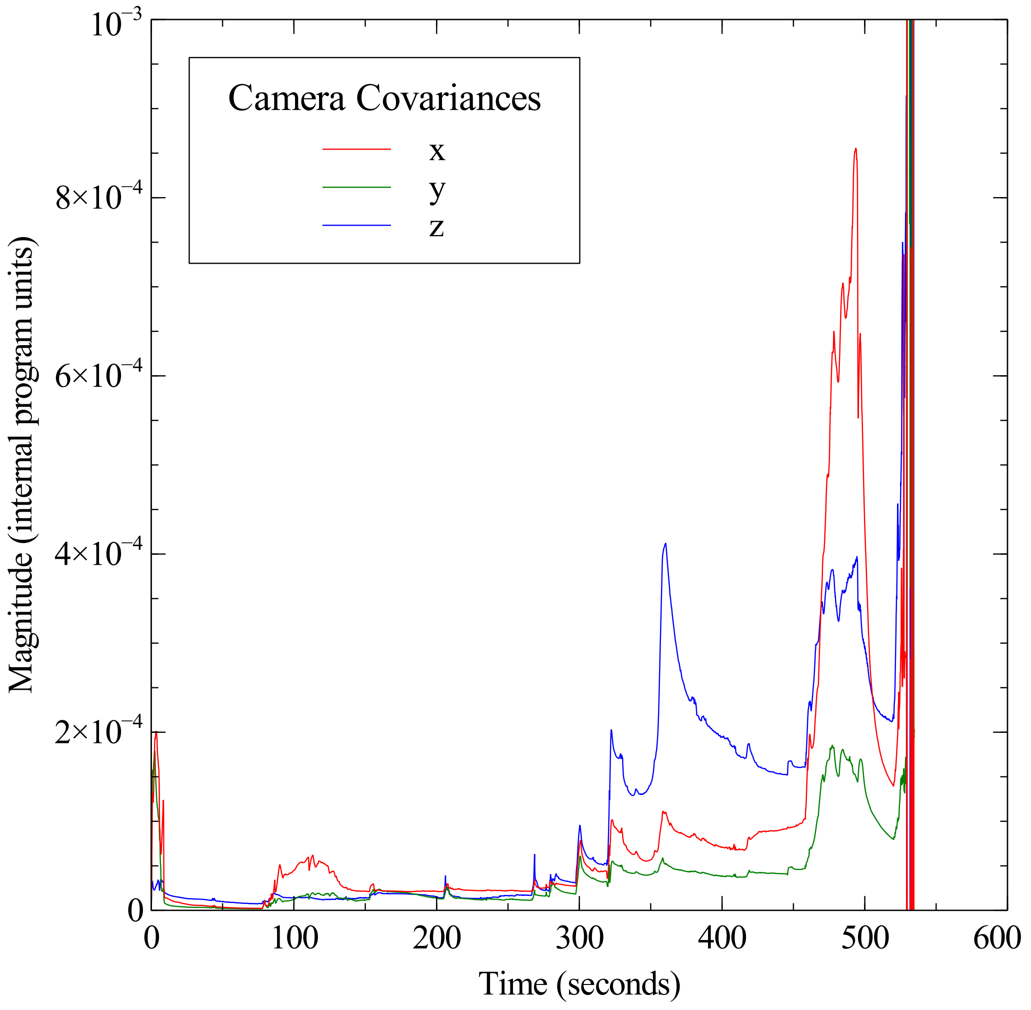


Figure 3.15: Inverse depth parametrisation EKF covariance for camera translation states. Peaks represent whenever a new feature is being added, yet after approximately 90s the algorithm is not able to reduce uncertainty. This leads to failure in properly update the EKF covariance matrix around 530s, hence the overshoot in the values. Covariance magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

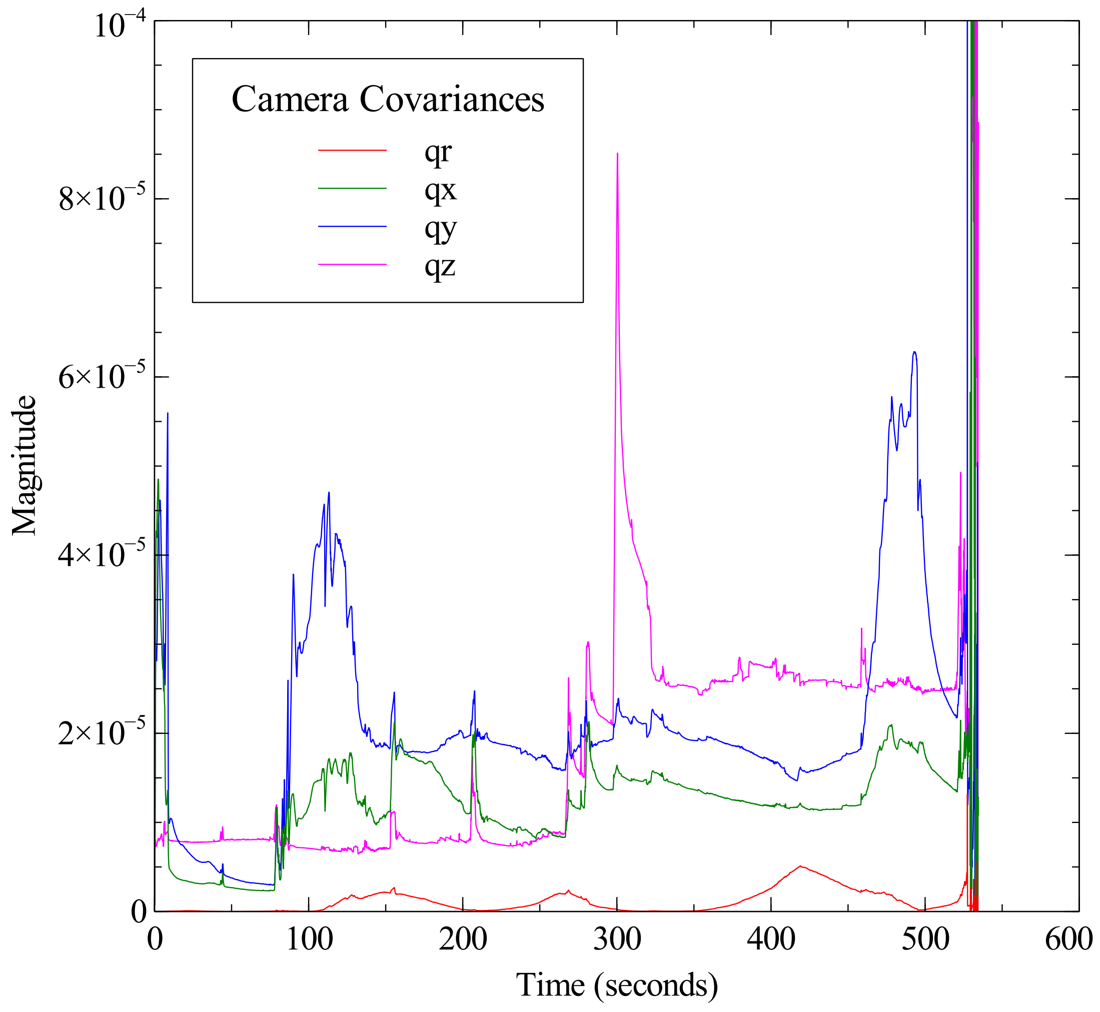


Figure 3.16: Inverse depth parametrisation EKF covariance for camera quaternion states. Peaks represent whenever a new feature is being added, yet after approximately 90s the algorithm is not able to reduce uncertainty. This leads to failure in properly update the EKF covariance matrix around 530s, hence the overshoot in values.

particularly in Equations (3.71) and (3.73).

Equation (3.71) considers the inversion of the innovation covariance matrix, giving as a result the Kalman gain $\mathbf{K}_k$, which subsequently affects both state $\mathbf{x}_k$ and process covariance $\mathbf{P}_k$ updates. Whereas OpenCV (OpenCV, n.d.) offers its own library for matrix inversion, this resulted in random slow downs of the algorithm. Hence it was decided to change the matrix inversion with OpenCV for the same operation using the library *Eigen* (Eigen, n.d.). This resulted in a noticeable performance boost over the execution of the algorithm, but did not resolve the issues of instability.

The calculation of $\mathbf{P}_k$, Equation (3.73), in the EKF can be seen as the main source of program malfunction. This conclusion came after changing the covariance update to $\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_k^-$ as suggested in the original code[*] of SLAM with inverse depth parametrisation and in (Zarchan and Musoff, 2009). This partially resolved the instability issues within the algorithm.

Despite this, the EKF algorithm in this particular implementation can still gather enough drift which would eventually lead to make the EKF algorithm unstable. This might be explained by incorrect state or covariance updates, wrong initialisation parameters or by too big or small values caused by the matrix inversion in Equation (3.71).

3.5 Concluding Remarks

Up until now the entire SLAM with inverse depth parametrisation has been shown. From here it is possible to highlight some key elements in this SLAM approach:

- There is no full compilation of the tools required for a monocular SLAM implementation. This includes much of the EKF formulation as well as considering image defects within the program. This is presented here in order to understand the inner workings of both EKF and camera considerations, with the latter being used for many other vision applications.

- The EKF algorithm presented in Section 3.1 offers an elegant framework for

[*]This is a commented line in Civera's Monocular SLAM code from *http://openslam.org/*

SLAM, as it allows to consider both robot and feature uncertainties. These are stored in a covariance matrix which is then used with an innovation matrix, obtained from the covariances of the obtained measurements to produce a Kalman gain. This last parameter dynamically weights both model predictions and measurements, allowing to obtain a better estimate than if model or measurements were to be used separately.

- Any kind of feature extracted from an image must account for lens deformations. A good solution capable of dealing with radial distortions is the Brown-Conrady model in Equations (3.24) and (3.28), offering also good invertible properties, i.e. it is possible to remove and add deformations to a feature. This allows for more robust feature re-observation.

- Very local features which are automatically extracted are used, for all sorts of operations in SLAM as seen in Section 3.2.1. Therefore, unless dense acquisition is performed they offer little environment description, but dense mapping presents high computational requirements as well. Therefore, user interaction might offer a viable alternative to dense mapping, focusing itself on meaning rather than on tiny elements of an object.

- Monocular SLAM with inverse depth parametrisation is an algorithm that is prone to instabilities, mainly due to the inversion step of the covariance matrix in the EKF as explored in Section 3.4. As features accumulate over time the covariance matrix grows in size, making more difficult to find its inverse. Very small values become very big values and vice versa, making also the inversion prone to numerical errors.

Whereas, theoretically, EKF SLAM is an elegant solution, in practice it tends to present many stability issues. This is possibly caused by constantly adding and removing features from both EKF state and covariance. However, keeping many features is not a solution, as both state and covariance update increase in time according to how many features are being kept.

Therefore, a novel attempt to solve this issue involves active user interaction. This is because a person can set fewer but meaningful features, which can be kept for longer periods of time. This sets the basis in the search for a method which allows to include user interactivity capable of including input semantics.

A first approach for this involves active contours or snakes, allowing user input and at the same time delivering tracking information. Said snakes require external forces which drive them towards a desired contour, therefore a Gradient Vector Flow force is needed for them to work. As such, the next chapter explores Monocular SLAM aided by active contours.

Chapter 4

Active Contours for Interactivity in Vision-SLAM

Chapter 3 offered a detailed analysis of monocular SLAM with inverse depth parametrisation. This also revealed the limitation of using very local features like points, as they require dense representations in order to offer a meaningful description of the environment.

However, instabilities in the algorithm were also seen by several factors which included hand held and tripod constrained motion. This is as the EKF with inverse depth parametrisation algorithm (Montiel et al., 2008), used in Chapter 3 is sensitive to hand-held shaking in feature observations. At the same time it is affected by unrecoverable drift when constrained in a tripod, due to small erroneous estimates that have little impact in feature appearance. Hence, the algorithm obtains observations and takes them as valid for an erroneous estimate.

There is no exact tool for unifying both improved stabilisation and aid one of the main aims in this thesis, which is increased feature meaning. Nevertheless, the clues in Section 2.3 suggested active contours (snakes) based on their deforming and attaching capabilities. These rely on forces generated by objects in the image, which drive and focus them on any immediate image silhouette. This means that initialisation plays an important role for active contours, as this decides to which object of interest they become attached. Dedicated algorithms can be used for this initial step, but at the same time this can be done manually allowing for interactivity in object selection.

Therefore, this chapter presents an investigation related to improve the instabilities found in Chapter 3 whilst finding a solution, for more meaningful generated maps in SLAM without requiring dense representations. This novel approach for monocular SLAM involves an user in the map generation process through interactive active contour initialisation, allowing also for labelling which directly reflect semantics of interest for a person:

- Monocular SLAM with inverse depth parametrisation has been chosen, as it permits to work with only one video feed. This allows real-time operation with a single image source, enabling further processing without severe impact in algorithm performance.
- There is no exact tool that improves over dense feature mapping and shaking in hand-held motion for a baseline monocular-SLAM implementation. Active contours is presented as a novel way to overcome these problems, yet this approach also increases computational requirements. This is as a snake evolves according to external forces generated from a camera picture, partly explaining its missing inclusion within SLAM until now.
- Active contours are also a good option to introduce interactivity in SLAM, thanks to their user driven functionality and tracking capabilities as presented in Section 2.3.1. This enables the user to focus onto object of interest for mapping purposes.
- Generating image forces requires real-time video processing whilst considering available computational power. However, for the case of Gradient Vector Flow (GVF) several iterations are required to attain convergence (Smistad et al., 2012). A General Purpose Graphics Processing Unit (GPGPU) implementation of GVF is used in this investigation allowing to use active contours in SLAM at video rate speeds, enabling real-time interaction within the algorithm.

Section 4.1 describes the methodology required to obtain GVF forces, including the relevant equations in Section 4.1.1, the generation of a binary image in Section 4.1.2, the use of finite differences to extract its gradient in Section 4.1.3 and the required Laplacian in Section 4.1.4. Some improvements in the original methodology are explored in Section 4.2, including a different mask for the edge map and a second order finite difference for gradients in Section 4.2.1, as well as different Laplacian masks in Section 4.2.2.

Later on active contours are explored in Section 4.3, which begins by exploring the formulation of B-Splines, its basis functions and its derivatives in Section 4.3.1. Curve fitting is also detailed in Section 4.3.2 as it is an important part for user feature selection, allowing to create an initial B-Spline. Finally, the active contour evolution based on GVF forces and internal parameters is presented in Section 4.3.3.

The results of applying GVF forces and active contours into monocular SLAM are described in Section 4.4. This includes first active contours using GVF forces accelerated by GPGPU in Section 4.4.1. Applying active contours with GVF forces as features in inverse depth monocular SLAM is presented in Section 4.4.2. Finally concluding remarks are provided in Section 4.5.

4.1 Gradient Vector Flow (GVF)

An active contour is driven by internal and external forces: the former control elasticity properties in the snake, whereas the latter attracts it towards a local minima in an image. GVF belongs to the set of external forces and it is particularly characterised by its good performance in force field generation, which is able to drive snakes even inside of concavities (recall Figure 2.4).

The iterative process of GVF also allows for increased force field range, useful in scenarios where snake initialisation is not precise around an object of interest. This is ideal for a novel interactive vision-SLAM scenario, where feedback is directly given to an user through the camera as it enables user feature selection without requiring precise aim.

As Monocular SLAM relies on single camera usage, GVF can make use of this image to generate forces from it. It is worth of mention that images are not the only source of forces that can drive a snake (Kass and Witkin, 1987). However, image forces are prioritised in this chapter as the main aim is to aid SLAM through object abstraction and information input. Therefore, the basic process involved in GVF force generation is described in Figure 4.1. The following subsections then relate to this steps by exploring mask filtering and finite differences, which allow to prepare an input image for GVF.

4.1.1 Gradient Vector Flow Equations

External image forces generated from an object of interest allow an active contour to deform and follow its shape. This is as the object produces changes in contrast, which in turn produce gradients and later forces resembling their silhouettes as in Figure 4.1d. These also represent a local minima within the output of GVF after the image has been processed.

Therefore, given an image a GVF vector field $\mathbf{v}(x, y) = [u(x, y), v(x, y)]$ can be obtained by minimising the energy functional E_{GVF} (Xu and Prince, 1998b):

$$E_{GVF} = \iint \mu(u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla f|^2 |\mathbf{v} - \nabla f|^2 dxdy \tag{4.1}$$

where the vertical and horizontal forces are represented by u and v respectively, each one mapping a two dimensional space with u_x, u_y, v_x and v_y components. The μ constant is a term that weights the first operand $(u_x^2+u_y^2+v_x^2+v_y^2)$ and must be set according to the noise present in the image. However, setting a constant value of 0.1 for this parameter is recommended (Xu and Prince, 1998b). The gradient of a binary edge map such as the one in Figure 4.1c is represented by ∇f, which remains constant within iterations.

In order to find $\mathbf{v}$, calculus of variations can be used to solve Equation (4.1), as it is assumed that E_{GVF} possesses a local minima, resulting in the following set of Euler equations (Gelfand and Fomin, 2000, Wikipedia, n.d.):

$$\mu\nabla^2 u - (u - f'_x)(f'^2_x + f'^2_y) = 0 \tag{4.2a}$$

$$\mu\nabla^2 v - (v - f'_y)(f'^2_x + f'^2_y) = 0 \tag{4.2b}$$

In order to solve Equations (4.2a) and (4.2b) a numerical implementation is required, which can be achieved by treating u and v as functions of time (Xu and

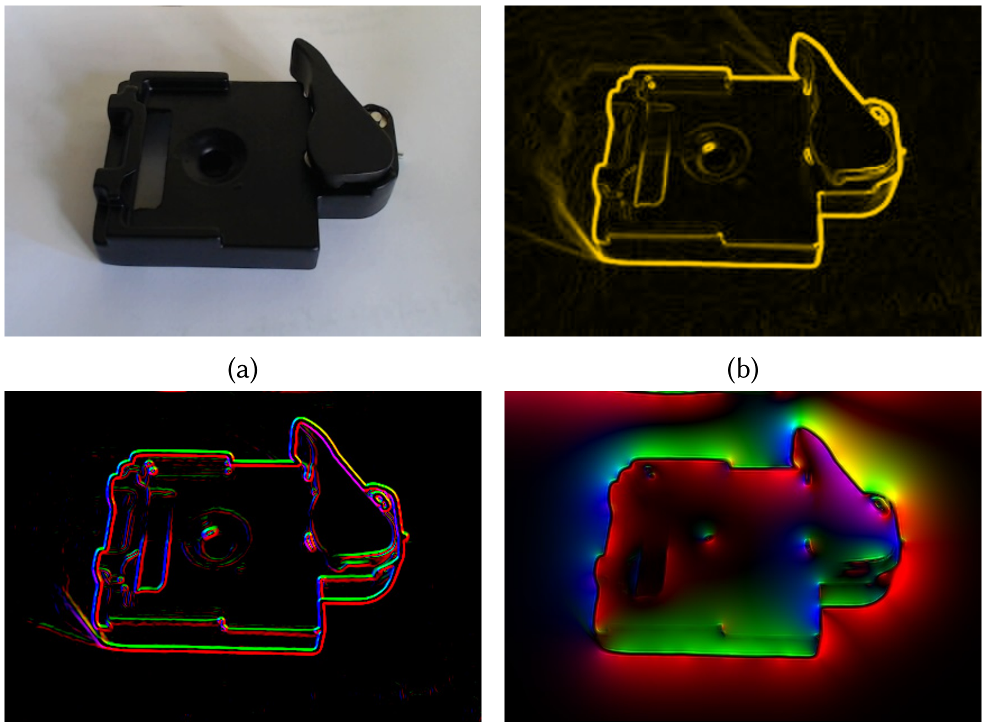


Figure 4.1: Gradient Vector Flow steps. (a) depicts the image as obtained from the source, with all its red, green and blue (RGB) pixels. (b) illustrates the binary edge map image obtained using masks from (a). Note that prior to generating a binary edge map image the original picture is simplified through greyscale conversion, so only intensity values remain as in Equation (3.22). (c) shows the u and v components of the binary edge map image from applying finite differences in x and y over (b). Finally (d) iteratively uses data from (c) within the GVF algorithm, resulting in a "washing" effect. Colours indicate the forces, if a snaxel (snake element) were to be put near any of the colours it would get attracted towards the local minima (black silhouette) of the image contour.

Prince, 1998b):

$$u_t(x,y,t_k) = \mu\nabla^2 u_t(x,y,t_{k-1}) - \left[u_t(x,y,t_{k-1}) - f'_x(x,y)\right] \cdot \left[f'_x(x,y)^2 + f'_y(x,y)^2\right] \tag{4.3a}$$

$$v_t(x,y,t_k) = \mu\nabla^2 v_t(x,y,t_{k-1}) - \left[v_t(x,y,t_{k-1}) - f'_y(x,y)\right] \cdot \left[f'_x(x,y)^2 + f'_y(x,y)^2\right] \tag{4.3b}$$

where $u_t(x,y,t_k)$ and $v_t(x,y,t_k)$ are the GVF horizontal and vertical forces respectively at iteration t_k; outputs from applying finite differences (Figure 4.1c) to an image's binary edge map (Figure 4.1b) deliver the gradients $f'_x(x,y)$, $f'_y(x,y)$, discussed in more detail in Section 4.1.3. These outputs remain constant within all the GVF iterations executed over the picture. The Laplacian operator applied to $u_t(x,y,t_{k-1})$ and $v_t(x,y,t_{k-1})$ is represented as $\nabla^2 u_t(x,y,t_{k-1})$ and $\nabla^2 v_t(x,y,t_{k-1})$, with t_{k-1} indicating its past GVF iteration output. For the first run of GVF, $u_t(x,y,t_{k-1})$ and $v_t(x,y,t_{k-1})$ are just copies of the gradients $f'_x(x,y)$ and $f'_y(x,y)$, the results $u_t(x,y,t_k)$ and $v_t(x,y,t_k)$ recursively become $u_t(x,y,t_{k-1})$ and $v_t(x,y,t_{k-1})$ on the next iteration. The Laplacian operator will be discussed in Section 4.1.4.

Equations (4.3a) and (4.3b) are used for an iterative GVF implementation allowing to form a vector field (Figure 4.1d), taking each element from previously obtained binary and gradient output images. The next sections pertain to the acquisition of these required images, which mainly rely on mask that are special matrices multiplying each pixel over a simplified greyscale image.

4.1.2 Binary Edge Map

A binary edge map must be generated prior to obtaining the gradients $f_x(x,y)$ and $f_y(x,y)$ in Equation (4.3) (Xu and Prince, 1998b). This allows to obtain silhouettes produced from objects in images which will serve as the local minima for GVF, removing necessary information like colour*. Therefore, a greyscale picture is first obtained from the original image taking in account each red, green and blue elements like in Equation (3.22). This produces an image $I(x,y)$ with only intensity values contained in it.

*A Gaussian filter can also be applied right after the greyscale conversion to smooth out some of the noise present in the picture.

The creation of the edge map can be achieved by using the Sobel or the Scharr operators as described in (Sobel, n.d.). This operation can be executed by convolving the previous greyscale picture $I(x, y)$ with either one of the following set of two masks, which obtains two sets of data $b_x(x, y)$ and $b_y(x, y)$:

$$\text{Sobel Masks} \qquad G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.4a}$$

$$\text{Scharr Masks} \qquad G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \qquad G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \tag{4.4b}$$

$$\text{Magnitude} \qquad G = \sqrt{G_x^2 + G_y^2} \tag{4.4c}$$

An example of image convolution with these masks is shown in Figure 4.2. The magnitude in Equation (4.4c) produces a binary image $b(x, y)$, which is obtained by using the elements of two arrays $b_x(x, y)$ and $b_y(x, y)$. An example of a resulting binary image is illustrated in Figure 4.3, showing the application of G_x and G_y Scharr mask set in an intensity image $I(x, y)$ from Equation (4.4b). Once the binary image has been extracted the gradients are then obtained through finite differences, which then serve as input for GVF.

4.1.3 Finite Differences

The past section presented the calculation of an edge map with help of Equation (4.4c), which delivers a binary image $b(x, y)$ containing mostly silhouettes or differences in contrast from the input greyscale image $I(x, y)$. For the numerical implementation of GVF, derivatives from the generated binary image are needed following Equation (4.3) in Section 4.1.1, but as the array is in discrete form these derivatives must be calculated using finite differences.

Thus, a first order central difference approximation to the derivative is defined

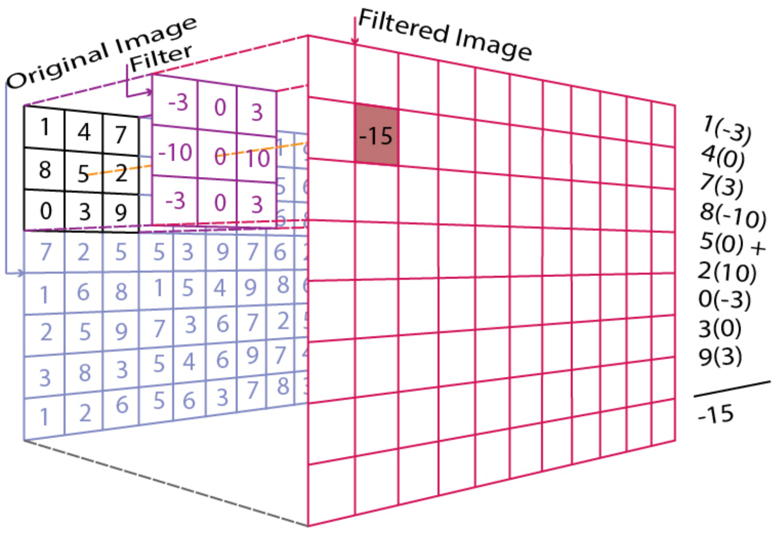
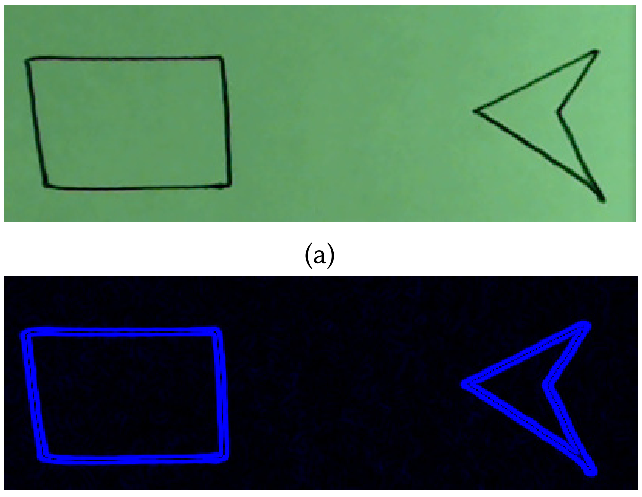


Figure 4.2: A filter mask convolving with an image. A pixel value is selected from any RGB channel or greyscale picture and acts as the centre of the mask. Surrounding pixels are then read according to the mask size so that each of them is multiplied in a 1×1 relationship, the result is then summed and placed exactly on the same position as the centred pixel.



(a)

(b)

Figure 4.3: Binary Image. A source image depicts simple shapes in (a), whereas (b) shows the resulting image after applying the Scharr masks prior conversion to greyscale of (a). These matrices detect changes in contrast obtaining a high value whenever one is found, with contours representing these intensity changes in (b).

in the following way (Mathews and Fink, 2004):

$$f'(x) \approx \frac{b(x+h) - b(x-h)}{2h}$$

Obtaining a derivative from the edge map $b(x, y)$ requires two operands since the binary image $b(x, y)$ has discrete pixel coordinates (x and y) in a similar way to what was described in Section 4.1.2, requiring to apply finite differences in horizontal and vertical directions. Therefore, an approximation of the first derivative using central differences is given by:

$$x = 0 \cdots x_{img}$$

$$y = 0 \cdots y_{img}$$

$$f'_x(x, y) \approx \frac{b(x+h, y) - b(x-h, y)}{2h} \tag{4.5a}$$

$$f'_y(x, y) \approx \frac{b(x, y+h) - b(x, y-h)}{2h} \tag{4.5b}$$

where the total image dimensions are given by x_{img} and y_{img}. The displacement h is set to 1 since our next point of data is the following pixel. It is important to remark that this operation will yield two sets of data arrays $f'_x(x, y)$ and $f'_y(x, y)$, each one possessing x and y coordinates and that serve as input in Equation (4.3). Finally the last operation remaining for GVF is applying the operator ∇^2.

4.1.4 The ∇^2 operator in Gradient Vector Flow

As a final step in GVF the operator $\nabla^2 f(x, y)$ needs to be used in order to attain the numerical implementation of GVF seen in Equation (4.3), which is defined as[*]:

$$\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y) \tag{4.6}$$

However, this equation also needs to be applied to discrete data represented by 2 dimensional x and y arrays. To solve this numerically an approximation of the second derivative of $f(x)$ must be obtained, e.g. finite differences. As such, it is

[*]A more complete and comprehensive explanation can be found in (Keller, n.d.)

possible to use the following second order central formula:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{4.7}$$

Therefore, it is possible to use the following central difference approximations to the second derivatives of the $\nabla^2 f(x, y)$ operator:

$$\frac{\partial^2}{\partial x^2} f(x, y) = f''_x(x, y) \approx \frac{f(x+h, y) - 2f(x, y) + f(x-h, y)}{h^2} \tag{4.8a}$$

$$\frac{\partial^2}{\partial y^2} f(x, y) = f''_y(x, y) \approx \frac{f(x, y+h) - 2f(x, y) + f(x, y-h)}{h^2} \tag{4.8b}$$

where both resulting arrays have the same total elements consisting of x_{img} and y_{img} as the input image.

It is worth mentioning that the first run of GVF, $u_t(x, y, 0)$ and $v_t(x, y, 0)$ are $f'_x(x, y)$ and $f'_y(x, y)$ respectively from the binary edge map in Section 4.1.3. In subsequent runs $u_t(x, y, t_k)$ and $v_t(x, y, t_k)$ become the GVF outputs from Equations (4.3a) and (4.3b).

Considering Equation (4.6), together with a displacement h of 1 pixel in the approximations from Equations (4.8a) and (4.8b), the ∇^2 operator becomes:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{4.9}$$

If the terms of Equation (4.9) are arranged in a bi-dimensional matrix they yield:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.10}$$

which in image processing is more often known as the Laplacian mask.

With this, all the equations needed for a numerical implementation of GVF in Section 4.1.1 have been shown. These basically allow to apply a Sobel or Scharr mask to a greyscale image, which generates a binary image depicting only silhouettes. Finite differences are applied afterwards, which produce the gradients needed to use the ∇^2 operator.

One of the novelties in this investigation resides in the fact that GVF has been selected as a driving mechanism for snakes in SLAM. However, its CPU implementation makes it unsuitable for real-time operation. This is because image filtering is considered a highly demanding computational process in CPUs. Therefore, the next section explores the implementation of the GVF forces into a graphics processing unit (GPU) through GPGPU, allowing for huge parallel processing and high speed gains with it.

4.2 Gradient Vector Flow and operations in GPGPU

The basics covering the GVF have been discussed in Section 4.1, disclosing the equations needed for a numerical implementation of it using discrete images. However, by trying to use GVF as a real-time tool a considerable number of pixels needs to be processed many times a second. This is not only because of dimensions or total amount of pixels in the image but also due to the number of iterations N_{GVF} required. Indeed, it is noted that GVF must be iterated as much as needed in order to reach convergence, with a number going around 400 to 512 to be said good enough and the lowest being 256 (Smistad et al., 2012).

Given this it might be argued that one reason for active contours to have been ignored within the SLAM context is because they require a huge amount of computational resources, which are often reserved for SLAM itself as it consumes also a considerable amount of CPU power. However, nowadays there are more dedicated tools for specific tasks like image manipulation which are often considered highly parallel processes. Compared to a serial workload like SLAM which extensively uses a CPU, parallel tasks can be set apart in a GPU, allowing to increase the performance of an algorithm like GVF.

This thesis proposes to use General Purpose computing on Graphics Processing Units (GPGPU)[*] for GVF in a novel SLAM approach. A basic diagram of this GPGPU GVF implementation is given in Figure 4.4. GPGPU are an accessible and affordable, highly parallel image processing tool allowing to pursue a novel interactive snake application in SLAM: Actively using GPGPU GVF forces, obtaining tracking information from the evolution of an active contour which is later

[*]Further information about GPUs and their inherent parallel processing capabilities are given in (Singer, 2013.)

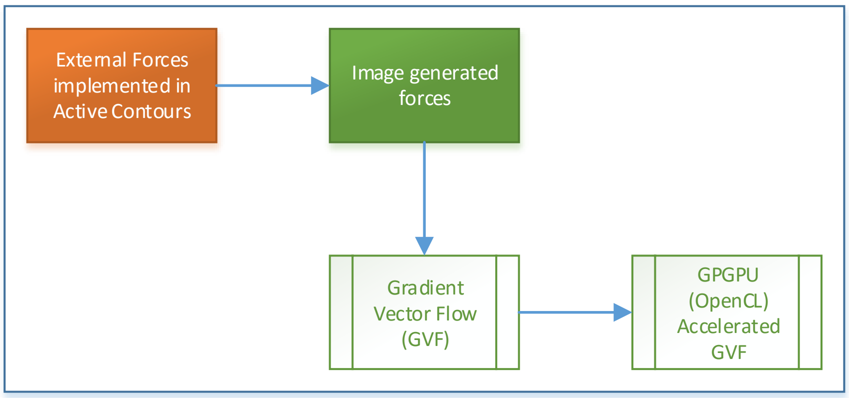

External Forces implemented in Active Contours
Image generated forces
Gradient Vector Flow (GVF)
GPGPU (OpenCL) Accelerated GVF

Figure 4.4: Gradient Vector Flow forces accelerated by GPUs for active contours. Image forces are part of the external forces that can drive a snake. GVF forces take images as input but require the processing of all pixels many times each iteration. A GPGPU implementation of GVF increases the overall performance when monocular SLAM is also considered, as the CPU is not taxed with GVF forces generation saving resources for the SLAM execution. This also allows to consider images containing more pixels in them, thus capturing more information usable in both GVF and monocular SLAM.

used for feature abstraction in SLAM. Therefore, following is a description of the used hardware as well as key practical implementations for improved GVF performance.

4.2.1 The Image's Binary Edge Map

In order to generate GVF forces through GPGPU acceleration a graphics card supporting an Application Programming Interface (API) such as OpenCL is required. This API allows to use a set of defined instructions that will deliver an expected determined result aided by GPUs*. In this case OpenCL can make use of GPUs of different architectures to handle highly parallelised workloads.

Once the image has been converted to greyscale the next step is to obtain its gradient. To calculate the gradient a 2 step process was devised: the first one

*More hardware is supported by OpenCL, however GPUs offer a cheap alternative present in almost all computers, including laptops. Note also that OpenCL is not the only API able to do this, the proprietary technology of CUDA in Nvidia cards can also be used. However, OpenCL has the advantage of being multi-device and multi-platform.

originally was the convolution of one pair of masks as seen in Equations (4.4). The Sobel or the Scharr 3×3 kernels are commonly used for this task. However, due to the advantage of using OpenCL a higher size mask would yield better results with almost zero impact on performance. The values for producing a 5×5 kernel are (Scharr, 2000)*:

$$\begin{bmatrix} 21.38 & 85.24 & 0 & -85.24 & -21.38 \end{bmatrix} /256 \tag{4.11a}$$

$$\begin{bmatrix} 5.96 & 61.81 & 120.46 & 61.81 & 5.96 \end{bmatrix} /256 \tag{4.11b}$$

The transposes of both (4.11a) and (4.11b) must be multiplied in order to find the 5×5 G_x mask used for the image convolution, G_y can be found by transposing G_x.

Next, it is required to apply the finite differences method as mentioned in Section 4.1.3. GPU acceleration with OpenCl can be used to improve this stage, by using a higher order central difference formula (Mathews and Fink, 2004):

$$f'(x) \approx \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x+2h)}{12h} \tag{4.12}$$

A remarked requirement for GVF is a binary edge map (Xu and Prince, 1998b). Hence, the method described here is not the only way to obtain it. Other examples include using Anisotropic Filtering, which is a non-linear process that is highly insensitive to noise but requires more computational resources (Perona and Malik, 1990). In this thesis Equation (4.12) is used due to its computational simplicity.

4.2.2 GVF and Other Laplacian Operators

The Laplacian mask is very simple to implement and swap on the OpenCL API. Nevertheless, there are some recent researches involving more elaborate Laplacian masks. These include using a 5×5 sized mask instead of the one shown in Equation (4.10), which allows to consider more surrounding pixels (Wu and Zhang, 2012). A similar approach uses a same sized 5×5 mask for a low pass filter and another one for noise reduction (Liu and Bovik, 2012). However, while very interesting in principle these approaches failed in tests. These filters amplify the noise contained within the image rather than to mitigate it as illustrated in Figure 4.5.

*This investigation also shows comparisons with other masks.

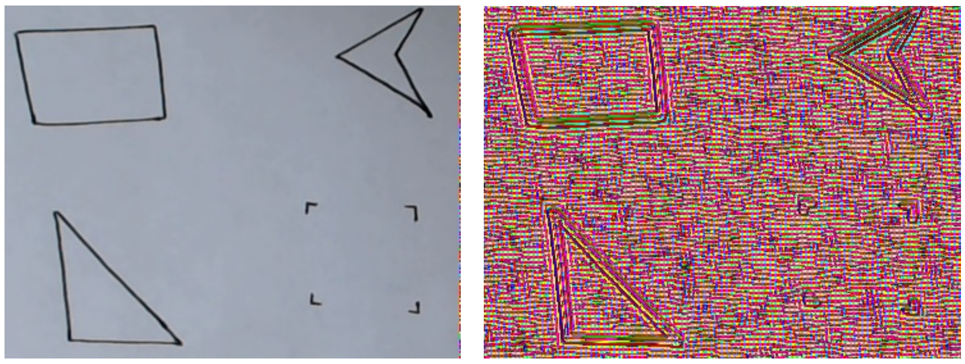

(a) (b)

Figure 4.5: GVF after applying newer Laplacian masks. (a) shows a sample image containing test silhouettes, whereas (b) illustrates their GVF output after the Laplacian masks in (Wu and Zhang, 2012, Liu and Bovik, 2012) were applied. Even when the silhouettes from (a) can be distinguished, the noise becomes severely amplified in the resulting GVF output in (b).

These results were unexpected considering how well they seem to appear at least on paper. Therefore, it was decided to keep the Laplacian mask in Equation (4.10) as this is already proven. The results show similar output to the one previously shown in Figure 4.6 as well as those in (Smistad et al., 2012)[*].

All these subsections have lead to create the external forces needed for guiding active contours. The latter is particularly important as is it more of an user assist tool, thus not relying on fully automated methods to select features within SLAM. As such, active contours are explored next.

4.3 Active Contours

Active contours allow to form a curve which dynamically deforms according to internal and external factors. The latter includes forces produced by objects, which then allow the contour to shape itself around them. An example of such forces is the GVF from Section 4.1, used in this investigation because of its performance, which can be greatly increased thanks to GPGPU acceleration as seen in Section 4.2.

[*]Author provided code usable only by command prompt and with a static image can be found in (Smistad, 2013).

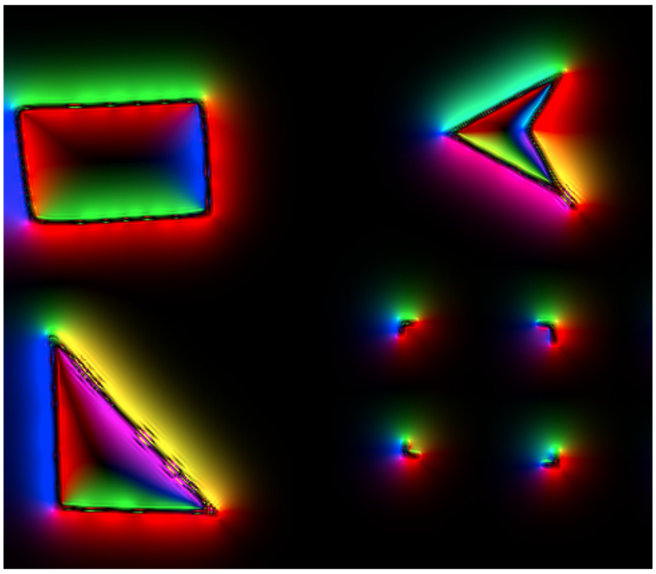


Figure 4.6: Gradient Vector Flow applied to different contours, colours represent the attractive forces on the image, well outside of their shapes. These are the forces capable of driving an active contour.

The fast generation of GVF forces allows an active contour to deform fast enough, so that it can keep a lock onto an object without any sort of help. Therefore, if tracking information is extracted this would be valuable for camera motion tracking. For SLAM, this is a new approach, as fast active contours are considered for map and motion estimation. The main interest is to focus onto an object rather than localised and automated features, which then allows for localisation and improved meaningful mapping.

However, the focus onto an object is not an automated task in this investigation. Here it is attempted to use a person's capability to discern objects, thus initialising an active contour around them by selecting points in an image. An initial B-Spline would be created that interpolates these points, with GVF forces driving it eventually taking the object's shape. Figure 4.7 illustrates all the stages needed for active contour evolution.

Active contours make use of the calculus of variations. Therefore, an introduction can be found in (Arfken et al., 2013) whereas more abstract information can be read from (Gelfand and Fomin, 2000). However, here the simpler theoretical foundations needed for its implementation will be presented in the next subsections following (Rogers, 2001). These include the basics behind B-Splines

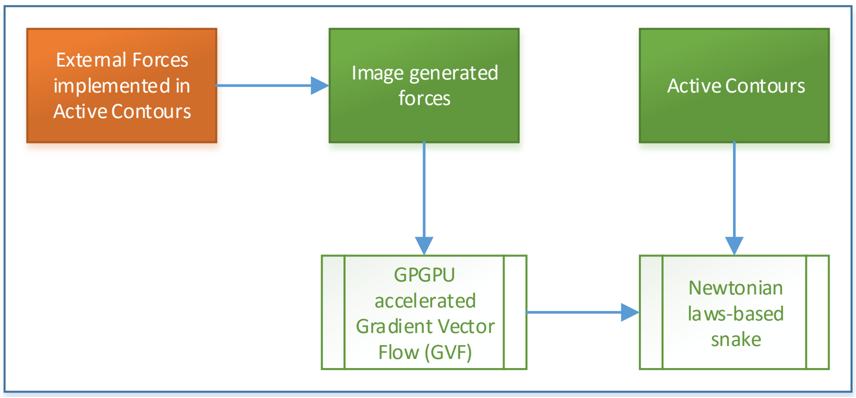

External Forces implemented in Active Contours
Image generated forces
Active Contours
GPGPU accelerated Gradient Vector Flow (GVF)
Newtonian laws-based snake

Figure 4.7: Active contours require the external forces which guide them towards a desired local minima. Using active contours in SLAM was presumably overlooked as GVF forces are computationally demanding for CPUs. However, GPGPU acceleration allows to increase the performance of GVF forces by a considerable margin. Thus allowing video rate GVF forces to be used in snake evolution.

and its derivatives, which allow to perform curve fitting using a series of points based on a lower count of control polygon points and basis functions. Joining both GVF forces and user initialised B-Splines gives way to a novel monocular SLAM approach, which allows inferring camera location with object tracking through snake evolution. This also allows for included description in the moment of object selection, improving meaning in the generated SLAM map.

4.3.1 B-Splines, Basis and its Derivatives

A B-spline is a tool that allows to perform smoothing using data points. This is done using multiple polynomial functions, i.e. piecewise polynomial. This means that it is possible to manipulate a B-Spline locally, changing only desired parts of the data smoothing. The B stands for basis which are the B-Spline pieces uniquely defined given a set of *knots*, with the latter being the parts where the pieces meet. An example of a cubic B-Spline is seen in Figure 4.8.

Part of the novelty in this investigation is its proposition on interactivity within vision-SLAM. For this, B-Splines offer a good solution as their initialisation can be automated or user guided. This not only introduces a person into the SLAM algo-

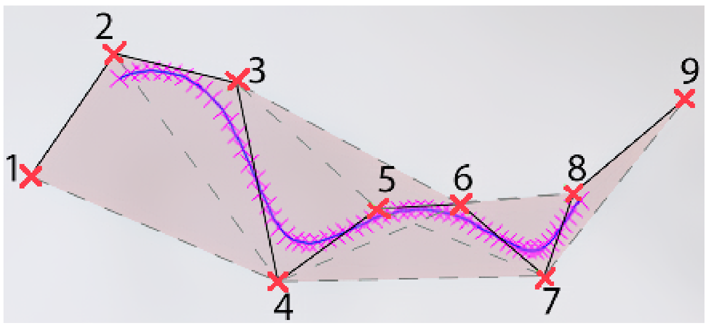
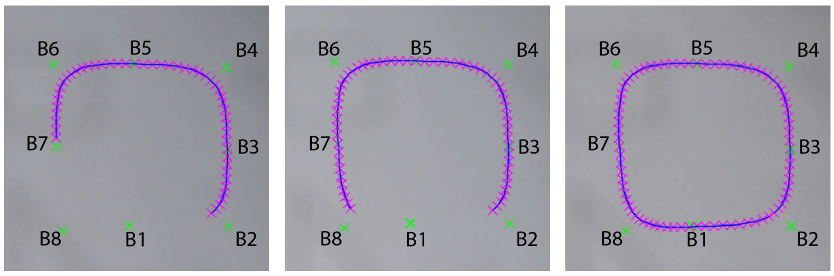


Figure 4.8: A cubic (order k = 4, degree 3) uniform (equidistant knot vector numbers) B-Spline with 9 control polygon points (red crosses, n = Control Polygon Points − 1 = 8). The B-Spline (blue) and its discrete points (purple crosses) lie always within the convex hull (segmented line) created by the control polygon points. The higher the order the more the area the convex hull produces, for k = 2 this hull disappears and the B-Spline can only follow the control polygon line (black). Note how the curve does not begin on the first control polygon point nor ends at the last one, this is because the parameter range t is directly influenced by the order of the B-Spline: the higher the order the smoother the B-Spline becomes but also the more the parameter range is reduced, thus shorter distance drawn by the B-Spline.

rithm, but also allows to include other information along with the object selected for map meaning or context purposes. To initialise the B-Splines an user is able to select points over an object of interest from the camera image, which later serve as the data points used for smoothing. The generated GVF forces would then drive the B-Spline closer to the object's silhouette, allowing to track the object. This is because the B-Spline responds to changes when responding to the attractive forces of GVF, constantly changing small portions if it to better fit the contour of the object.

As such, B-Splines are a first step into active contours. Some of their main properties are described next (Cox, 1972):

1. The B-Spline order k must be $>$ 2 but less than or equal to the total number of control polygon vertices. The maximum degree n is one less (k − 1).
2. The B-Spline exhibits the variation-diminishing property. This means that the produced curve does not oscillate about any straight line more often than its control polygon oscillates about the line.
3. The B-Spline generally follows the shape of the control polygon.



(a) (b) (c)

Figure 4.9: Open and closed cubic B-Splines. (a) depicts a B-Spline with no repeated polygon vertices { B1 B2 B3 B4 B5 B6 B7 B8 } while (b) illustrates the same B-Spline with 1 repeated control polygon point { B1 B2 B3 B4 B5 B6 B7 B8 B1 }. Finally a fully closed B-Spline is obtained in (c) with k − 2 extra polygon points aside from just B1, i.e. { B1 B2 B3 B4 B5 B6 B7 B8 B1 B2 B3 }.

4. Any affine transformation is applied to the B-Spline by applying it to the control polygon vertices.

5. The B-Spline lies within the convex hull of its control polygon.

The knot vector used for creating B-Splines is a vector **g** containing a series of monotonically increasing real numbers $g_i \leq g_{i+1}$, e.g. [0 1 2 3 4] or [0 0.3 0.5 0.85 1]. A periodic uniform vector contains numbers equally distanced between the range $0 \leq g \leq n+k$ and it is used for creating a closed uniform B-Spline. Note that non-uniform knot vectors can also be used for B-Splines but they are not necessary for the demonstration of interactivity in SLAM.

Of particular note is that some control polygon points must be repeated in order for a B-Spline of order $k > 2$ to be fully closed. This is because the parameter range t is defined as $k-1 \leq t \leq n+1$ for an uniform knot vector. Therefore, a higher order B-Spline effectively reduces the parameter range t. For example Figure 4.9a shows what happens if a B-Spline set with $k = 4$, $n = 7$ and a uniform knot vector ranging from $0 \leq g \leq n + k$ or $0 \leq g \leq 11$ is set. The parameter range becomes $k - 1 \leq t \leq n + 1$ or $3 \leq t \leq 8$ and the curve becomes open. Figure 4.9b shows that even closing the control polygon vertices will not suffice to close the B-Spline, therefore a total of $k - 2$ repeated control polygon points must be added to the knot vector **g**. This is done at the beginning or end of the knot vector, besides the first repeated control polygon point, in this case $0 \leq g \leq 14$ which also changes the parameter range to $4 - 1 \leq t \leq 10 + 1$ or $3 \leq t \leq 11$.

Given all of this, it is now possible to introduce a more formal B-Spline definition:

$$P(s,t) = \sum_{i=1}^{n+1} B_i N_{i,k}(t) \qquad t_{min} \le t < t_{max} \qquad 2 \le k < n+1 \tag{4.13}$$

where n is a desired number of control polygon points, k is the order, $P(s,t)$ is the B-Spline with a total of M elements ($s = 0 \cdots M-1$), B_i are the control polygon points and $N_{i,k}$ are the basis functions, which can be calculated recursively using the following formulae (Cox, 1972):

$$\begin{aligned} N_{i,1}(t) &= \begin{cases} 1 & \text{if } g_i \le t < g_{i+1} \\ 0 & \text{Otherwise} \end{cases} \\ N_{i,k}(t) &= \frac{(t-g_i)N_{i,k-1}(t)}{g_{i+k-1}-g_i} + \frac{(g_{i+k}-t)N_{i+1,k-1}(t)}{g_{i+k}-g_{i+1}} \qquad k > 1 \end{aligned} \tag{4.14}$$

where g represents the elements of the knot vector, i.e. $0 \le g \le n+2k-1$ for a closed B-Spline, considering $k-2$ extra control polygon points. Lastly, in this recursion the parameter range t follows the formula:

$$t = k - 1 + \frac{n}{M}(s) \tag{4.15}$$

recalling that n is the number of control polygon points minus 1.

Taking into account Equation (4.13) its derivatives are extracted using the formulae:

$$P'(s,t) = \sum_{i=1}^{n+1} B_i N'_{i,k}(t) \quad \text{and} \quad P''(s,t) = \sum_{i=1}^{n+1} B_i N''_{i,k}(t) \tag{4.16}$$

Therefore for $P'(t)$:

$$\begin{aligned} N'_{i,1}(t) &= 0 \\ N'_{i,2}(t) &= \frac{N_{i,1}(t)}{g_{i+1}-g_i} - \frac{N_{i+1,1}(t)}{g_{i+2}-g_{i+1}} \\ N'_{i,k}(t) &= \frac{N_{i,k-1}(t) + (t-g_i)N'_{i,k-1}(t)}{g_{i+k-1}-g_i} + \frac{(g_{i+k}-t)N'_{i+1,k-1}(t) - N_{i+1,k-1}(t)}{g_{i+k}-g_{i+1}} \qquad k > 3 \end{aligned} \tag{4.17}$$

and finally for $P''(t)$:

$$\begin{aligned}
N''_{i,1}(t) &= 0\\
N''_{i,2}(t) &= 0\\
N''_{i,3}(t) &= 2\left(\frac{N'_{i,2}(t)}{g_{i+2}-g_i} - \frac{N'_{i+1,2}(t)}{g_{i+3}-g_{i+1}}\right)\\
N''_{i,k}(t) &= \frac{2N'_{i,k-1}(t) + (t-g_i)N''_{i,k-1}(t)}{g_{i+k-1}-g_i} + \frac{(g_{i+k}-t)N''_{i+1,k-1}(t) - 2N'_{i+1,k-1}(t)}{g_{i+k}-g_{i+1}} \quad k > 4
\end{aligned} \tag{4.18}$$

Using a predetermined quantity of control polygon points it is possible to calculate all the basis functions $N_{i,k}(t)$ and their derivatives $N'_{i,k}(t)$ and $N''_{i,k}(t)$ in advance. In this case the basis functions and their derivatives repeat for the extra control polygon points added for a closed B-Spline. Hence, the parameter range t is used as if it was the case of an open uniform B-Spline.

After both the basis functions and their derivatives have been calculated, they can be saved for later use as there is no need to recalculate them afterwards. However, in order to fit a B-Spline to user given points other operations must be taken into account.

4.3.2 B-Spline Curve Fitting

SLAM Interactivity is to be achieved by manipulating the initial positions of the control polygon points according to user input. Curve fitting is therefore needed in order to allow the initial B-Spline to include the user selected points. This B-Spline is later on driven by GVF forces, making it an active contour. However, it is the user that places it initially surrounding an object of interest.

The previous subsection detailed how to obtain the basis functions of a B-Spline, but how to initialise the control polygon points B_i in Equation (4.13) was not covered. Therefore, the B-Spline curve fitting process starts by using the user input points to approximate parameter values t_j. This can be done using chord lengths:

$$t_1 = 0, \quad t_j = \sum_{r=2}^{l} \frac{|D_r - D_{r-1}|}{\sum_{r=2}^{j}|D_r - D_{r-1}|} \quad 2 \leq l \leq j, \quad t_j \leq 1$$

where $|D_r - D_{r-1}|$ represents the magnitude of the (x, y) coordinates, obtained from the subtraction of one user given point D_r and the previous point D_{r-1}. Now with the approximated parameter range values t_j it is possible to use Equation (4.14) to obtain a set of basis functions $\mathbf{N}_D$. Note that these basis functions are only used to find the initial control polygon points positions, differing from the basis functions obtained using the parameter range t in Equation (4.15) for active contour evolution.

An initial B-Spline of degree n will interpolate a number j of user points, this follows Equation (4.13)*:

$$\begin{aligned}
D1_x(t_1) &= N_{D_{1,k}}(t_1)B1_{x\text{init}} + N_{D_{2,k}}(t_1)B2_{x\text{init}} + \cdots + N_{D_{n+1,k}}(t_1)B_{(n+1)_x\text{init}} \\
D2_x(t_2) &= N_{D_{1,k}}(t_2)B1_{x\text{init}} + N_{D_{2,k}}(t_2)B2_{x\text{init}} + \cdots + N_{D_{n+1,k}}(t_2)B_{(n+1)_x\text{init}} \\
&\vdots \\
D_{j_x}(t_j) &= N_{D_{1,k}}(t_j)B1_{x\text{init}} + N_{D_{2,k}}(t_j)B2_{x\text{init}} + \cdots + N_{D_{n+1,k}}(t_j)B_{(n+1)_x\text{init}}
\end{aligned}$$

as well as

$$\begin{aligned}
D1_y(t_1) &= N_{D_{1,k}}(t_1)B1_{y\text{init}} + N_{D_{2,k}}(t_1)B2_{y\text{init}} + \cdots + N_{D_{n+1,k}}(t_1)B_{(n+1)_y\text{init}} \\
D2_y(t_2) &= N_{D_{1,k}}(t_2)B1_{y\text{init}} + N_{D_{2,k}}(t_2)B2_{y\text{init}} + \cdots + N_{D_{n+1,k}}(t_2)B_{(n+1)_y\text{init}} \\
&\vdots \\
D_{j_y}(t_j) &= N_{D_{1,k}}(t_j)B1_{y\text{init}} + N_{D_{2,k}}(t_j)B2_{y\text{init}} + \cdots + N_{D_{n+1,k}}(t_j)B_{(n+1)_y\text{init}}
\end{aligned}$$

where $\mathbf{D}_{(1\cdots j)_x}$ and $\mathbf{D}_{(1\cdots j)_y}$ represent user point coordinates x and y, which depend on a specific parameter range t_j, the set of basis functions $\mathbf{N}_D$ and the initial control polygon points $\mathbf{B}_{(1\cdots n+1)_x\text{init}}$ and $\mathbf{B}_{(1\cdots n+1)_y\text{init}}$. All of this can be put in matrix form:

$$\mathbf{D}_{(1\cdots j)_x} = \mathbf{N}_D\mathbf{B}_{(1\cdots n+1)_x\text{init}} \tag{4.19}$$

$$\mathbf{D}_{(1\cdots j)_y} = \mathbf{N}_D\mathbf{B}_{(1\cdots n+1)_y\text{init}} \tag{4.20}$$

*Note that this formulation is a general representation for both open and closed B-Splines, as the latter only adds repeated control polygon points with its multiplied basis.

where

$$\mathbf{D}_{(1\cdots j)_x} = \begin{bmatrix} D_{1_x}(t_1) \\ D_{2_x}(t_2) \\ \vdots \\ D_{j_x}(t_j) \end{bmatrix} \qquad \mathbf{D}_{(1\cdots j)_y} = \begin{bmatrix} D_{1_y}(t_1) \\ D_{2_y}(t_2) \\ \vdots \\ D_{j_y}(t_j) \end{bmatrix}$$

$$\mathbf{B}_{(1\cdots n+1)_x init} = \begin{bmatrix} B_{1_x init} \\ B_{2_x init} \\ \vdots \\ B_{(n+1)_x init} \end{bmatrix} \qquad \mathbf{B}_{(1\cdots n+1)_y init} = \begin{bmatrix} B_{1_y init} \\ B_{2_y init} \\ \vdots \\ B_{(n+1)_y init} \end{bmatrix}$$

and

$$\mathbf{N}_D = \begin{bmatrix} N_{D_{1,k}}(t_1) & \cdots & \cdots & N_{D_{n+1,k}}(t_1) \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ N_{D_{1,k}}(t_j) & \cdots & \cdots & N_{D_{n+1,k}}(t_j) \end{bmatrix}$$

The matrix $\mathbf{N}_D$ is square if the condition $2 \leq k \leq n + 1 = j$ is met. However, if $2 \leq k \leq n + 1 < j$, i.e. fewer control polygon points than user given points, then the matrix $\mathbf{N}_D$ is no longer square. In order to curve fit user given points for this case, the least squares method can be used to find both x and y control polygon point coordinates:

$$\mathbf{D} = \mathbf{N}_D\mathbf{B}$$
$$\mathbf{N}_D^T\mathbf{D} = \mathbf{N}_D^T\mathbf{N}_D\mathbf{B}$$

solving for **B** yields*:

$$\mathbf{B} = \left[\mathbf{N}_D^T\mathbf{N}_D\right]^{-1}\mathbf{N}_D^T\mathbf{D} \qquad (4.21)$$

Given the initial control polygon point positions and the calculated basis functions in Equation (4.14), it is possible to use Equation (4.13) which will give a whole curve that passes through the user given points **D**.

In the SLAM context this is a novel way to employ B-Splines, as an user has

*Note that $\mathbf{N}_D^T\mathbf{N}_D$ is symmetrical, so that Cholesky decomposition can be applied which is a very fast way to calculate a matrix inverse.

the ability to initialise them over an object of interest. This is done by using points selected through an image directly seen by the user and curve fitting over this data using B-Splines. Other approaches have used B-Splines, e.g. to map riverine areas but they are constrained to environments in which the ground offers visible gradients (Pedraza et al., 2007, 2009). In this thesis the focus on B-Splines is primarily in objects rather than on the environment, as they provide valuable information for camera localisation in SLAM following its evolution and deformation due to GVF forces.

Therefore, a B-Spline needs to be evolved according to external image forces produced by objects of interest as well as internal parameters which control its localised deformations. For this their first and second derivatives found in Equation (4.16) need to be calculated, and this information will be used for its evolution as seen in the next section.

4.3.3 Snake Evolution

Section 4.3.1 considered the creation of a static B-Spline, whereas Section 4.3.2 fitted user given points to it with no evolution involved. In order to obtain tracking information GVF is assumed to drive the B-Spline, therefore becoming an active contour.

An active contour evolution can be done in several ways, a commonly found approach is its original implementation or derivatives of it (Kass and Witkin, 1987, Ivins and Porrill, 1995). However, in this investigation an approach using a tangential redistribution term is used*. The main advantages of using this method are a more robust snake evolution and avoiding instabilities that may be caused by entangling. This evolution follows (Srikrishnan and Chaudhuri, 2009):

$$\frac{\partial \mathrm{P}(\mathrm{s},\mathrm{t},\mathrm{t_e})}{\partial \mathrm{t_e}} = \alpha(\mathrm{s},\mathrm{t_e})\mathbf{T}(\mathrm{s},\mathrm{t_e}) + \beta(\mathrm{s},\mathrm{t_e})\mathbf{N}(\mathrm{s},\mathrm{t_e}) \tag{4.22}$$

where $\partial \mathrm{P}(\mathrm{s},\mathrm{t},\mathrm{t_e})$ represents the B-Spline parameters, with $\mathrm{t_e}$ being its time evolution (displacements in $\mathrm{P}(\mathrm{s},\mathrm{t})$ from Equation (4.13)), $\mathbf{T}(\mathrm{s},\mathrm{t_e})$ represents the tangential component of the curve and $\mathbf{N}(\mathrm{s},\mathrm{t_e})$ its inward normal. For ease of notation

*Original C implementation can be found at (Krishnan, 2013).

the parameter t_e will be discarded.

As such, this implementation requires the first and second B-Spline derivatives to be calculated using Equation (4.16) in Section 4.3.1. The unit normal vector **N**(s), tangent **T**(s), curvature k(s) and speed g(s) can be obtained from each of the x and y elements ($s = 0 \cdots M - 1$) in $P'(t)$ and $P''(t)$ as follows:

$$\mathbf{N}(s) = \frac{\left[-P'_{y_s}(t), P'_{x_s}(t)\right]}{\sqrt{(P'_{x_s}(t))^2 + (P'_{y_s}(t)_s)^2}} \qquad \mathbf{T}(s) = \left[\mathbf{N}_y(s), -\mathbf{N}_x(s)\right]$$

$$k(s) = \frac{P'_{x_s}(t)P''_{y_s}(t) - P'_{y_s}(t)P''_{x_s}(t)}{((P'_{x_s}(t))^2 + (P'_{y_s}(t))^2)^{\frac{3}{2}}} \qquad g(s) = \sqrt{(P'_{x_s}(t))^2 + (P'_{y_s}(t))^2}$$

A normal force $\beta(s) = \beta_{GVF}(s)$ which uses GVF is defined as:

$$\beta_{GVF}(s) = \mathbf{F}\cdot\mathbf{N}(s) + \mu k(s) = u_t(P_x(t), P_y(t))\cdot\mathbf{N}_x(s) + v_t(P_x(t), P_y(t))\cdot\mathbf{N}_y(s) + \mu k(s) \quad (4.24)$$

where $u_t(P_x(t), P_y(t))$ and $v_t(P_x(t), P_y(t))$ are the GVF output forces of the current image from Equation (4.3)*, with magnitude values extracted from locations in the GVF image given by all the M elements from the B-Spline (x and y coordinates). Note that μ here is a weighting term set very low† and β_{GVF} is a matrix containing a number of elements equal to M.

Given this, the following partial differential equation must be solved in order to find $\alpha(s)$ in Equation (4.22):

$$\frac{\partial \alpha(s)}{\partial s} = K - g(s) + g(s)k(s)\beta_{GVF}(s) \quad (4.25)$$

which can be achieved using the following discrete approximation:

$$\alpha(s+1) = \alpha(s) + K - g(s) + g(s)k(s)\beta_{GVF}(s) \quad \text{with} \quad s = 0 \cdots M - 1 \quad (4.26)$$

where K is defined as:

$$K = L - \int_0^1 k(s)\beta_{GVF}(s)g(s)\, ds \quad (4.27)$$

with L representing the length of the curve. If the values of g(s), k(s) and $\beta_{GVF}(s)$

*As this is the output GVF image the third parameter t_K is discarded for simplicity.

†This is set to 0.0001 in (Krishnan, 2013)

are neglected in comparison to L the approximation[*] K = L may be used (Srikrishnan and Chaudhuri, 2009):

$$L = \int_0^1 g(s)\, d(s) \tag{4.28}$$

Once all these values are obtained they can be plugged into Equation (4.22), iteratively obtaining B-Spline displacements $P(s, t, t_e)$. New control polygon points are calculated by setting the displacements in the matrix **D** in Equation (4.21), using the new displacement positions in the elements of the B-Spline. These are ultimately summed to the previous control polygon points, allowing to recreate a newly evolved B-Spline which is affected by GVF forces.

Whereas much of this information uses ideas from established works in the vision branch, the main novelty and importance in this investigation is its implementation in vision-SLAM. This allows to introduce interactivity within the algorithm and at the same time makes use of tracking information, thanks to real-time B-Spline deformations without severe computational cost due to accelerated GVF forces in a GPU. Therefore, the next section will show results which include a demonstration of the whole idea of interactivity in SLAM. Focusing primarily on user intervention and on-the-fly semantics.

4.4 Implementation Results

This section shows the results involving active contours and their inclusion into monocular SLAM. This is because the basic idea and the novelty of this investigation lies in the given user input, made possible though interactive snake usage. This allows the user to see and choose objects of interest, giving description or semantics in real-time. The main difference against many other approaches in SLAM is that a person is now part of the cycle, which is done for the reasons expressed in Section 2.2.4.

The algorithm was run on a computer with an Intel 4790K processor, fitted with a R9 290X GPU from AMD and the Logitech C920 camera. Processor is in charge of all the EKF related operations and snake evolution, whereas the GPU is used to accelerate image processing tasks like greyscale conversion, gradient

[*]This is equivalent to averaging the whole B-Spline P(t) in Equation (4.13).

and video-rate GVF force generation. Each EKF cycle is constrained to be of 0.08 seconds.

4.4.1 Active Contours with Gradient Vector Flow Forces

First the capabilities of the snakes surrounding objects were tested. Figure 4.10 shows two active contours already initialised by placing points randomly around two objects. Evolution is rapidly performed with the snake following the forces generated by silhouettes around the objects. However, this is not perfect since the active contours do not evenly take the shape of the objects' silhouette.

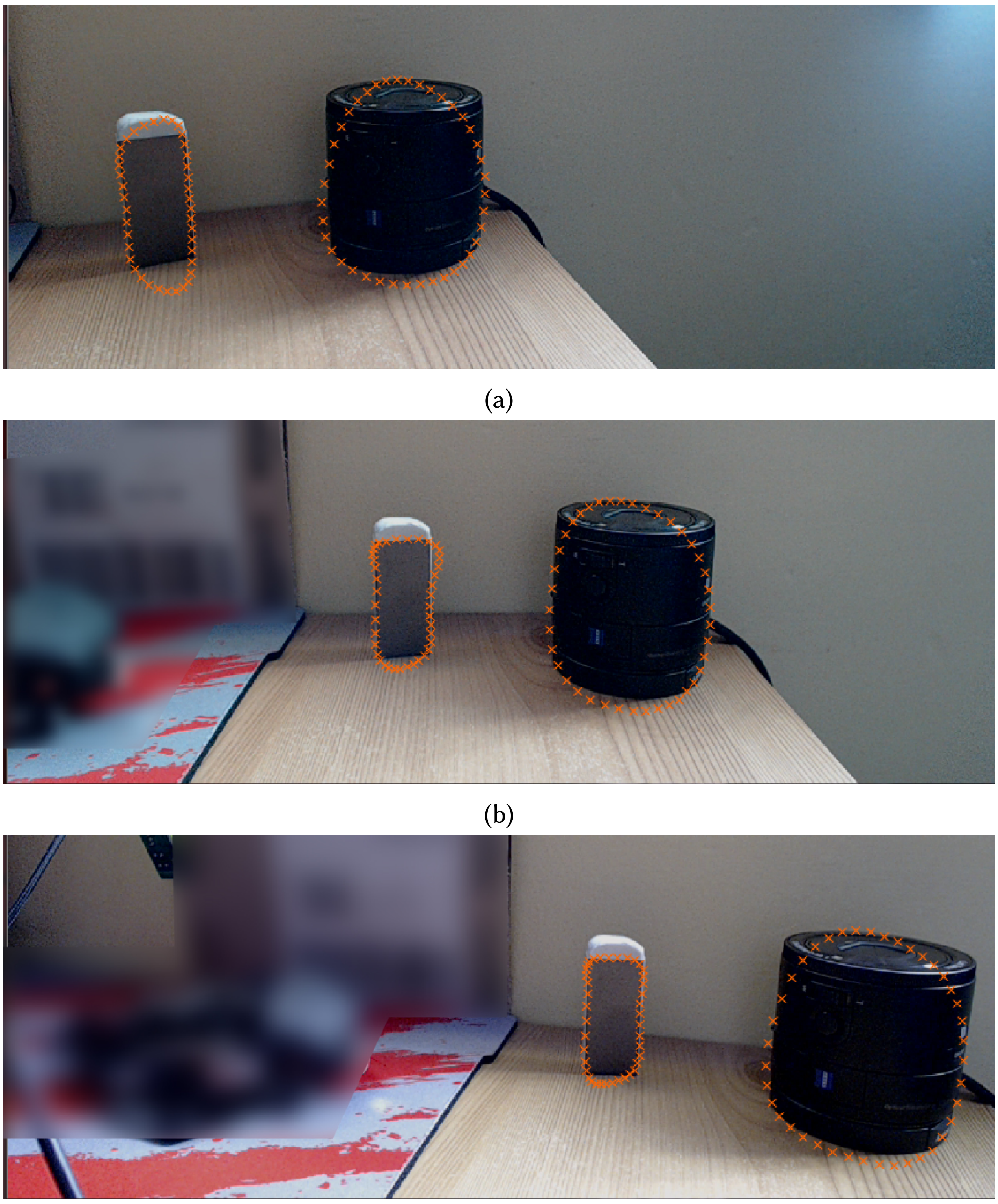
Snake object lock-on works if the movement of the camera is slow, as there is no control other than the image forces produced by GVF. Despite this, undesired behaviours from this simplistic implementation are apparent: The snake may get distracted by other forces outside the boundaries of the object of interest, since every single contour on the picture generates forces. However, objects with high contrast silhouettes diminish these caveats and can be used to perform object tracking.

Therefore, the next section demonstrates the idea of using snakes in vision-SLAM, involving B-Spline user initialisation around shapes. This includes real-time GVF forces allowing the active contour to deform fast enough, so that it becomes attached onto an object of interest and keeps track of it whilst in camera motion. This leads to obtain tracking data which can be used by the EKF, leading to feature and camera estimation as well as allowing user descriptive input.

4.4.2 EKF SLAM using Snakes, Including Interactivity

As in Section 4.4.1, the overall idea is to make use of an interface in which an user initiates a snake. The active contour will lock onto a chosen object using attractive GVF forces, generated in parallel thanks to GPGPU.

For a demonstration of interactivity within vision-SLAM simple rectangular objects are considered. Tracking is performed with an active contour (B-Spline order $k = 2$) and a total of 4 control polygon points, causing the latter to fall on



(a)

(b)

(c)

Figure 4.10: Simple active contour demonstration. Two snakes are seen following the forces produced by two objects in the image. The camera is slowly rotated from left in (a), to the middle in (b) and to the right in (c). The active contour follow the forces produced by the two objects in the image. No additional controls that could drive the snake were used.

Algorithm 4.1 Interactive vision-SLAM methodology pseudo code.

```
1:  procedure INTERACTIVE SLAM
2:      Acquire new image;
3:      Interactive user feature selection through user clicks;
4:      Initialise active contour; /* Section 4.3.2 */
5:      for i ← 0 to $N_{GVF}$ do
6:          Gradient vector flow iteration; /* from current image. Section 4.1 */
7:      end
8:      while User input do
9:          /* user decides when the snake is fully attached to the feature */
10:         Snake evolution; /* Section 4.3.3 */
11:     end
12:     while SLAM set to start do
13:         Acquire new image;
14:         for i ← 0 to N do
15:             Gradient vector flow iteration /* from current image. Section 4.1*/
16:         end
17:         /* active contour follows the feature. Section 4.3.3 */
18:         Snake evolution;
19:         if High-level feature not initialised in SLAM then
20:             Add feature to state and covariance /* Section 3.2.3 and 3.2.4 */
21:         else
22:             Obtain observations and update EKF /* Section 3.3.4 */
23:         end
24:     end
```

the corners of a selected object. This is because the B-Spline falls always within the convex hull created by the control polygon points, and for $k = 2$ this hull disappears with the B-Spline following control polygon lines (recall Figure 4.8).

In this form the object's corner coordinates can be set into the SLAM algorithm, first as new features $\mathbf{y}_{f_i}$ in Equation (3.39) and later as observations $\mathbf{z}_{f_i}$ as seen in Section 3.3.4. As the camera changes position the snake will provide tracking data to the EKF, allowing to estimate position for both object and camera. This approach is detailed in Algorithm 4.1.

The implementation works in real-time by using a source video avoiding using frames individually after each time step. It is worth mentioning that this does not consider meaningful units for mapping, as the main interest is to ensure stability within the implementation. Therefore, the EKF algorithm works using its own units. For this scenario the camera was hand-held approximately performing the following motion patterns as follows: backward translation, yaw left rotation, yaw right rotation, yaw left rotation towards centre position, backward translation, pitch tilting down, pitch tilting up, pitch tilting down towards centre position, backwards translation, yaw left rotation, yaw right rotation, yaw left rotation towards centre position, pitch tilting up, pitch tilting down, pitch tilting up towards centre position, forward translation, rolling left, rolling right, rolling left towards centre position, forward translation.

The novelty of this approach relies on interactive object selection, allowing to include input for feature description and semantics (Figures 4.11 and 4.13). In order to select a new object the user is able to stop the algorithm as needed, which then allows to mark points used for snake initialisation around the object of interest. Later on, descriptive input from the selected object can be entered. This is reflected in the generated map which now is able to display fewer features, but still retains meaningful user input (Figures 4.12, 4.14 and 4.15).

Compared to a baseline implementation of SLAM with inverse depth parametrisation as in Chapter 3, the active contours inclusion is observed to smooth feature measurements. As a result this produced more stable runs, but did not completely eradicate instabilities found in the baseline implementation. The camera translation and quaternion orientation values can be seen in Figures 4.16 and 4.17. A conversion from the quaternion states to a more readable angle in degrees can be seen in Figure 4.18, allowing to see hand-held rotations applied to the camera. This also demonstrates how a pure EKF implementation falls in performance compared to the approach using snakes presented in this chapter, as the pure EKF approach does not properly estimate all pitch rotations, failing to register the first camera up tilt and registering other yaw movements not performed. Roll seems to follow almost the same estimates in both approaches, possibly by the fact that observations present a discernible change in its location compared to pitch or yaw motions.

Uncertainty behaviour for the camera states can be seen in Figures 4.19 and 4.20, which explains in part why the EKF with active contours approach performed better. In these images it can be seen that whereas uncertainty increases rapidly in the snake approach, it tends to decrease steadily until a second object is introduced by the user. This creates an small peak in uncertainty and then continues to decrease until reaching almost zero. For the EKF only approach this uncertainty starts low, but as time progresses more uncertainty accumulates in the form of steps. This is caused sometimes when the camera is tilted or pitched in the opposite direction, as the observations are lost without recovery forcing the algorithm to obtain new ones.

With this it has been seen that the performance of EKF SLAM inverse depth parametrisation, whereas still prone to stability issues, was improved thanks to the use of active contours. This allowed to keep fewer features into the EKF and at the same time smoothed hand-held feature observation.

4.5 Concluding Remarks

Monocular SLAM aided by active contours was presented as an alternative form to improve performance in vision-SLAM. This novel attempt relies on keeping fewer but meaningful features, which are interactively selected by the user. From this approach the following conclusions are drawn:

- Snakes are a computationally expensive tool to use, as their continuous deformations depend on considerable amount of image processing in real-time. GPGPU techniques alleviate greatly the image processing tasks, as GVF is an independent operation applied over each image pixel as seen in Section 4.2. This allows fast snake deformations, which allow for a new approach in EKF SLAM that employs snakes for tracking objects.

- Snakes heavily rely on contours with their tracking performance depending on fast deformations. However, an image has potentially many gradients these might distract the snake and hamper its tracking performance as seen in Section 4.4.1. Therefore, high contrast objects are preferred to be usable in vision-SLAM aided by active contours. However, it is worth mentioning that in these situations the active contour yields stability in observations, as the snake dampens erratic hand-held camera motion avoiding the loss of observations. As a consequence uncertainty remains low compared to a baseline inverse depth parametrisation EKF SLAM approach, since there is no added features that increase uncertainty in the system.

- Semantics are an inherent part of interactivity, as they can be given right in the moment of feature selection by an user as seen in Section 4.4.2. This in-

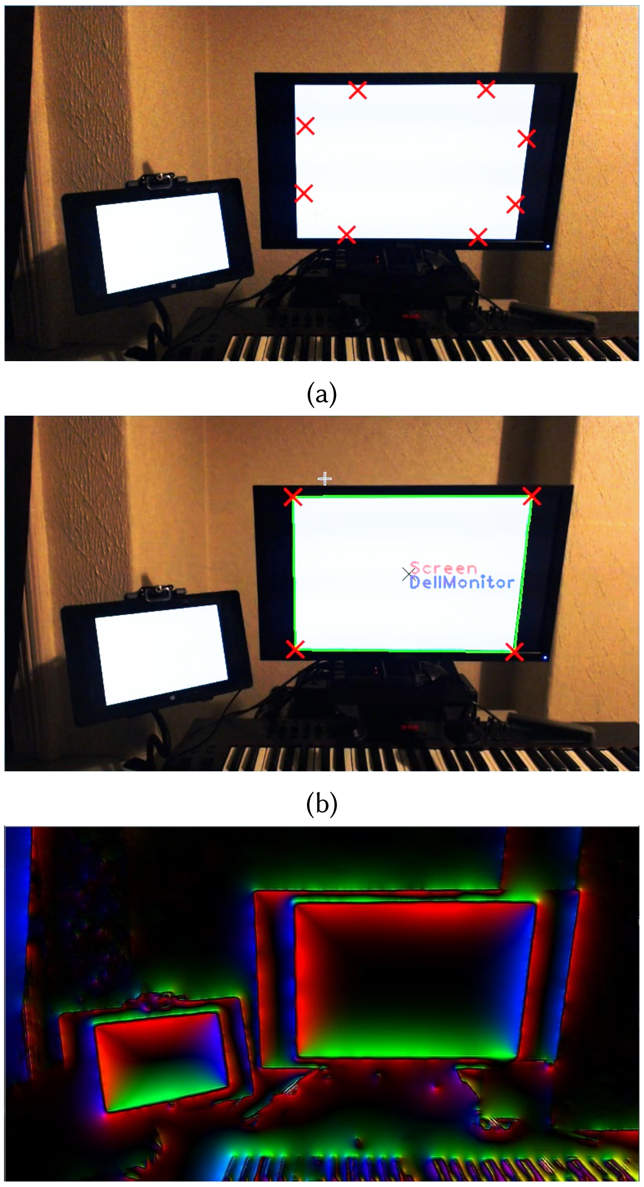
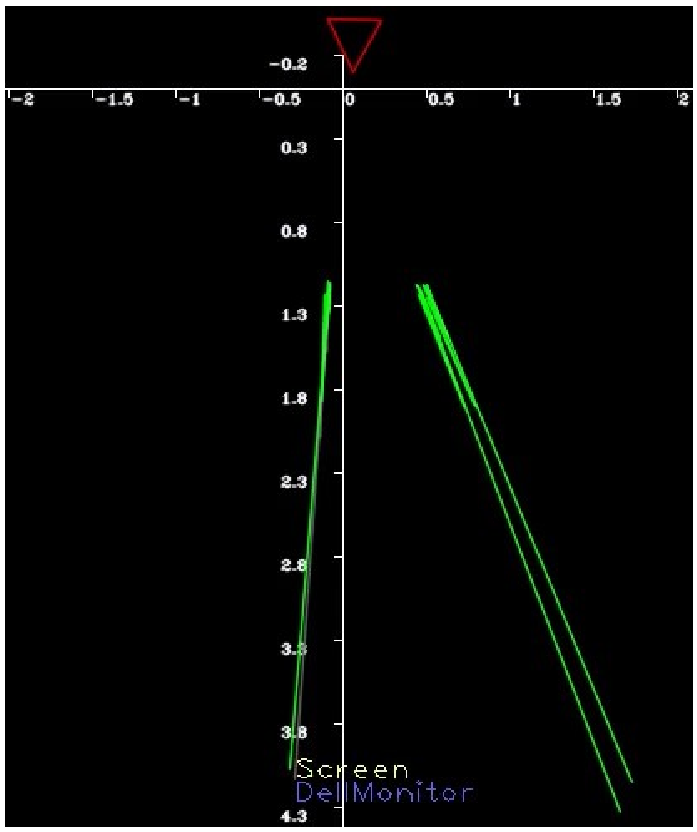
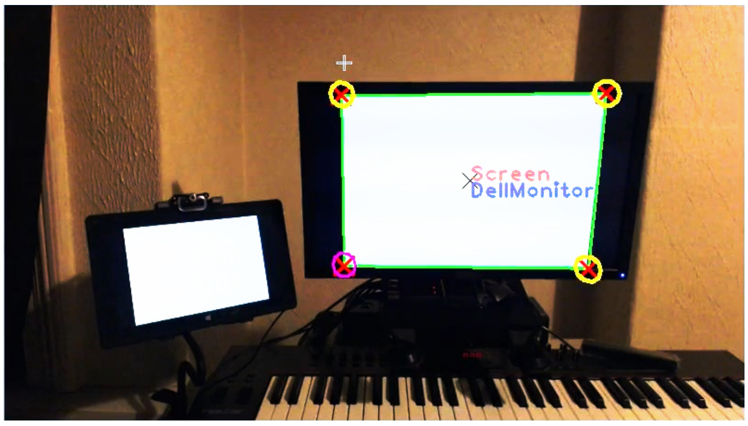


Figure 4.11: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, initial steps. An user is allowed to select a suitable object in order to perform camera localisation, such as a computer screen in (a). The user does not need to be precise in the selection of the object, as user clicks (red crosses) just need to be around a high contrast part of the image. When given the command, the active contour is initialised with k = 2, producing an snake that is limited to follow lines between the control polygon points. For this example they have been defined to a number of 4, since this would create rectangular contours that fit and lock the screens better. The user then proceeds to add description and semantics which are shown as image overlays in (b). The generated GVF forces in real-time which keep the snake locked onto the selected object are displayed in (c).

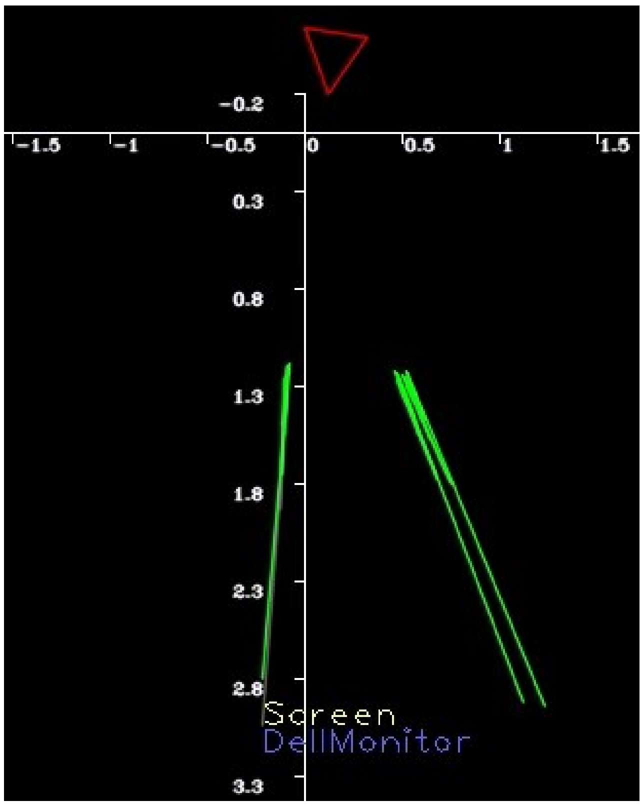


(a)

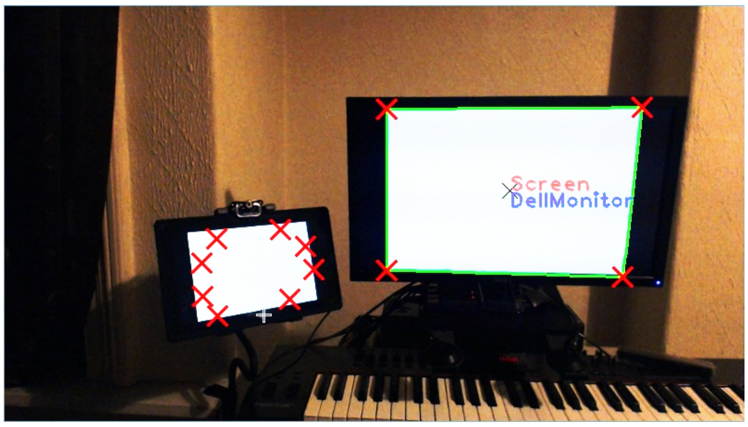


(b)

Figure 4.12: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, feature map initialisation. Initialised features are mapped in (a) with long elongated ellipses (green), as depth uncertainty is initially big and only reduced after camera movements (red triangle). These features correspond to the ones obtained from the corners of active contour shown in (b) (yellow circles). Description and semantics are also displayed in both map and in camera image. Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.

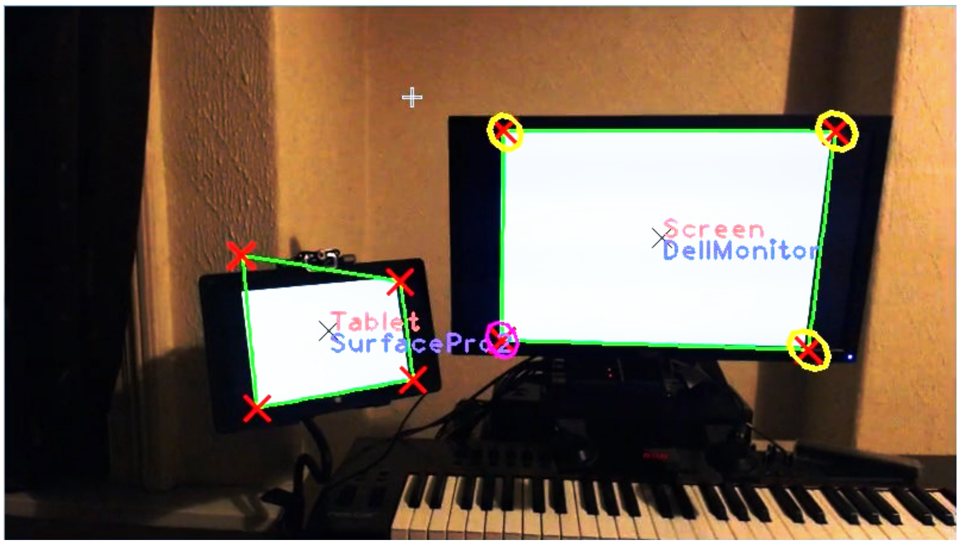


(a)

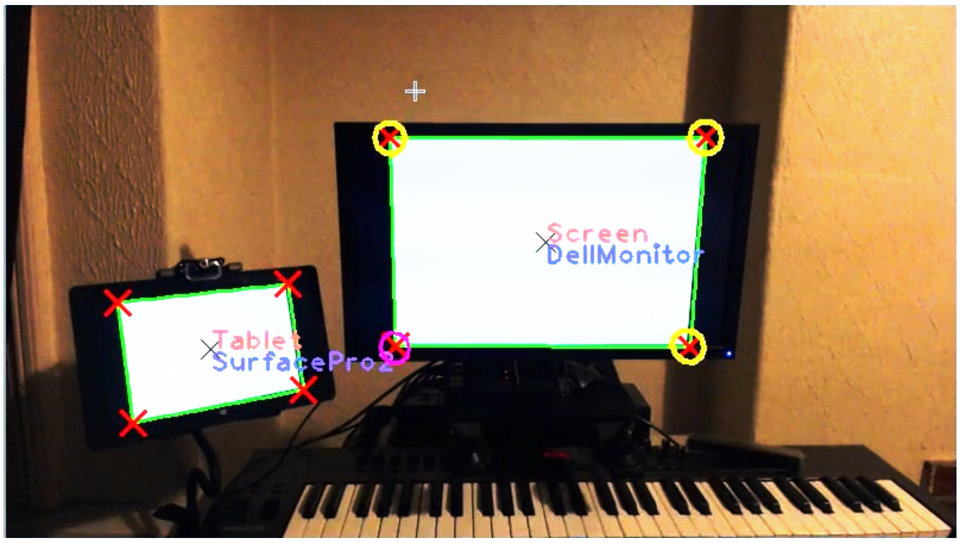


(b)

Figure 4.13: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, adding another object. The algorithm can be paused in order to select other features. The already mapped features can still be seen in (a) with reduced uncertainty ellipses (green) after camera movements (red triangle). Description and semantics remain also displayed in both map and in camera image, showing user clicks (red crosses) for the second selected object on the left (b). Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.

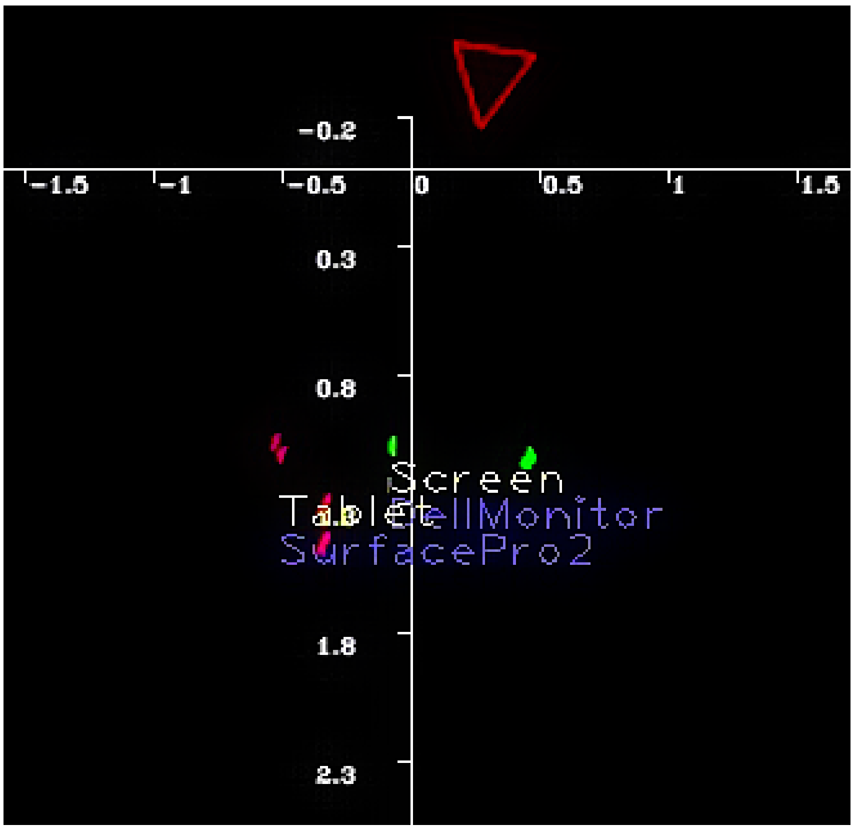


(a)

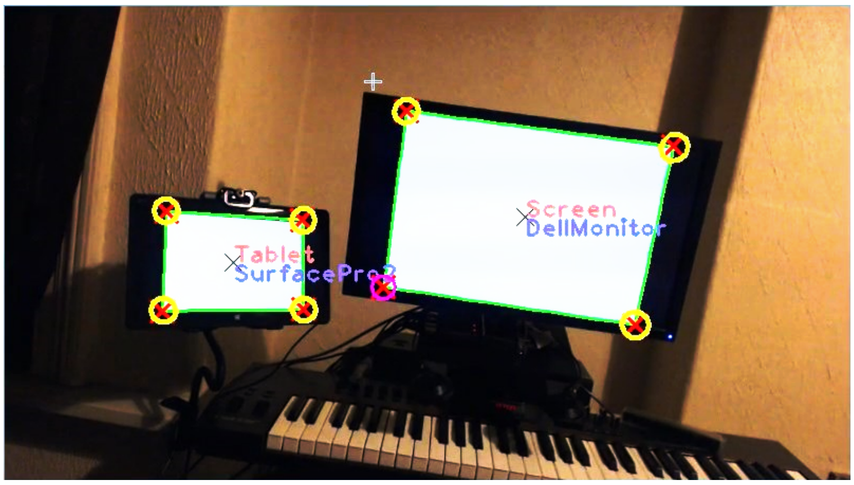


(b)

Figure 4.14: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, initialising another object. Once a snake has been set onto a second object it can be initialised on user request, as in some situations the snake may need more time to fully settle onto the silhouette (a). Description and semantics remain in camera image in (b). This information is already stored in a container within the algorithm, thus preserving user descriptive input in the map.



(a)



(b)

Figure 4.15: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, feature reduced uncertainty. Mapped features can be seen in (a) with low uncertainty for the first (green) and for the second (purple) objects after camera several movements (red triangle). Description and semantics remain also displayed in both map and in camera image, showing user clicks for the second selected object (b). Numbers in (a) are not representative of any unit as the algorithm works with its own internal units.

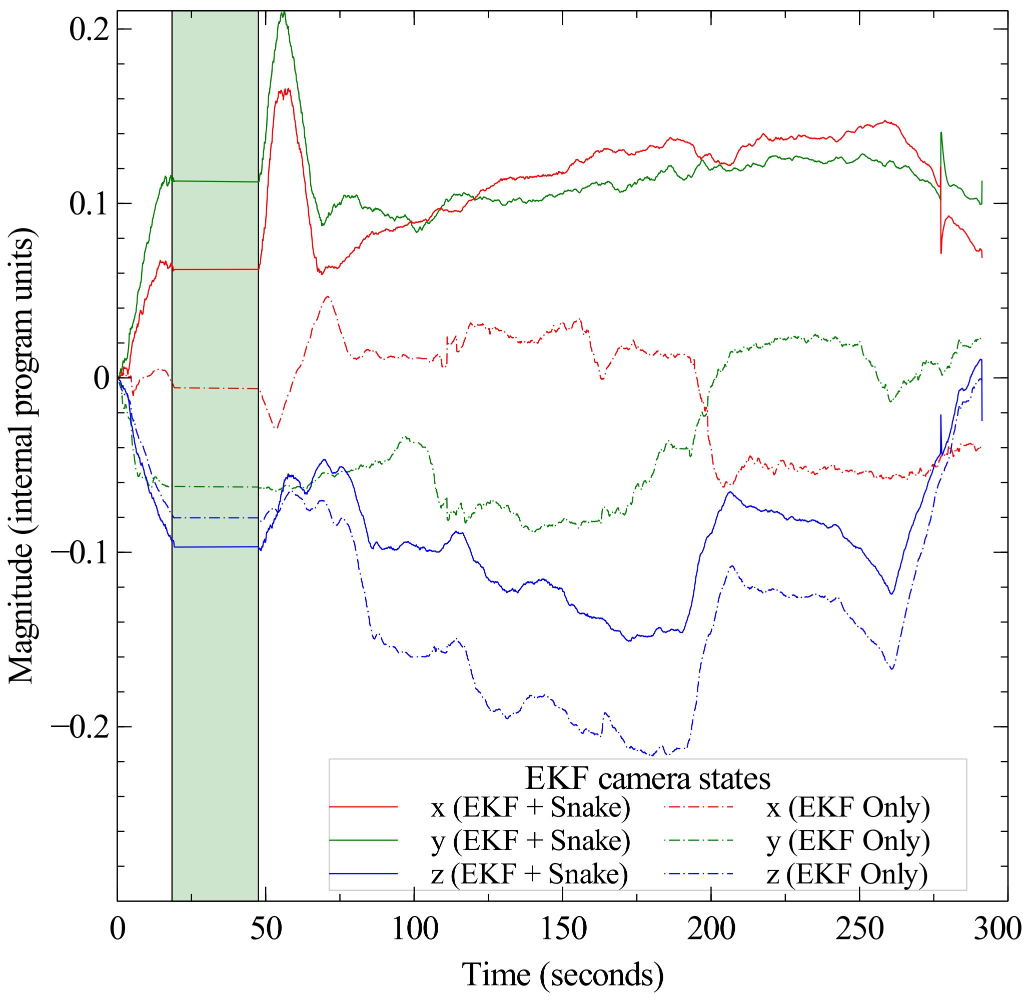


Figure 4.16: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, camera translation states. The algorithm performs a successful execution up to approximately 300 seconds. The green bar indicates the time the video was paused in order to select a second object, this can be considered as stopping the algorithm with the camera remaining still. The performance of the EKF approach aided by snakes is considered better in our test, as major displacements only occurred in the z axis. In the EKF only approach both x and y axis show translations which were not performed. Magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

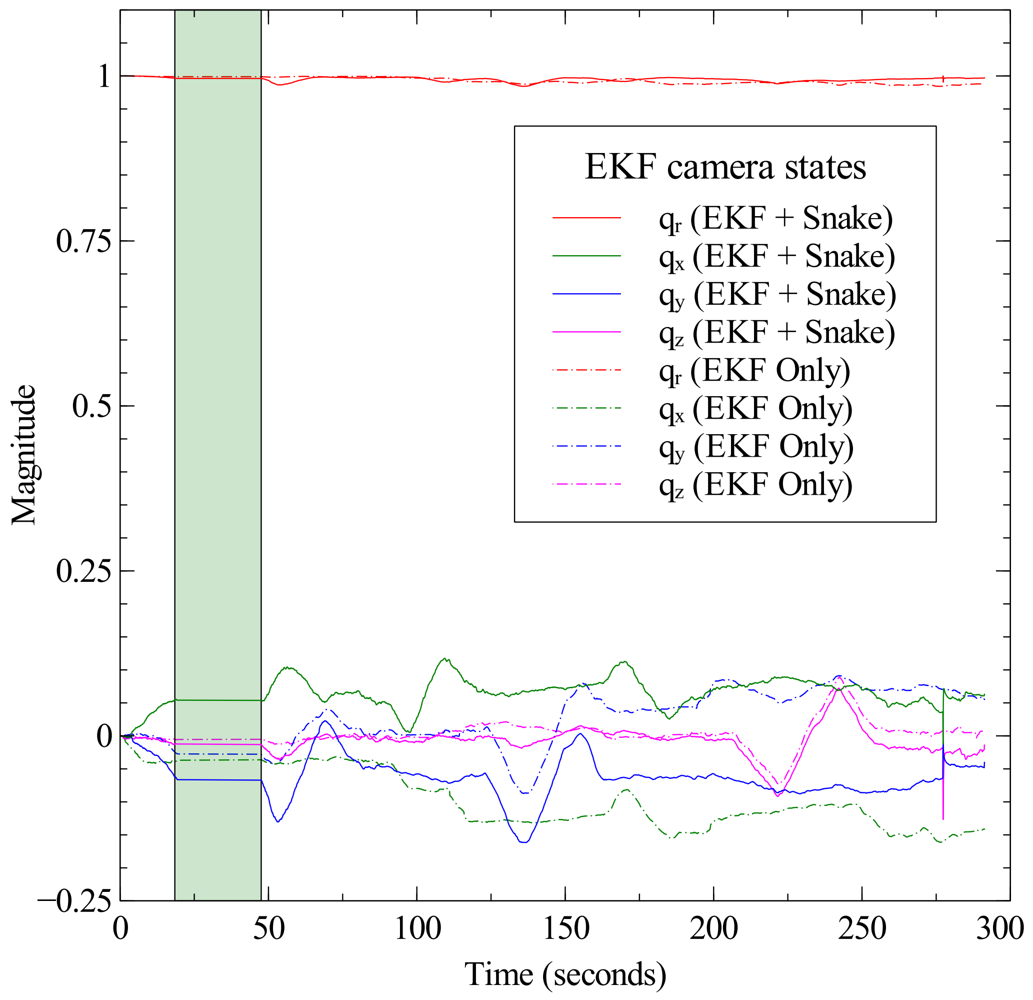


Figure 4.17: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, quaternion states. The algorithm performs a successful execution up to approximately 300 seconds. The green bar indicates the time the video was paused in order to select a second object, this can be considered as stopping the algorithm with the camera remaining still. The EKF approach aided by snakes is better considering that in the EKF only approach some of the rotations disappear as reflected in q_x. Magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

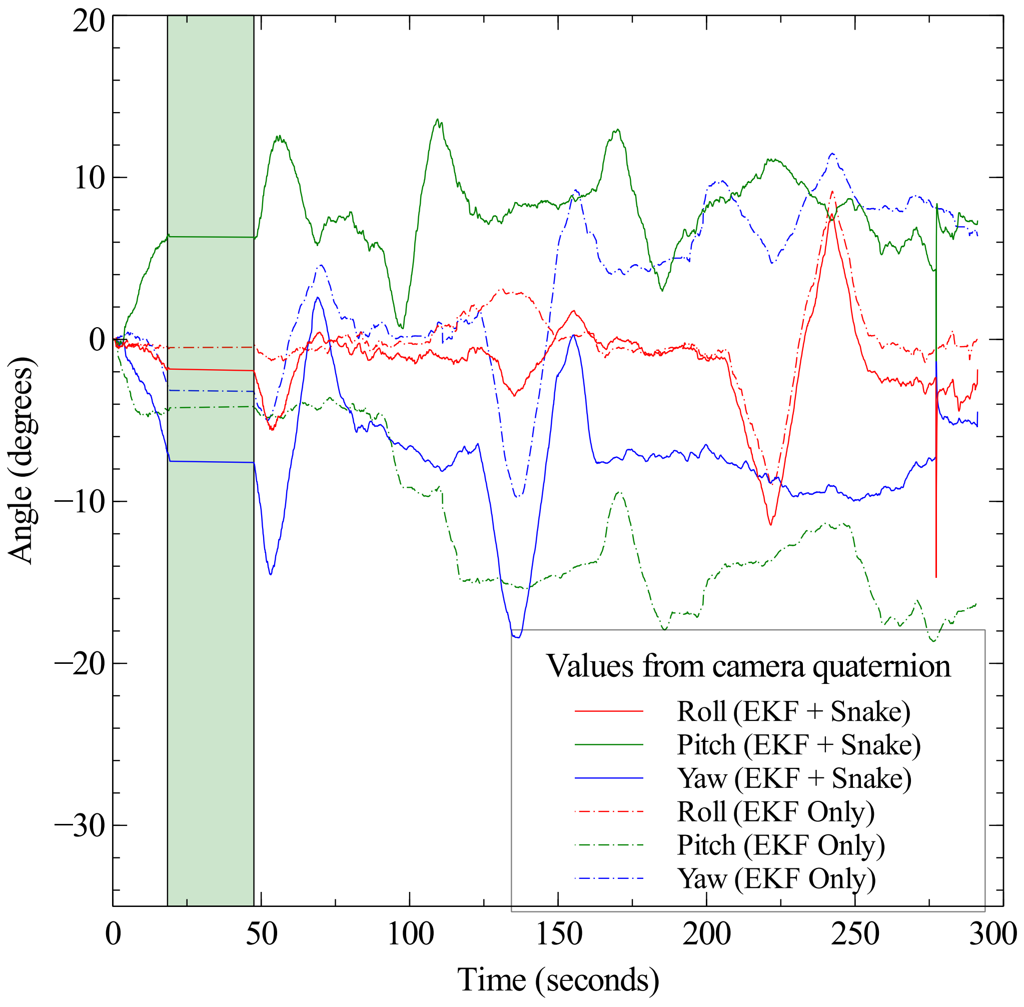


Figure 4.18: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, roll pitch and yaw from quaternion states. The values of the yaw angles follow the rotating and swinging motion used. However, pitch and roll present better estimations compared to SLAM with inverse depth parametrisation. The green bar indicates the time the video was paused in order to select a second object, this can be considered as stopping the algorithm with the camera remaining still. The EKF approach aided by snakes is better considering that in the EKF only approach some of the rotations disappear, as reflected the pitch which also drifts towards negative numbers. These values have been obtained from the quaternion states in Figure 4.17, according to (Berner, 2008).

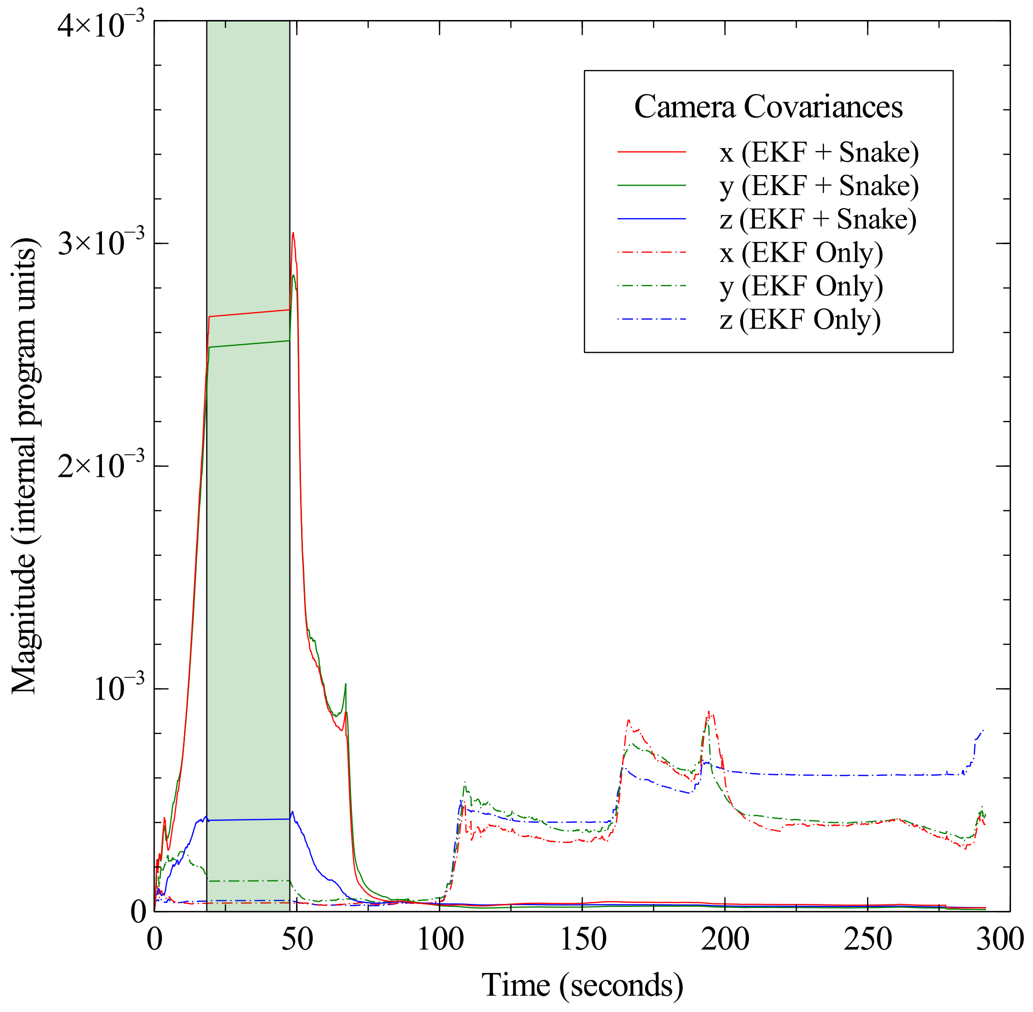


Figure 4.19: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, covariance for camera translation states. Uncertainty decreases as motion affects camera image in the EKF snake aided approach, whereas with EKF only the uncertainty increases after a while when the camera changes its rotation and looses track of the current features on screen. This forces the algorithm to obtain new features which also introduce uncertainty into the system. The green bar indicates the time the video was paused in order to select a second object, this can be considered as stopping the algorithm with the camera remaining. The EKF only approach data has been accounted for this by giving a time offset. Covariance magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

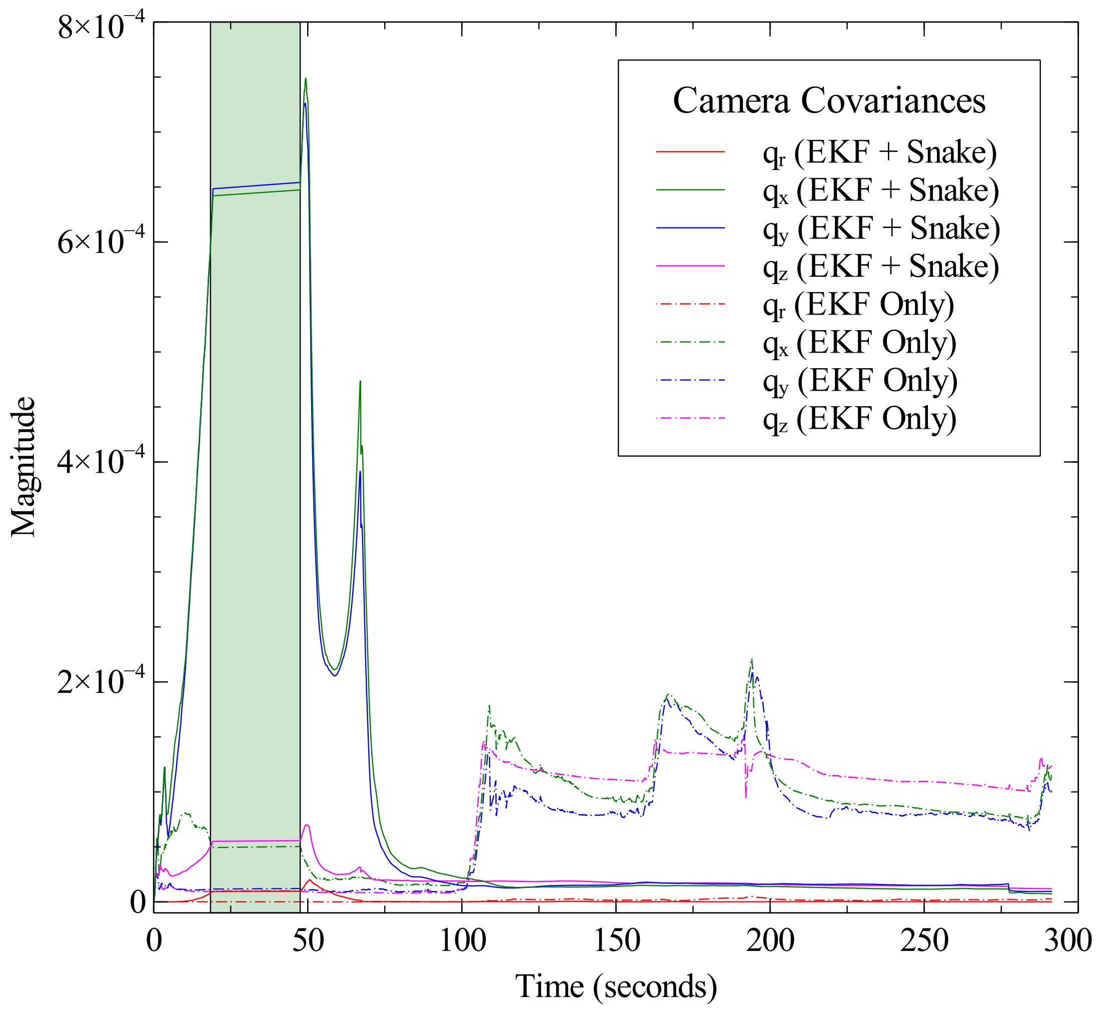


Figure 4.20: EKF SLAM with inverse depth parametrisation aided by interactivity through snakes, covariance for camera quaternion states. Uncertainty decreases as motion affects camera image in the EKF snake aided approach, whereas with EKF only the uncertainty increases after a while when the camera changes its rotation and looses track of the current features on screen. This forces the algorithm to obtain new features which also introduce uncertainty into the system. The green bar indicates the time the video was paused in order to select a second object, this can be considered as stopping the algorithm with the camera remaining still. The EKF only approach data has been accounted for this by giving a time offset. Covariance magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

formation can be displayed by means of colours or messages onto a monitor, and also can be interpreted by the robot or by other users in collaborative environments. This information is an improvement in map generation, which deviates from many other methodologies based on dense representations.

- Also important is active contour evolution, since snakes can become entangled and its tracking performance might be affected. The snake evolution seen in Section 4.3.3 offered stable active contour deformations, thanks to its dependency of the tangential term which keeps the snake steady, even after fast deformations caused by image and object changes due to camera motion.

Overall this chapter demonstrated the concept of interactivity within SLAM by means of active contours and user input, which then would track simple high contrast objects. However, the same EKF performance limitations seen in Section 3.4.2 are inherited. Despite this, stability improvements were seen by keeping fewer but meaningful features compared to a baseline implementation of SLAM with inverse depth parametrisation. Most importantly for this thesis was to introduce a new branch within SLAM, which focuses on pursuing interactivity with added benefits to localisation and mapping.

Alternatives were sought to overcome the caveats in these implementations, particularly focusing on the stability issues as longer runs would be preferred for SLAM. As such, a potential solution required to find an estimator that bypassed linearisation whilst also avoiding the need of complex matrix operations. This is as both of these introduce noise in the system, whilst also presenting numerical issues for the latter in the EKF update step as it requires a matrix inversion.

An algorithm presenting these properties goes by the name of particle filtering, which compared to EKF relies on particles representing a hypothesis (e.g. a possible camera pose). Therefore, next chapter explores this estimation technique, whilst trying to find a way of including interactivity for mapping and localisation purposes.

Chapter 5

SLAM Interactivity Through Particle Filtering

Chapter 4 introduced interactivity into SLAM, by means of using a powerful user assisted tool such as GVF guided active contours. The main challenge for this approach was fast execution, as GVF is a computationally intensive algorithm and hence the GPGPU technologies were explored for fast snake deformations.

However, this still shared some caveats with a baseline implementation of monocular SLAM using inverse depth parametrisation. Being the case that an EKF is used within the algorithm, this means that linearisation and complex matrix operations are involved which introduce uncertainty in the system. This becomes exacerbated when feature predictions are made but its corresponding observations are not found in the image, which causes the algorithm to obtain a new set of features whose new uncertainty is also added. This is mainly due to direction changes, particularly in yaw or pitch opposite camera rotations as seen in Section 4.4.2. Therefore, it can be concluded that using an EKF for vision-SLAM might not be the best approach for interactivity.

As such, this chapter presents another form of user interaction within vision-SLAM, taking a different approach in feature and camera estimation. This relies instead on a particle filtering estimator which has the advantage of avoiding linearisation, as observations are taken “as-is” based on several different camera position hypotheses. These receive an score based on how well they match an observation, i.e. the hypotheses represent a prediction of the camera pose which is expected to match an observation. Based on how well a hypothesis is matched it receives an score or weight, allowing for more particles to be replicated with slight variations for the next cycle.

Whereas there are other approaches that already employ particle filtering for SLAM (Montemerlo and Thrun, 2003, Elinas et al., 2006, Tomono, 2007, Silveira Vidal et al., 2015), the novelty in this chapter is for interactivity to influence particle weighting. This is done once again thanks to user intervention, which now is able to select lines that describe an object within the image. Observations take the form of pixels depicting an expected line, which then can be used for particle weighting.

In order to perform the weighting a novel approach involving the Hough transform is used, which is adapted to count the pixels that form a line. This is as the transform in its original implementation depends of a voting mechanism and an accumulator, from which many lines can be detected using pixels from gradients. However, this voting is prone to add invalid votes as it might include pixels generated from noise. Therefore the algorithm is adapted to count votes from lines with no discontinuities. Once line selection has been performed the user maintains control of line parameters, including its three dimensional positioning. This is as changes in perspective caused by camera motion will make initial user selection to fail matching, therefore the lines can be further on fitted to improve matching in the object of interest.

Section 5.1 provides a brief overview of the particle filter algorithm, beginning with Monte Carlo approximations in Section 5.1.2. Later on resampling is explored in Section 5.1.3. These include the general equations, which can be used as an initial framework for particle filtering, allowing to use different kinds of observations to estimate non-linear systems.

Next, Section 5.2 discusses the Hough transform and its importance in obtaining lines. Discussing first its equations in Section 5.2.1 and later on its accumulator in Section 5.2.2.

Once a method for detecting prominent lines has been devised, novel ways to implement them into the particle filtering weighting is sought in Section 5.3, exploring both a simple approach in Section 5.3.1 as well as a new and improved approach which yields better prominent line detection in Section 5.3.2. This is complemented with interactive adjustments made over the lines as initial guesses in depth become erroneous after camera motion in Section 5.3.3.

Later on feature initialisation and results are shown in Section 5.4 for a novel interactive SLAM implementation involving particle filtering, as well as an improved and adapted Hough line detection. Finally, the concluding remarks for this chapter are presented in Section 5.5.

5.1 Monte Carlo Approximations

Monte Carlo approximations are useful when estimating systems with many variables. They are often employed when other methods prove to be inadequate, such as when linearisation techniques do not yield a confident enough estimate in non-linear systems. Camera localisation alone in Vision-SLAM already has many of these properties, e.g. it possesses 3 translation and 4 quaternion orientation variables together with its non-linear motion model, thus the complexity of its EKF implementation in Chapter 3.

EKF is based on Bayesian inference which is an approach in which prior information $p(\mathbf{x})$ is conditioned with observations $\mathbf{z}_1, \cdots, \mathbf{z}_T$, yielding a posterior of $\mathbf{x}$ which is $p(\mathbf{x}|\mathbf{z}_{1:T})$ (Johnson, 2010). This is later on approximated to a Gaussian distribution with mean $\mathbf{m}$ and covariance $\mathbf{P}$ or $\mathbf{N}(\mathbf{x}|\mathbf{m}, \mathbf{P})$ as shown in Section 3.1.1. This formulation can also be rewritten in order to obtain the expectations from a posterior distribution (Särkkä, 2013):

$$\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] = \int \mathbf{g}(\mathbf{x})p(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x} \tag{5.1}$$

where $\mathbf{g} : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is an arbitrary function. However, this integral can be evaluated only on special cases and often not analytically. Note, however, that there is an equivalence to $p(\mathbf{x}|\mathbf{z}_{1:T})$ which will be seen later.

Therefore, Monte Carlo approximations are tools that allow to compute the expectation integral in Equation (5.1) by using a total of 1 to N samples $\mathbf{x}^{(i)}$ from the distribution and then obtaining estimates based on averaging (Särkkä et al., n.d., Särkkä, 2013):

$$\begin{aligned} i &= 1, \cdots, N \\ \mathbf{x}^{(i)} &\sim p(\mathbf{x}|\mathbf{z}_{1:T}) \quad (5.2) \\ \mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] &\approx \frac{1}{N}\sum_{i} \mathbf{g}(\mathbf{x}^{(i)}) \quad (5.3) \end{aligned}$$

which is guaranteed to converge by the central limit theorem (Lapeyre, 2007, Särkkä, 2013). One of the main advantages of this method is that its error term defined as

$$\epsilon_N = \mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] - \frac{1}{N}\sum_{i} \mathbf{g}(\mathbf{x}^{(i)})$$

is invariant with respect to the dimensions of $\mathbf{x}$. This makes it superior to many other methods where $\mathbf{x}$ has multiple dimensions, including the EKF used in Chapters 3 and 4 which considers a system with growing dimensionality per feature acquired as well as camera location in Equation (3.74).

Therefore, The following subsections proceed to understand the inner workings in Monte Carlo methods and sampling techniques, allowing to adapt particle filtering to an interactive SLAM approach using lines.

5.1.1 Importance Sampling for obtaining Particles

A distribution $p(\mathbf{x}|\mathbf{z}_{1:T})$ sometimes does not allow for a straightforward way to obtain samples as in Equation (5.2). This is as $p(\mathbf{x}|\mathbf{z}_{1:T})$ might contain regions that possess very small values or might be a tail within the distribution, as a consequence samples $\mathbf{x}^{(i)}$ might fail lie in these regions with low values. Therefore, sampling in this situation can deliver an incorrect Monte Carlo approximation.

In these situations an importance distribution $\pi(\mathbf{x}|\mathbf{z}_{1:T})$ can be used, allowing to sample from it and thus overweighting or giving *importance* to a region presenting these low values:

$$\begin{aligned} i &= 1, \cdots, N \\ \mathbf{x}^{(i)} &\sim \pi(\mathbf{x}|\mathbf{z}_{1:T}) \quad (5.4) \end{aligned}$$

This needs to be accounted for in Equation (5.1), by adding and removing the importance distribution within the integral as follows:

$$\begin{aligned}\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] &= \int \mathbf{g}(\mathbf{x})p(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x} \\ &= \int \left[\frac{\mathbf{g}(\mathbf{x})p(\mathbf{x}|\mathbf{z}_{1:T})}{\pi(\mathbf{x}|\mathbf{z}_{1:T})}\right]\pi(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x}\end{aligned}$$

The estimates can be obtained through averaging as in Equation (5.3), but now this also accounts for the importance distribution:

$$\begin{aligned}\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] &\approx \frac{1}{N}\sum_i \left[\frac{p(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}{\pi(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}\right]\mathbf{g}(\mathbf{x}^{(i)}) \qquad (5.5)\\ &\approx \sum_i \tilde{w}^{(i)}\mathbf{g}(\mathbf{x}^{(i)}) \\ \tilde{w}^{(i)} &= \frac{1}{N}\frac{p(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}{\pi(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})} \qquad (5.6)\end{aligned}$$

Nonetheless, as Equation (5.4) is not the original distribution in Equation (5.2), the former has to have an associated weight for correction as given by Equation (5.6).

All of this so far assumes that the samples from a posterior can be directly evaluated, i.e. $p(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})$. When this is not possible Bayes' rule can be used, which allows to rewrite the sampling of a posterior as follows:

$$p(\mathbf{x}^{(i)}|\mathbf{z}_{1:T}) = \frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{\int p(\mathbf{z}_{1:T}|\mathbf{x})p(\mathbf{x})d\mathbf{x}} \qquad (5.7)$$

The numerator in Equation (5.7) can often be evaluated directly, but the normalisation constant (the denominator) cannot. However, importance sampling can still be applied after using Bayes' rule. First, $p(\mathbf{x}|\mathbf{z}_{1:T})$ in Equation (5.1) is put into this form:

$$\begin{aligned}\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] &= \int \mathbf{g}(\mathbf{x})p(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x} \\ &= \frac{\int \mathbf{g}(\mathbf{x})p(\mathbf{z}_{1:T}|\mathbf{x})p(\mathbf{x})d\mathbf{x}}{\int p(\mathbf{z}_{1:T}|\mathbf{x})p(\mathbf{x})d\mathbf{x}}\end{aligned}$$

Then importance sampling can be applied to both the numerator and denominator:

$$\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] = \frac{\int \left[\frac{p(\mathbf{z}_{1:T}|\mathbf{x})p(\mathbf{x})}{\pi(\mathbf{x}|\mathbf{z}_{1:T})}\mathbf{g}(\mathbf{x})\right]\pi(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x}}{\int \left[\frac{p(\mathbf{z}_{1:T}|\mathbf{x})p(\mathbf{x})}{\pi(\mathbf{x}|\mathbf{z}_{1:T})}\right]\pi(\mathbf{x}|\mathbf{z}_{1:T})d\mathbf{x}}$$

Which can now be approximated using averaging as in Equation (5.5):

$$\begin{aligned}\mathbf{E}[\mathbf{g}(\mathbf{x})|\mathbf{z}_{1:T}] &\approx \frac{\frac{1}{N}\sum_{i=1}^{N}\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{\pi(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}\mathbf{g}(\mathbf{x}^{(i)})}{\frac{1}{N}\sum_{j=1}^{N}\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(j)})p(\mathbf{x}^{(j)})}{\pi(\mathbf{x}^{(j)}|\mathbf{z}_{1:T})}} \\ &= \sum_{i=1}^{N}\left[\frac{\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{\pi(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}}{\sum_{j=1}^{N}\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(j)})p(\mathbf{x}^{(j)})}{\pi(\mathbf{x}^{(j)}|\mathbf{z}_{1:T})}}\right]\mathbf{g}(\mathbf{x}^{(i)}) \qquad (5.8) \\ w^{(i)} &= \frac{\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(i)})p(\mathbf{x}^{(i)})}{\pi(\mathbf{x}^{(i)}|\mathbf{z}_{1:T})}}{\sum_{j=1}^{N}\frac{p(\mathbf{z}_{1:T}|\mathbf{x}^{(j)})p(\mathbf{x}^{(j)})}{\pi(\mathbf{x}^{(j)}|\mathbf{z}_{1:T})}} \qquad (5.9)\end{aligned}$$

where the denominator in Equation (5.9) can be seen as a cumulative sum of the numerator, thus effectively normalising Equation (5.8).

This method of importance sampling considers all measurements and states, which might be detrimental in some scenarios. Therefore, the next section improves over this approach by adapting it to a recursive implementation, allowing to consider only information obtained from the previous cycle.

5.1.2 Sequential Importance Sampling

Section 5.1 considered a case where the sampling could contain all the observations from 1 to T. In some cases this might be a problem, e.g. when there is limited computational resources. However, Sequential Importance Sampling (SIS) can be used for distributions of the form:

$$\mathbf{x}_k \sim p(\mathbf{x}_k|\mathbf{x}_{k-1})$$
$$\mathbf{z}_k \sim p(\mathbf{z}_k|\mathbf{x}_k)$$

where $\mathbf{x}_k \in \mathbb{R}^n$ is the state and $\mathbf{z}_k \in \mathbb{R}^m$ are the observations at time step k. Therefore, the approximation of the arbitrary function $\mathbf{g}(\cdot)$ is performed at each step k using weights $w_k^{(i)}$ and particles $\mathbf{x}_k^{(i)}$:

$$\begin{aligned} i &= 1, \cdots, N \\ \mathbf{E}[\mathbf{g}(\mathbf{x}_k)|\mathbf{z}_{1:k}] &\approx \sum_i w_k^{(i)} \mathbf{g}(\mathbf{x}_k^{(i)}) \end{aligned} \tag{5.10}$$

It is possible to obtain recursion for Equation (5.10) by using an equivalence to a posterior probability density approximation* $p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$ (Särkkä, 2013):

$$\begin{aligned} \mathbf{E}[\mathbf{g}(\mathbf{x}_k)|\mathbf{z}_{1:k}] &\approx \sum_i w_k^{(i)} \mathbf{g}(\mathbf{x}_k^{(i)}) \\ p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) &\approx \sum_i w_k^{(i)} \delta(\mathbf{x}_k - \mathbf{x}_k^{(i)}) \end{aligned} \tag{5.11}$$

Therefore, the Bayes' rule can be used to rewrite $p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$:

$$\begin{aligned} p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) &= \frac{p(\mathbf{z}_{1:k}|\mathbf{x}_{0:k})p(\mathbf{x}_{0:k})}{p(\mathbf{z}_{1:k})} \quad \text{Bayes' rule} \\ &= \frac{p(z_k|\mathbf{z}_{1:k-1}, \mathbf{x}_{0:k})p(\mathbf{z}_{1:k-1}|\mathbf{x}_{0:k})p(\mathbf{x}_k|\mathbf{x}_{0:k-1})p(\mathbf{x}_{0:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})p(\mathbf{z}_{1:k-1})} \end{aligned} \tag{5.12}$$

Equation (5.12) can be simplified by applying Bayes' rule again, considering the likelihood $p(\mathbf{z}_{1:k-1}|\mathbf{x}_{0:k})$ and using Markov properties. This is as the last state in $\mathbf{x}_{0:k}$ will have no effect on the observations $\mathbf{z}_{1:k-1}$:

$$p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1}) = \frac{p(\mathbf{z}_{1:k-1}|\mathbf{x}_{0:k})p(\mathbf{x}_{0:k-1})}{p(\mathbf{z}_{1:k-1})} = \frac{p(\mathbf{z}_{1:k-1}|\mathbf{x}_{0:k-1})p(\mathbf{x}_{0:k-1})}{p(\mathbf{z}_{1:k-1})}$$

*Note that the argument $\mathbf{x}_{0:k}$ is in the discrete domain, but for purposes of this demonstration this notation is not needed and thus left away (Süzen, 2014).

Therefore, the terms in Equation (5.12) are rearranged allowing to simplify:

$$\begin{aligned} p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) &= \frac{p(z_k|\mathbf{z}_{1:k-1},\mathbf{x}_{0:k})p(\mathbf{x}_k|\mathbf{x}_{0:k-1})p(\mathbf{z}_{1:k-1}|\mathbf{x}_{0:k})p(\mathbf{x}_{0:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})p(\mathbf{z}_{1:k-1})} \\ &= \frac{p(z_k|\mathbf{z}_{1:k-1},\mathbf{x}_{0:k})p(\mathbf{x}_k|\mathbf{x}_{0:k-1})p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \end{aligned} \tag{5.13}$$

Finally, under Markov assumptions the state $\mathbf{x}_k$ only depends on $\mathbf{x}_{k-1}$ and the observations $\mathbf{z}$ are independent between themselves in Equation (5.13). Therefore:

$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k)}$$

A proportional approximation can be obtained if the normalisation term is removed:

$$p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) \propto p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1}) \tag{5.14}$$

Taking a similar approach to Equation (5.6), it is possible to sample from an importance distribution $\mathbf{x}_{0:k}^{(i)} \sim \pi(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$, which together with Equation (5.14) gives the importance weights:

$$w_k^{(i)} \propto \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})p(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})}{\pi(\mathbf{x}_{0:k}^{(i)}|\mathbf{z}_{1:k})}$$

Then the importance distribution can be decomposed as:

$$\pi(\mathbf{x}_{0:k}|\mathbf{z}_{1:k}) = \pi(\mathbf{x}_k|\mathbf{x}_{0:k-1},\mathbf{z}_{1:k})\pi(\mathbf{x}_{0:k-1}|\mathbf{z}_{1:k-1})$$

which leads to:

$$w_k^{(i)} \propto \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})p(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})}{\pi(\mathbf{x}_k^{(i)}|\mathbf{x}_{0:k-1}^{(i)},\mathbf{z}_{1:k})\pi(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})} \tag{5.15}$$

However, the last terms in numerator and denominator demonstrate to be the weight at the previous step, i.e. $w_{k-1}^{(i)}$:

$$w_{k-1}^{(i)} = \frac{p(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})}{\pi(\mathbf{x}_{0:k-1}^{(i)}|\mathbf{z}_{1:k-1})}$$

Therefore, Equation (5.15) becomes:

$$w_k^{(i)} \propto \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})}{\pi(\mathbf{x}_k^{(i)}|\mathbf{x}_{0:k-1}^{(i)}, \mathbf{z}_{1:k})} w_{k-1}^{(i)} \tag{5.16}$$

Given Equation (5.16), an interesting result can be obtained if the importance function is arbitrarily selected as the dynamic model, i.e. $\pi(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{z}_{1:k}) = p(\mathbf{x}_k|\mathbf{x}_{k-1})$:

$$w_k^{(i)} \propto \frac{p(\mathbf{z}_k|\mathbf{x}_k^{(i)})p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})}{p(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)})} w_{k-1}^{(i)}$$

Finally, simplifying terms:

$$w_k^{(i)} \propto p(\mathbf{z}_k|\mathbf{x}_k^{(i)}) w_{k-1}^{(i)} \tag{5.17}$$

These weights now can be used to obtain $\mathbf{E}[\mathbf{g}(\mathbf{x}_k)|\mathbf{z}_{1:k}]$ in Equation (5.10) or its equivalent $p(\mathbf{x}_{0:k}|\mathbf{z}_{1:k})$ in Equation (5.11) , noting that only the measurement $\mathbf{z}_k$ is needed compared to $\mathbf{z}_{1:T}$ in Equation (5.9). However, using the dynamic model $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ might require a high number of samples for proper estimation (Särkkä, 2013).

In this recursive form the algorithm is prone to particle degeneracy. This means that eventually the weights will be too small or close to zero, rendering this approach unusable. In the next section some improvements are explored over this implementation are made to mitigate this negative effect.

5.1.3 Resampling and the Bootstrap Filter

Particle degeneracy is solved through resampling. This allows to discard particles with low weight associated to them, whilst replicating higher weighted particles.

There are different ways to perform this technique, with examples including multinomial, systematic and residual systematic resampling (Sileshi et al., 2013a). Both multinomial and systematic resampling are considered here as they are related. This is because while other approaches may offer better performance, research shows that the gains from other resampling techniques might not be that big whereas their implementation increases in complexity (Douc and Cappé, 2005, Hol et al., 2006).

A commonly found implementation of multinomial resampling is known as Ripley's method, which if programmed in sufficiently low level is considered really fast. Nevertheless, an improved version that avoids programming loops can also be used to increase performance (Gustafsson, 2010).

A multinomial resampling can therefore be obtained in the following way: first the summation of all the weights is calculated from Equation (5.17):

$$i = 1, \cdots, N$$
$$s_i = \sum_i w_k^{(i)}$$

then the weights are normalised, so that all of them sum to 1:

$$w_{k_n}^{(i)} = \frac{w_k^{(i)}}{s_i}, \qquad \sum_i w_{k_n}^{(i)} = 1$$

Subsequently an array is created, containing both the weight indices and normalised summed values:

$$\begin{bmatrix} 1 & 2 & \cdots & N-1 & N \\ w_k^{(1)} & w_k^{(1)} + w_k^{(2)} & \cdots & w_k^{(1)} + w_k^{(2)} + \cdots + w_k^{(N-1)} & w_k^{(1)} + w_k^{(2)} + \cdots + w_k^{(N-1)} + w_k^{(N)} \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 2 & \cdots & N-1 & N \\ w_k^{(1)} & w_k^{(1)} + w_k^{(2)} & \cdots & w_k^{(1)} + w_k^{(2)} + \cdots + w_k^{(N-1)} & 1 \end{bmatrix} \tag{5.18}$$

Another array is created similar to Equation (5.18) with indices going from N+1 to M, where M is the desired number of resampled particles. However, the main difference lies in which the weights values for this array are randomly obtained

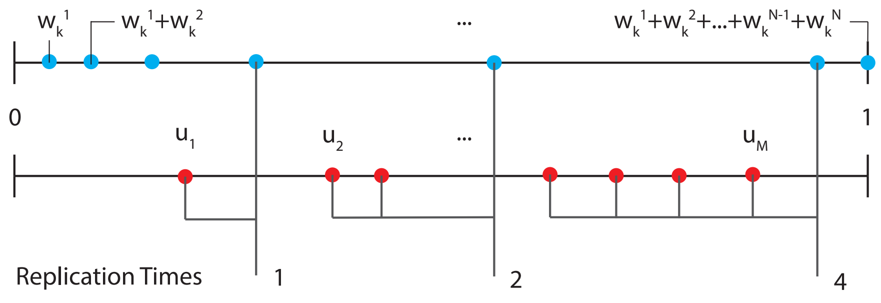


Figure 5.1: Particle resampling. The blue circles represent the normalised weight array in Equation (5.18), whereas the red circles represent the weights obtained by an uniform distribution $u \sim U(0,1)$ in Equation (5.19) or by using a predefined number $u_1 \sim U(0,1)$ in Equation (5.20). After concatenation and ordering of their indexes, the red circles (weights from polynomial or systematic resampling) will fall in between the blue circles (particle weights). Resampling of the immediate particle represented by a blue circle depends on the number of red circles in between itself and the previous particle. Note that any particle that does not have any red circles preceding it is simply discarded.

from a uniform distribution which ranges from 0 to 1, i.e. $u \sim U(0,1)$:

$$\begin{bmatrix} N+1 & N+2 & \cdots & N+M-1 & N+M \\ u_1 \sim U(0,1) & u_2 \sim U(0,1) & \cdots & u_{M-1} \sim U(0,1) & u_M \sim U(0,1) \end{bmatrix} \tag{5.19}$$

Next both Equations (5.18) and (5.19) are concatenated and sorted according to their weight index number. After the concatenation and ordering of the entire resulting array according to their weight, there will be indices greater than N (up to M) which are in between the indices below or equal than N. Therefore, when there are two weights values with indices below or equal than N whose distance between themselves is big, there will be more weight values falling in between these which possess indices greater than N (up to M).

The remaining step is to count how many indices greater than N (up to M) are in between the ones lower than N, which will indicate how many times to resample a particle. The resampled particles will be the ones with immediate next index lower or equal to N, which will be replicated as many times as the count of the indices greater than N (up to M) in between as illustrated in Figure 5.1.

It is worth mentioning that the resampled particles M can vary or remain constant in Equation (5.19). In both cases the weights of the resampled particles are reset, with each particle taking a new weight value of $\frac{1}{M}$. The bootstrap filter takes Equation (5.17) and uses the multinomial resampling each cycle, allowing for an straight forward implementation (Gong et al., 2012, Särkkä, 2013).

Multinomial resampling presents itself as one of the easiest methods to implement. However, at the same time it allows for slight tweaks in it to produce systematic resampling which in turn increases its performance (Douc and Cappé, 2005, Hol et al., 2006, Sileshi et al., 2013b). Therefore, instead of obtaining weight values from an uniform distribution $u \sim U(0, 1)$ as in multinomial resampling, only a random number $u_1 \sim U(0, 1)$ is obtained from a similar distribution for systematic resampling. Later on this number u_1 and the indices M are used, replacing the weight values in Equation (5.19):

$$\begin{bmatrix} N+1 & N+2 & \cdots & N+M-1 & N+M \\ u_1 & u_1 + \frac{1}{M} & \cdots & u_1 + \frac{M-2}{M} & u_1 + \frac{M-1}{M} \end{bmatrix} \tag{5.20}$$

with the rest of the process remaining almost the same as the multinomial approach, which now concatenates (5.18) and (5.20). Sorting is performed and in its resulting array the indices greater than N (up to M) are detected and counted, resampling its corresponding particle as in Figure 5.1.

All of this leads to the bootstrap filter, also known as the particle filter algorithm, which after defining a total number of N particles can be summarised as follows:

- In the beginning of each time step or iteration k, the weights $w_k^{(i)}$ are all initialised with a value of $1/N$.
- The particles $\mathbf{x}_k^{(i)}$ are generated according to $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. For the purposes of camera localisation, each particle $\mathbf{x}_k^{(i)}$ is generated from a previous particle $\mathbf{x}_{k-1}^{(i)}$ which has been perturbed by short linear and angular displacements. These differ in magnitude as they are caused by randomly generated velocities multiplied by a short time interval.
- For each particle $\mathbf{x}_k^{(i)}$ its weight $w_k^{(i)}$ is obtained following Equation (5.17), which uses the likelihood $p(\mathbf{z}_k|\mathbf{x}_k)$.
- Resampling is performed, adding all the weights $w_k^{(i)}$ and then normalising them so that their total sum is equal to 1. Afterwards the array in Equation

(5.18) is formed together with the array from Equation (5.20). This allows to perform systematic resampling as in Figure 5.1 after reordering terms using the weight values. This marks also the start of a new cycle.

So far information pertaining particle filtering has been gathered here, as an introduction to this estimation technique. Remarked importance is given to two main concepts: weighting and the likelihood $p(\mathbf{z}_k|\mathbf{x}_k)$. This is as the two of them can be linked and related to user input, thanks to the flexibility in hypothesis generation for particle filtering. Therefore, a method that allows to obtain both particle weights and include interaction with features is explored. This method is able to extract lines from images, serving also as hypothesis to which weights can be assigned.

5.2 The Hough Transform

Edge extraction from images has huge interest in image processing, since they are often found in man made structures whilst possessing good distinction from other characteristics in objects. As such, an automated line detection algorithm is preferred capable of discerning points that belong to an edge or a line avoiding image noise.

A known algorithm within vision for extracting lines is the Hough transform, which can also be considered as a hypothesis testing methodology. Hence, it allows to discern from lines based on the votes given by edges on the image. This algorithm takes votes into a two dimensional accumulator, in which each bin tells how much an hypothesis of a line is supported (Herout et al., 2013). This voting concept conveniently presents similarities with weighting in particle filtering, thus allowing to use this idea in line detection and for estimation purposes.

There are other methods for detecting lines, such as PCLines (Dubská et al., 2011, Markéta, 2011). However, they still make use of an accumulator and votes as well of a threshold or clipping plane to detect lines. This is done in order to filter stray lines from those that are prominent (enough votes). However, the Hough transform is a well understood method for detecting lines allowing for an easier implementation. Therefore, the next sections explore the Hough transform equations and their application into an user enabled SLAM approach using particle filtering.

5.2.1 Hough Lines

The principle of the Hough transform starts by considering pixels in an image depicting a line or edge. Later on, voting is performed in which the pixels from this line will cast votes over a particular Hough accumulator bin. The bins represent all the possible lines within the image and the amount of votes dictates how many pixels are part of the line. The line represented by the accumulator bin is described in Hough terms, which represent a line that transverses the entire image, overlaying all the pixels that were part of the votes in this particular bin.

The Hough transform formula is then defined as follows:

$$x_H \cos \theta_H + y_H \sin \theta_H = \rho_H \tag{5.21}$$

where θ_H is the angle formed from the x-axis towards a line from the centre $(0_H, 0_H)$ in the direction of a point (x_H, y_H) which is perpendicular to the line that needs to be described, whilst ρ_H is the Euclidean distance from the centre $(0_H, 0_H)$ towards the same point (x_H, y_H) (Figure 5.2).

In order to introduce interactivity in this process an user can be allowed to overlay a line over the edge of an object, by selecting the start and end locations within the image. However, Equation (5.21) cannot be directly used as the parameters ρ_H and θ_H need to be obtained according to the line generated through the two user selected points.

Therefore, the following example is considered to obtain a Hough transform with two user selected points $p1_H$ and $p2_H$. These points are extracted from an image that has been corrected from lens deformations as in Section 3.2.2. An offset is later applied to the absolute coordinates h_{u_x} and h_{u_y} described by the user points, being half of the total horizontal and vertical resolution respectively (Figure 5.3):

$$\begin{aligned} x_H &= h_{u_x} - \frac{\text{Horizontal Image Resolution}}{2} \\ y_H &= h_{u_y} - \frac{\text{Vertical Image Resolution}}{2} \end{aligned} \tag{5.22}$$

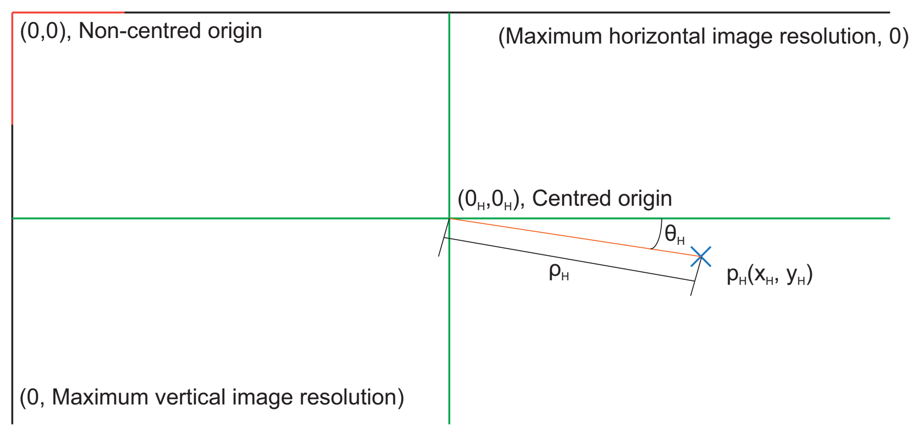

(0,0), Non-centred origin
(Maximum horizontal image resolution, 0)
(0H,0H), Centred origin
θH
ρH
pH(xH, yH)
(0, Maximum vertical image resolution)

Figure 5.2: Hough transform parameters. In this approach a magnitude ρ_H and an angle θ_H mark a point p_H. The line represented by the Hough transform is perpendicular to the point p_H. Assuming that these parameters contain a value, only a single line perpendicular to the obtained point p_H can be described within the image.

Once the coordinates x_H and y_H have been defined for both of the points $p1_H$ and $p2_H$, a line can be obtained in terms of the Hough transform. The idea of this is to find a common angle θ_H by assuming an equal distance ρ_H, using Equation (5.21) for each of the two points $p1_H$ and $p2_H$ each one with coordinates $(x1_H, y1_H)$ and $(x2_H, y2_H)$ respectively:

$$x1_H \cos\theta_H + y1_H \sin\theta_H = \rho_H \quad \text{First point} \tag{5.23}$$

$$x2_H \cos\theta_H + y2_H \sin\theta_H = \rho_H \quad \text{Second point} \tag{5.24}$$

As ρ_H is assumed the same the expressions become equal to each other:

$$x1_H \cos\theta_H + y1_H \sin\theta_H = x2_H \cos\theta_H + y2_H \sin\theta_H$$

$$(x1_H - x2_H) \cos\theta_H = -(y1_H - y2_H) \sin\theta_H$$

Therefore, solving for θ_H will yield a common angle between the two points:

$$\theta_H = \arctan\left(\frac{-x1_H - x2_H}{y1_H - y2_H}\right) \tag{5.25}$$

Finally, the obtained θ_H can be used in any of Equations (5.23) or (5.24) to yield a

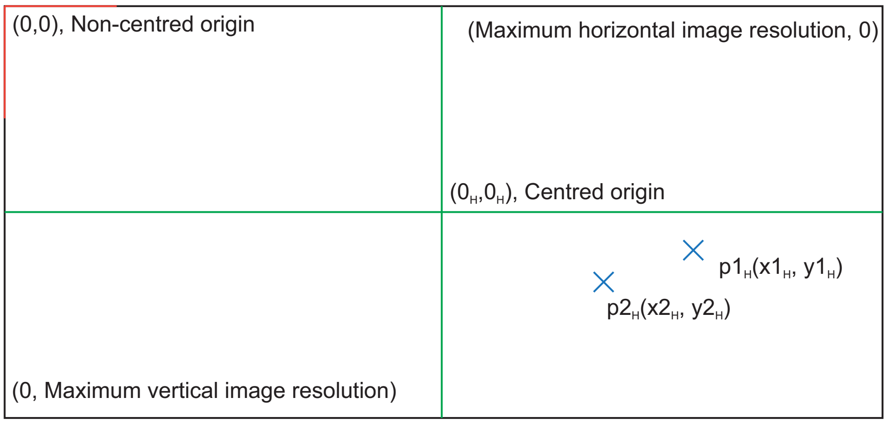

(0,0), Non-centred origin
(Maximum horizontal image resolution, 0)
(0H,0H), Centred origin
p1H(x1H, y1H)
p2H(x2H, y2H)
(0, Maximum vertical image resolution)

Figure 5.3: Two points with centred coordinates. Both $p1_H$ and $p2_H$ are points referenced to the centre of the image, which are obtained by substracting half of the horizontal and vertical image resolution to the user selected point coordinates h_{u_x} and h_{u_y} respectively. This effectively changes their zero reference from the upper left corner of the image towards its centre.

common ρ_H for the two points. The representation of the Hough imaginary line passing through the points $p1_H$ and $p2_H$ can be seen in Figure 5.4.

From the previous example it can be inferred that many lines are generated when considering an image containing many objects, as only two points or pixels are required to generate a line. Therefore, the next section explores the Hough accumulator as it allows to discern prominent lines within an image.

5.2.2 The Hough Accumulator

The Hough transform can make use of a binary image obtained as in Section 4.1.2, allowing to detect predominant lines by means of voting. The votes are stored in bins, each one representing a particular combination of ρ_H and θ_H. All these combinations and their respective bins form the Hough accumulator.

For example, the x-axis of the accumulator represents ρ_H, which can go in a negative distance from the centre to the upper left corner of the image, and in positive distance from the centre to the lower right corner of the image; the y-axis of the accumulator represents θ_H which goes from zero to 180 degrees*.

*For performance reasons this is usually from 0 to 180 degrees, but it can also be from 0 to 360

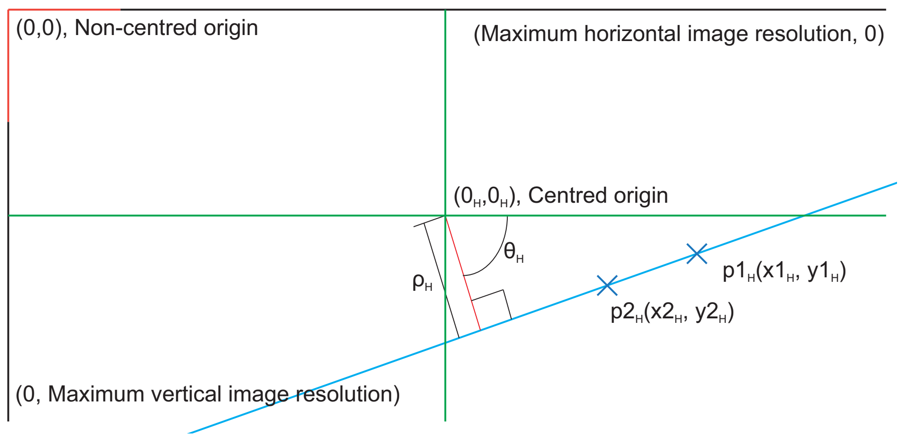

(0,0), Non-centred origin
(Maximum horizontal image resolution, 0)
(0_H,0_H), Centred origin
θ_H
ρ_H
$p1_H(x1_H, y1_H)$
$p2_H(x2_H, y2_H)$
(0, Maximum vertical image resolution)

Figure 5.4: Hough transform for two points. A line (blue) described in Hough terms travels across two defined points $p1_H$ and $p2_H$. This is done by finding a common angle θ_H and distance ρ_H for the two points using Equation (5.21), e.g. using Equations (5.23) and (5.24), with both θ_H and ρ_H becoming the Hough representation of the line.

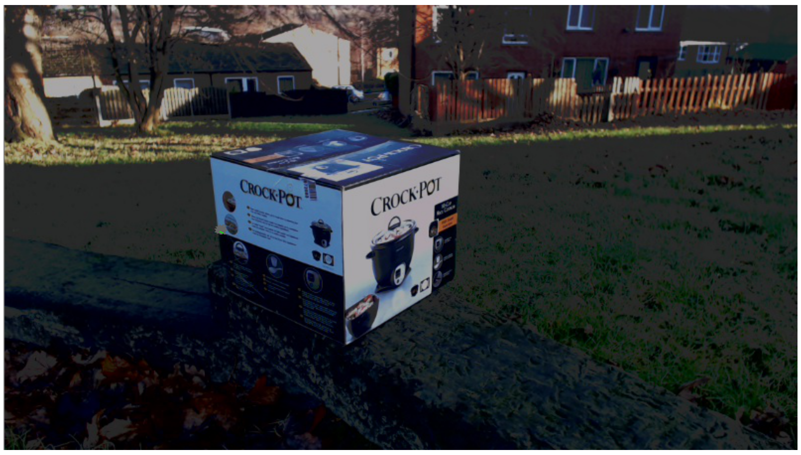
An advantage of using an accumulator is that predominant lines can also be extracted by setting a minimum vote threshold. This would filter stray lines produced by pixels scattered across the image, yielding only lines whose ρ_H and θ_H combinations exceed a number of votes as shown in Figure 5.5.

However, this voting might be considered a moderately heavy computation for a CPU depending on image size. This is because the distance from the image centre is computed for each pixel, by varying each step from 0 to 180 degrees in θ_H from Equation (5.21).

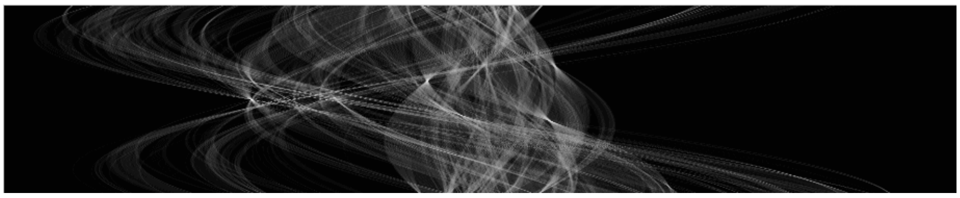
Nevertheless, the inherent individual pixel operations of the Hough Transform allow for GPGPU implementation in a similar fashion to GVF as follows: First, a binary image is obtained from the source video feed as in Section 4.1.2, then the distortion produced by the lens across the whole image is removed. This is done in order to obtain straight lines, using an inverse mapping method* as proposed in (Qureshi, 2011). Finally, for each one of the pixels of the binary image proceed to obtain votes through the variation of θ_H.

degrees with ρ_H being completely positive over the x-axis in the accumulator, i.e. from 0 to the top corner of the image in distance. Note however that if the former case is used, the sign of ρ_H might need to be changed, also an offset must be added to θ_H.

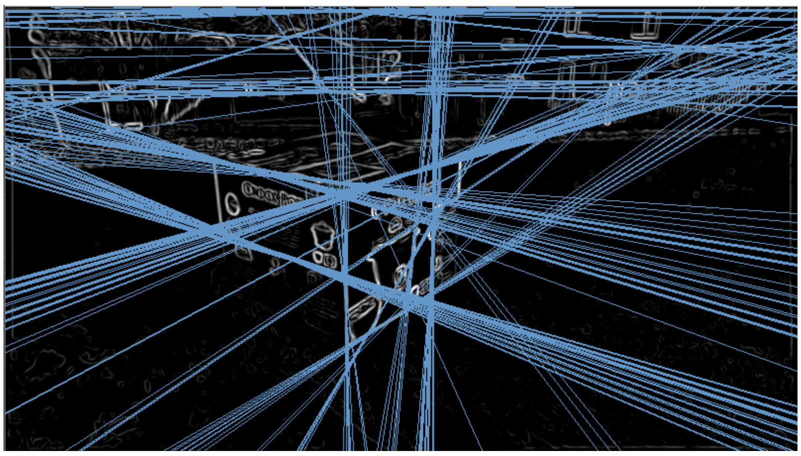
*This is how Civera's code was found to have inverted distorted to undistorted steps (*http://openslam.org/*).



(a)



(b)



(c)

Figure 5.5: Hough transform (pixel threshold 150). A Hough transform is applied to (a), which then produces the Hough accumulator in (b). The latter shows in white the bins that contain votes from those pixels whose intensity is above 150, with the brighter parts containing the most votes. Any other pixels falling below the intensity threshold are discarded in the voting. Note that the height of the accumulator represents θ_H which goes from zero to 180 degrees, hence the sinusoidal shaped lines. The lines generated from the combinations of ρ_H and θ_H in the accumulator (b) are further filtered, so that only those which their number of votes exceed 50 are shown in (c).

This follows the OpenCL Hough transform implementation which goes through all the pixels in a binary image as follows (Konrad, 2014): if the intensity is above zero the pixel belongs to a gradient and thus possibly to a line. If the pixel belongs to a gradient an offset is given to its coordinates using Equation (5.22), which change its reference from the upper left part of the image towards the centre. Vote casting is then performed by taking these new coordinates in Equation (5.21), then varying θ_{H} from 0 to 180 with step increments of 1. The accumulator is increased by 1 vote for each ρ_{H} obtained with the current θ_{H} step.

However, at the same time it is mentioned that for this implementation small values of ρ_{H} were filtered to avoid straight lines in the image centre (Konrad, 2014). Using the proposed code revealed a vertical black bar in the middle part of the Hough accumulator, with the width of the bar varying according to the threshold in small values of ρ_{H}. This was solved by using a float data type instead of an integer one for the values of ρ_{H} and rounding afterwards, which produces an accumulator similar to that of Figure 5.5b.

Now that a fast implementation of the Hough transform is obtained, the resulting information from its accumulator can be used for estimation purposes. Therefore, the next section explores how to use the obtained votes describing combinations of ρ_{H} and θ_{H} in particle filtering, which then represent lines over the image.

5.3 Coupling the Hough Transform with the Particle Filter

The Hough transform discussed in Section 5.2.1 allows to extract lines from an image, by means of storing votes in an accumulator. Previous to this Section 5.1.3 described a recursive implementation of particle filtering. This estimator is able to rely on a likelihood, particle (hypothesis) weights and resampling in order to continuously yield estimates. Therefore, similarities between Hough transform voting and particle weighting begin to arise. This is as votes can be considered a form of weighting, which can be used on hypothesis that consist of lines. This section looks into this relationship, unifying these two concepts in order to attain camera position estimation through image lines.

One of the main ideas in this research was inspired by real-time camera tracking using a particle filter (Pupilli and Calway, 2005). In this investigation particles represent different camera positions, using local image patches which are expected to be found within a sequence of images. These patches are used to build a likelihood allowing to cast votes, which then find their way into weighting. A hypothesis or particle of the camera position is most *likely* when there is a successful patch correlation between the stored image templates and the observed patches; it is also less likely when the correlation fails in some or all of the templates.

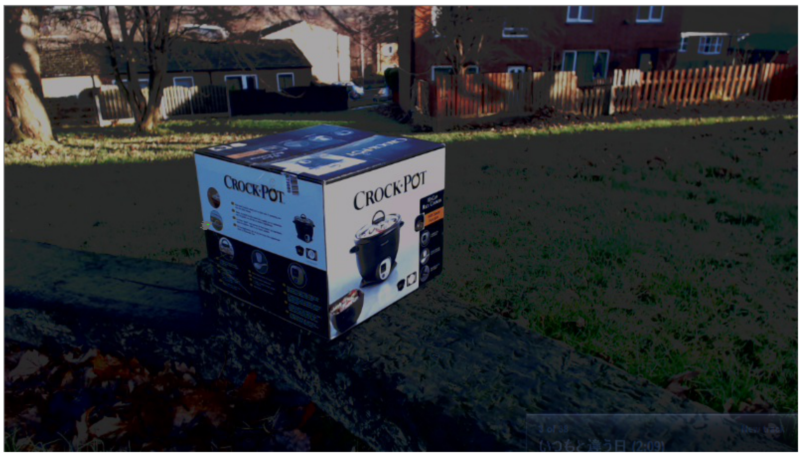
Using image templates for correlation with observed patches suggests great flexibility in selecting a likelihood for the camera position, as other voting methods besides template correlation can be used as long as they allow for weighting. Therefore, the Hough transform offers an alternative as its accumulator is already a voting mechanism for lines detected over an image.

However, a full Hough transform of an image would not be of much use, as an image can deliver many gradients in different parts of it. These are not necessarily congregated over an object and as a result, the Hough accumulator contains votes that are not part of edges but from scattered pixels across the image. As a result the following sections consider different ways to use the Hough Transform with an user approach, in a way to provide weighting for particle filtering.

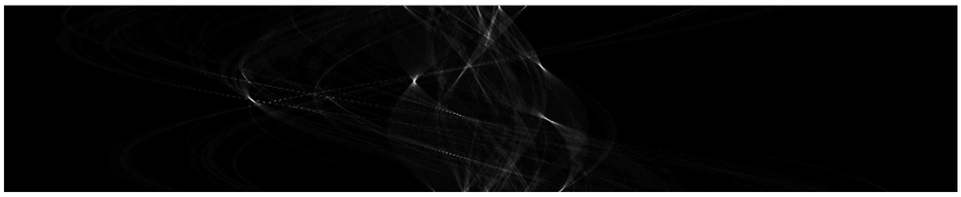
5.3.1 Simple Hough Likelihood

A first simple approach for a Hough likelihood might involve selecting a threshold for pixel intensity in the Hough transform. This assumes that the most prominent lines will possess gradients whose pixels have high values of intensity (Figure 5.6). However, repetitive line detection might be affected by motion blur, as the intensity of pixels in edges decrease in value due to image blurriness.

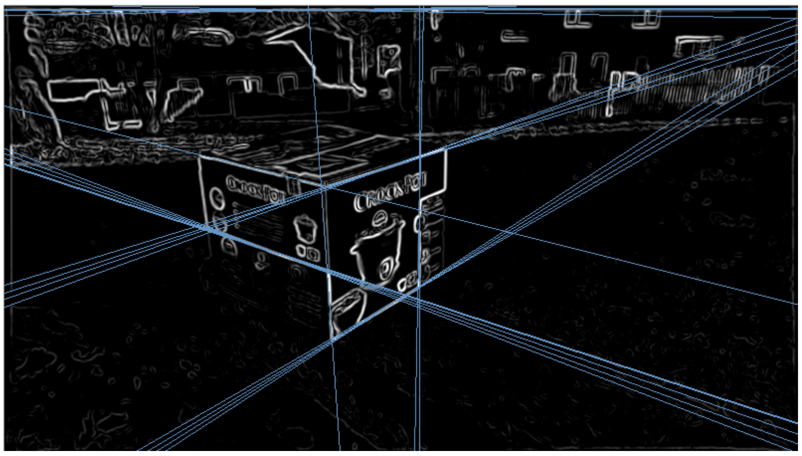
In order to avoid a strong pixel intensity threshold, another approach is to limit the region surrounding a feature. This might be selected by the user through the coordinates $(x1_{\text{Region}}, y1_{\text{Region}})$ and $(x2_{\text{Region}}, y2_{\text{Region}})$, forming a rectangular region surrounding an object of interest. From there the image can be cropped and later its Hough transform can be obtained, or the region coordinates can be directly



(a)



(b)



(c)

Figure 5.6: Hough transform (pixel threshold 200). A Hough transform is applied to (a), which then produces the Hough accumulator in (b). The latter shows in white the bins that contain votes from those pixels whose intensity is above of 200, with the brighter parts containing the most votes. Any other pixels falling below the intensity threshold are discarded in the voting. Note that the height of the accumulator represents θ_{H} which goes from zero to 180 degrees, hence the sinusoidal shaped lines. The lines generated from the combinations of ρ_{H} and θ_{H} in the accumulator (b) are further filtered, so that only those which their number of votes exceed 50 are shown in (c). Compared to Figure 5.5 the amount of lines is reduced, as shown by the accumulator in (b) and the obtained lines in (c).

given as parameters in a Hough transform GPGPU implementation, allowing to skip any operation outside of the given region.

However, limiting a region does not ensure that only predominant lines will be found. This is because the selected object of interest does not always possess a texture-less surface, with only a handful of well defined gradients in it.

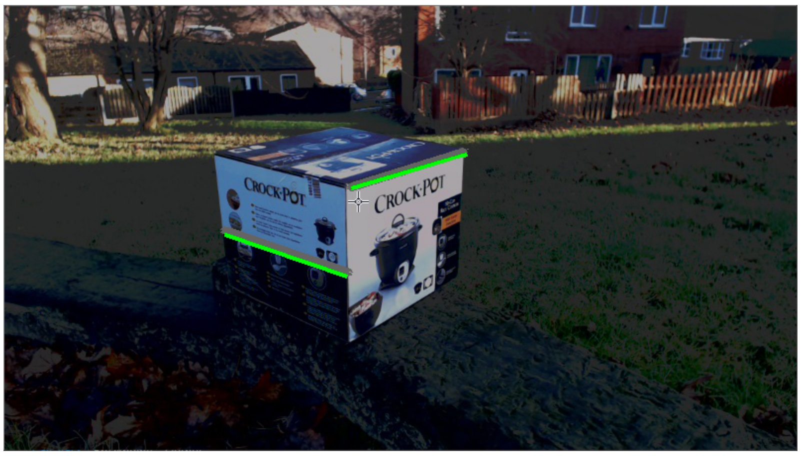
Interactivity allows an user to set a prior by selecting particular lines within the object. By choosing two points describing an object's line from beginning to end. Given two points selected by the user, Equations (5.25) and (5.21) can be used to obtain the Hough transform parameters ρ_H and θ_H representing a line that crosses them. These two parameters are used to create a reduced search of θ_H and ρ_H, allowing to select upper and lower thresholds in both variables (Figure 5.7). The extracted lines would be limited to only those resembling the previously given information, as only one combination of θ_H and ρ_H will result with the most votes in the accumulator. Given these restrictions the line selected by the user is assumed to be the most voted for.

Despite these limitations the results can still be improved. This is because even with user defined θ_H and ρ_H values, it is possible that these cover the entire image, taking data from gradients that are not those of the predominant lines. This also includes intersections with other predominant lines as shown in Figure 5.7c.

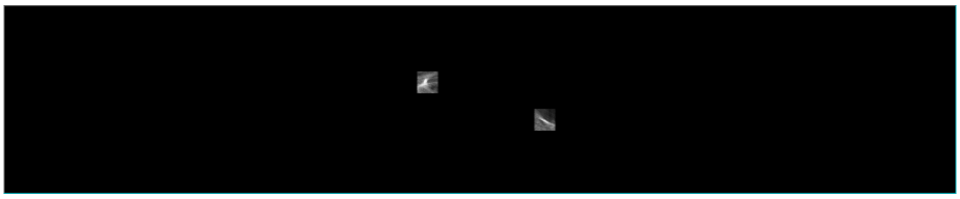
It was then decided to make the algorithm more robust, depending on the continuous length of the detected line. This is akin to drawing a line but instead it is tested for any discontinuities in it, allowing to select a good candidate which exceeds a selected threshold describing its minimal length. This algorithm is presented as an innovation, since its resilience to stray lines presents good results and it is a completely deterministic approach aided by GPGPU.

5.3.2 Hough-Bresenham Algorithm

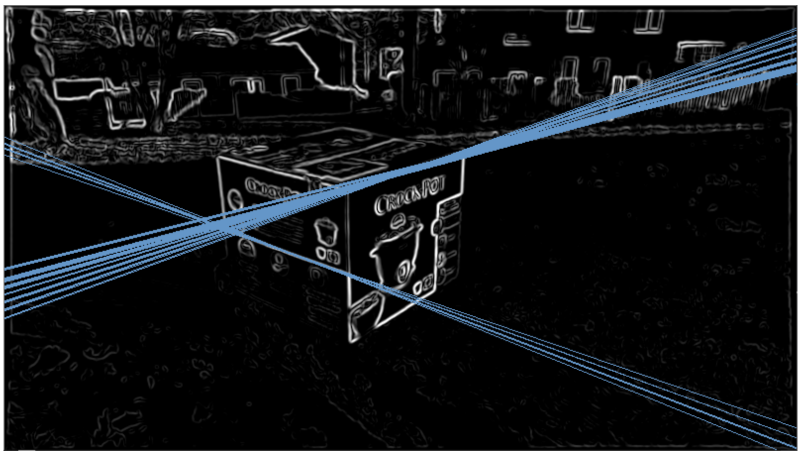
The line generation algorithm of Bresenham is widely known in computer graphics, as it allows to trace lines in an integer lattice, e.g. a computer monitor with pixels. In this section it is used as a tool for detecting good line candidates as images themselves are also lattices, including also binary representations that depict only contours.



(a)



(b)



(c)

Figure 5.7: Hough transform with limited θ_H and ρ_H accumulator values based on a user prior, which is seen as the green lines in (a). Any other pixels below the given intensity threshold, as well as those outside of θ_H and ρ_H ranges are not included in the voting, hence the squares in the Hough accumulator in (b). (c) shows all the lines generated from the accumulator whose number of votes exceed 150. Compared to just selecting an intensity threshold as in Figures 5.5 and 5.6, this method yields fewer lines which are concentrated over the user prior.

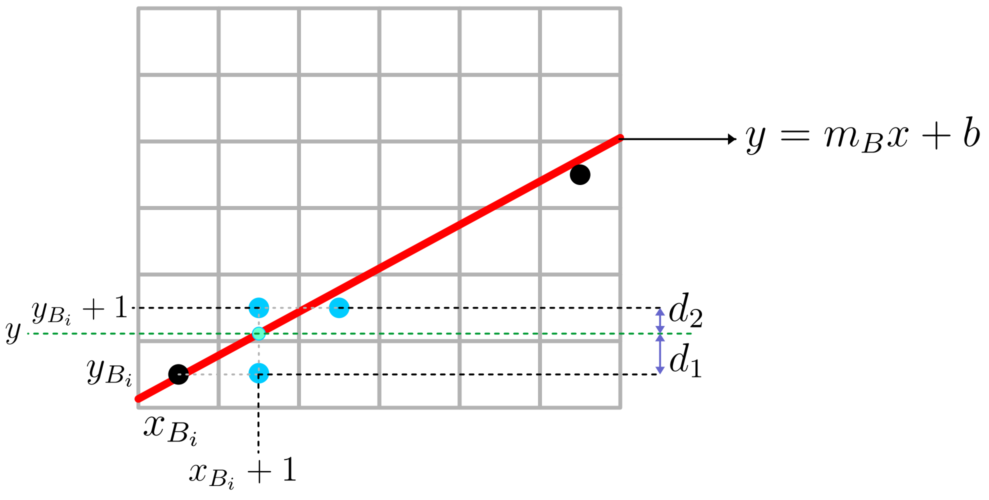


Figure 5.8: Bresenham Line Algorithm. The algorithm draws a line with defined endpoints, such that the general line equation $y = m_B x + b$ can be used. The advantage of this algorithm is that it does so by only using integers, which is important for many computer applications. A grey lattice illustrates pixels, in which the current position of the Bresenham line algorithm is (x_{B_i}, y_{B_i}). The next position moves in the x axis by one pixel or $x_{B_i} + 1$, whereas for the y axis a decision is made based on which distance is closer between d_1 and d_2, which it can be either y_{B_i} or $y_{B_i} + 1$.

To illustrate the basic principle of the algorithm the following assumptions are made: A line possesses endpoints with coordinates (X_1, Y_1) and (X_2, Y_2), in which $X_1 < X_2$. For simplification purposes also the line slope will lie within 0 and 45°, that is $0 < m_B \leq 1$ in the line equation $y = m_B x + b$. This as the algorithm changes slightly depending in which octet of a circle the line lies. The increments of x in the lattice where the line resides will be limited to only 1, whereas those of y will be obtained as seen next.

Assuming that the current line position is (x_{B_i}, y_{B_i}) (Figure 5.8), the algorithm selects to draw the next pixel based on the following criteria: if $d_1 - d_2 < 0$ then $y_{B_{i+1}}$ is the same as y_{B_i}, otherwise if $d_1 - d_2 > 0$ then $y_{B_i} + 1$ becomes the next pixel, i.e. $y_{B_{i+1}} = y_{B_i} + 1$. The distances d_1 and d_2 are calculated using the following formulae:

$$\begin{aligned} d_1 &= y - y_{B_i} = m_B(x_{B_i} + 1) + b - y_{B_i} \\ d_2 &= (y_{B_i} + 1) - y = (y_{B_i} + 1) - m_B(x_{B_i} + 1) - b \end{aligned}$$

with m_B defined as

$$m_B = \frac{\Delta y}{\Delta x} = \frac{Y_2 - Y_1}{X_2 - X_1} \tag{5.27}$$

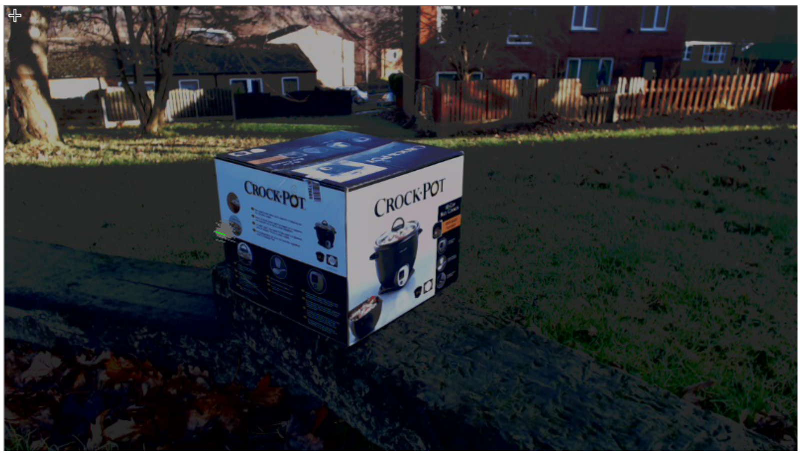
It is worth of mention that m_B from Equation (5.27) is not an integer. To overcome this it is possible to multiply by Δx in order to remove the denominator from $d_1 - d_2$, the result can be stored in a new parameter p_{B_i}:

$$p_{B_i} = \Delta x(d_1 - d_2)$$

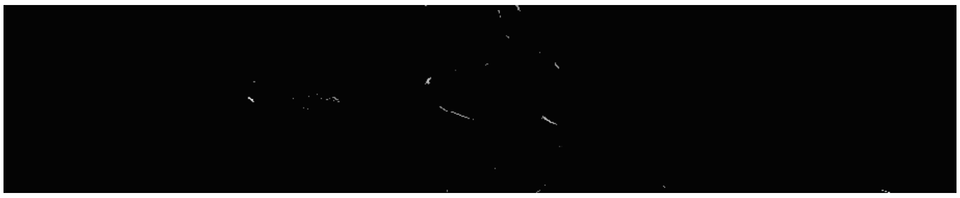
The sign of p_{B_i} will remain positive as we are under the assumption of $0 < m_B \leq 1$. However, the algorithm can be further generalised, making it recursive and without the slope limitations* (Claridge, n.d.).

As the Bresenham algorithm allows to generate lines using only integers, it is possible to use the pixel coordinates that describe a line in a binary image. Therefore, a pixel is considered valid if its intensity value goes beyond a threshold value, otherwise it would be considered a discontinuity within the line. At the same time, since a line is tested in each of its conforming pixels it is possible to obtain its endpoints (Figure 5.9).

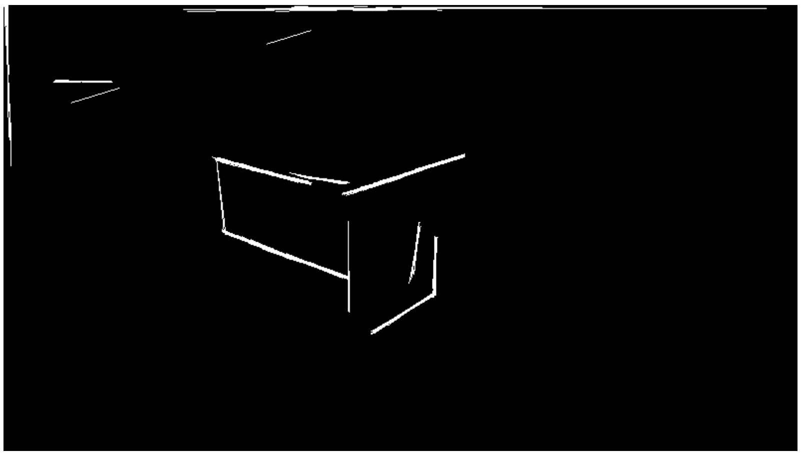
*A full C++ implementation can be found at (Roguebasin, n.d.).



(a)



(b)



(c)

Figure 5.9: Hough-Bresenham transform. The Hough transform and the Bresenham algorithm are coupled to obtain lines from (a) whose accumulator votes in (b) are greater than 50, with pixels whose intensity exceeds a value of 40. Note that the Hough accumulator has its votes concentrated on very small regions, as very few combinations of θ_H and ρ_H are valid. This is also since additional restrictions have been put, so that only lines which have at most 1 pixel of discontinuity and a length of at least 40 are included. It is possible to save the maximum and minimum coordinates of a detected line using the Bresenham algorithm, allowing draws them only within its start and end positions as displayed in (c).

Prominent line search can be constrained using the lines generated from different camera positions or particles, which in turn generate a variety of θ_H and ρ_H values. Therefore, the most supported hypothesis would be the one whose detected prominent line has the most valid pixels (Figure 5.10).

All this information leads to obtain lines that closely resemble an user prior. However, this is yet to assume camera movements and their effects in depth that affect the line end points. Therefore, next sections explore two methods to interactively correct depth considering camera displacements modelled using a familiar method seen in Section 3.3.1. The first considers a projection plane that modifies all the features within an object's surface and the second directly allows to modify the three dimensional coordinates if the line end points.

5.3.3 Depth Assumptions and Camera Motion

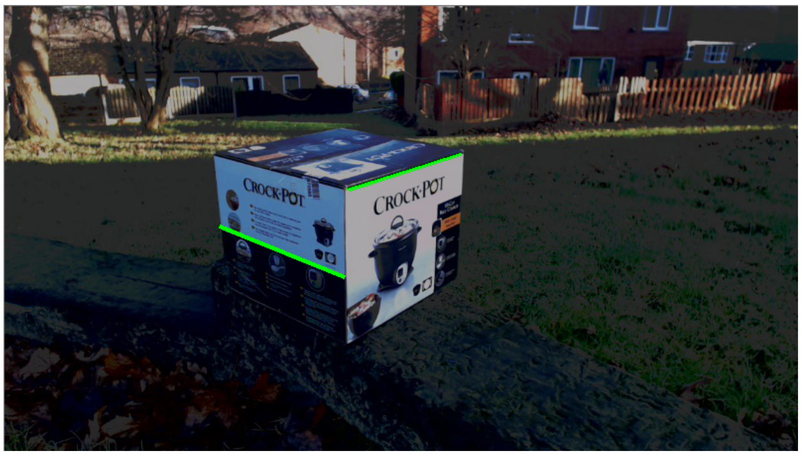
So far line detection has been used only in a static image. In order to perform SLAM the camera must move and estimate its position according to a feature of interest. However, as this investigation relies on more complex features like edges in objects, it requires a slightly different approach to initialisation and estimation.

In Chapters 3 and 4 points are used allowing to initialise them with a predefined depth. This initial depth will change according to the observations obtained whilst the camera is in motion, due to the measurement error caused by the depth assumption and the EKF update. Nevertheless, these initial assumptions sometimes lead to improper camera pose estimation and subsequently drift as seen in Sections 3.4 and 4.4.

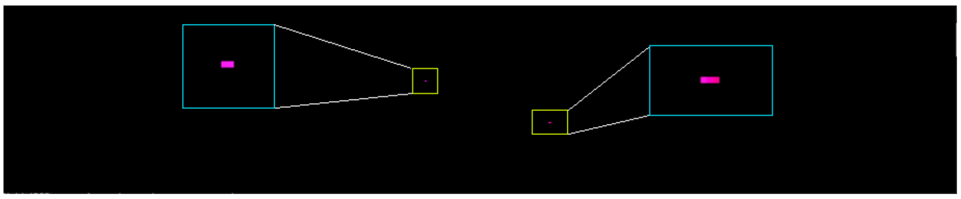
Therefore, in this section is explored a way to interactively change line parameters by means of a projection plane, or by repositioning the line ends. However, first the start and end line points must be added to a world map.

Adding Line End Points to a Map

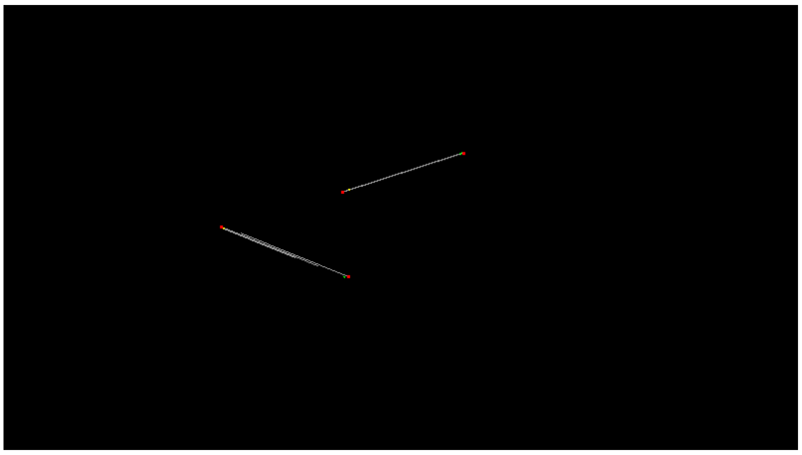
As line end points need to be added into a map an approach similar to the one presented in Section 3.2.3 can be used, in which a point (h_{u_x}, h_{u_y}) is assumed corrected from lens distortions. This can be initialised in the world frame according



(a)



(b)



(c)

Figure 5.10: Hough-Bresenham transform with particle filtering and user line selection. The transform obtains lines from a gradient image, using the green lines as priors which were given by user input in (a). The predominant lines in the image according to this criteria appear as very concentrated dots in the Hough accumulator (small dots within yellow rectangles), the zoomed in portions (blue rectangles) show that they have a very small range of θ_H and ρ_H (b). Only the prominent lines are shown in (c), other lines from which there was no user input are discarded.

to:

$$\begin{bmatrix} x_{f_{iw}} \\ y_{f_{iw}} \\ z_{f_{iw}} \end{bmatrix} = \mathbf{ROT}_{wc} \begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} = \mathbf{ROT}_{wc} \begin{bmatrix} \frac{h_{u_x} - X_c}{f_x} \\ \frac{h_{u_y} - Y_c}{f_y} \\ 1 \end{bmatrix} \tag{5.28}$$

The focal lengths f_x, f_y and the principal points X_c and Y_c are obtained through calibration software. The rotation matrix $\mathbf{ROT}_{wc}$ which allows a feature to be converted from camera to world frame is obtained through Equation (3.34). Similarly, to project back onto the image a line end point that has been added to the map using Equation (5.28) the following formulae can be used:

$$\begin{bmatrix} x_{f_i} \\ y_{f_i} \\ z_{f_i} \end{bmatrix} = \mathbf{ROT}_{cw} \begin{bmatrix} x_{f_{iw}} \\ y_{f_{iw}} \\ z_{f_{iw}} \end{bmatrix}, \quad \begin{bmatrix} h_{u_x} \\ h_{u_y} \end{bmatrix} = \begin{bmatrix} f_x \frac{x_{f_i}}{z_{f_i}} + X_c \\ f_y \frac{y_{f_i}}{z_{f_i}} + Y_c \end{bmatrix} \tag{5.29}$$

With Equation (5.29) the rotation matrix $\mathbf{ROT}_{cw}$ allows a feature to be converted from the world frame to camera frame, and it is obtained using Equation (3.34) and transposing it.

Note that in both Equations (5.28) and (5.29) an orientation quaternion from a camera motion model is required. Therefore, for the purposes of performing particle filtering aided by the Hough-Bresenham algorithm, this model is based on the one presented in Section 3.3.1. However, in this implementation the model is restricted only to translation and orientation:

$$p(\mathbf{x}_k | \mathbf{x}_{k-1}) = \mathbf{x}_{v_k}^- = \begin{bmatrix} \mathbf{r}_k^{wc^-} \\ \mathbf{q}_k^{wc^-} \end{bmatrix} = \begin{bmatrix} \mathbf{r}_{k-1}^{wc} + (\mathbf{v}_{k-1}^{w} + \mathbf{V}^{w})\Delta t \\ \mathbf{q}((\omega_{k-1}^{c} + \Omega^{c})\Delta t) \times \mathbf{q}_{k-1}^{wc} \end{bmatrix} \tag{5.30}$$

where $\mathbf{r}^{wc}$ and $\mathbf{q}^{wc}$ are the camera optical centre position and the camera orientation quaternion respectively, both w.r.t. the world frame as in Equations (3.33) and (3.50). The choice of removing both linear and angular velocities is made, as particle filtering allows to directly estimate positions in the form of particles. Therefore, only translation and orientation states are required to represent different camera position hypotheses. Both translation and orientation states are obtained by randomly generating linear and angular velocities $\mathbf{V}^w$ and Ω^c.

Plane Depth Correction

It is assumed that a feature of interest from the environment has distinctive gradients. For a simple but real example a box is considered allowing user selection, but this possesses gradients that are seen through a two dimensional image. This causes the initial selected line end points to have all z_{f_i} set to 1 in Equation (5.28), resulting in lines which have the same depth (Figure 5.11). Therefore, depth must be accounted for somehow to properly represent the object edges.

A projection plane can be modelled after an object's surface, which will allow to estimate the depth in different points within its surface by summing the distances d_{p1}, d_{p2} and d_{p3} (Figure 5.12). A point P_o will contain coordinates x_o, y_o, z_o within the camera frame. Particles projecting different points using the base coordinates x_o, y_o, z_o will eventually fail to offer a high likelihood, as camera movements will affect the depth z_o that has an influence over the image projection of the point P_o.

The user is allowed to change the two extra parameters ρ_p and θ_p, which will have an influence over all the points in the object's surface (Figure 5.12). This changes the distances d_{p1}, d_{p2} and d_{p3}, according to the following geometrical relationships:

$$\begin{aligned} d_{p1} &= (x_o - O_x)\tan\theta_p \\ s_1 &= (y_o - O_y)\tan\phi_p \\ s_2 &= (s_1)\cos\rho_p \\ d_{p2} &= (s_2)\tan\theta_p \\ d_{p3} &= (s_1)\sin\rho_p \end{aligned}$$

The final corrected d_c distance becomes:

$$\begin{aligned} d_c &= (x_o - O_x)\tan\theta_p + (y_o - O_y)\tan\phi_p\cos\rho_p\tan\theta_p + (y_o - O_y)\tan\phi_p\sin\rho_p \\ &= (x_o - O_x)\tan\theta_p + (y_o - O_y)\tan\phi_p\left[\cos\rho_p\tan\theta_p + \sin\rho_p\right] \end{aligned} \tag{5.31}$$

The origin point O is considered to be a user given point which represents a vertex, obtained whilst selecting lines over an object of interest (Figure 5.13). However, this method for adjusting a feature was shown to be unintuitive for the user, but more importantly it could not properly adjust the projection plane to

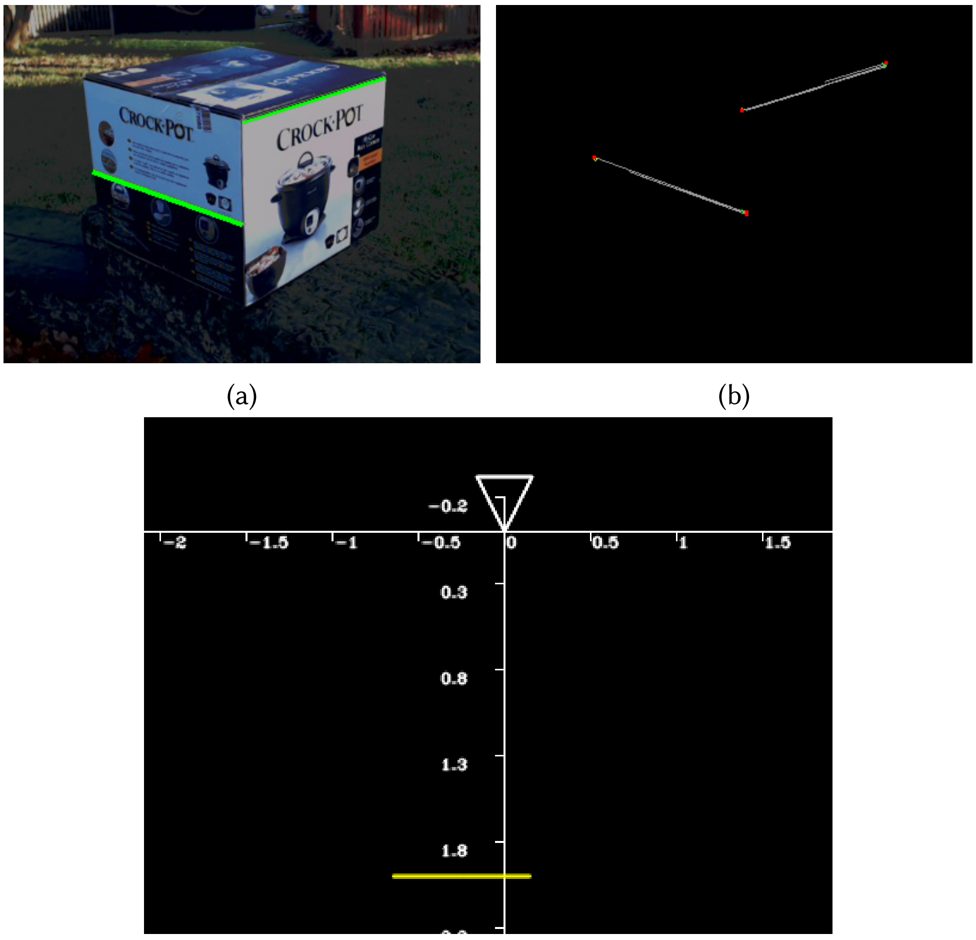
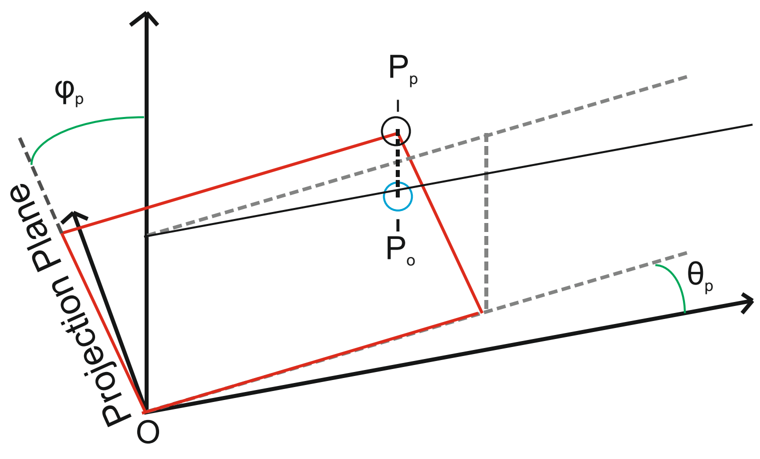
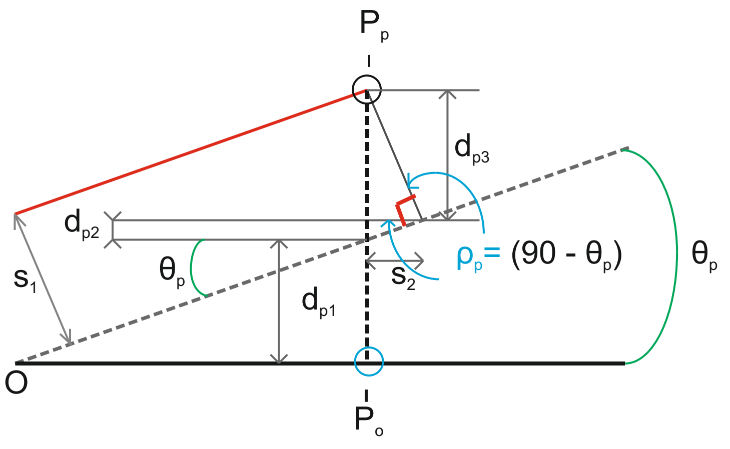


Figure 5.11: Line selection and initial depth as seen from the XZ plane. (a) shows green lines depicting the user gradient selection. (b) shows line detection using Hough-Bresenham transform of Section 5.3.2. (c) shows the projection in the XZ planes of both selected lines. Note that because they are initialised with the same initial depth assumption, the two lines appear to be a single straight line. Numbers in (c) are not representative of any unit as the algorithm works with its own internal units.



(a)



(b)

Figure 5.12: Plane depth estimation over a surface. (a) shows an objects' surface (red rectangle) that is not fully aligned to the camera, presenting both ρ_p and θ_p angles with respect to the camera orientation. (b) shows a top view of the left figure, allowing to see the correct depth of a point P_o in the projected surface and thus becoming P_p. Depth is given by summing the three distances d_{p1}, d_{p2} and d_{p3} through geometrical relationships.

the surface of interest. As such the next method focuses on manual adjustment of user selected lines, based on observation from the camera movements.

Manual Feature Line Adjustment

As using projection planes did not provide any advantage to the user in order to infer depth from an object of interest, a manually guided option was added in which the extremities of lines can be manipulated. This is individual for each line, allowing to observe the changes in their map projection.

Changing only $z_{f_i w}$ in Equation (5.28) will not suffice for getting a better line estimate, as depth alone affects the entire line projection in an image and thus also failing to obtain line estimates from the edges in objects. This is also the reason why using projection planes did not work, as only depth was accounted to be affected when modifying the plane parameters (Figure 5.14). The user is then given the option to change the components $x_{f_{iw}}$, $y_{f_{iw}}$ and $z_{f_{iw}}$ from the line ends. This allows to change depth and correct for any position deviations in the image (Figure 5.15).

Finally, the particle filter algorithm in Section 5.1.3 can be modified to introduce the Hough-Bresenham algorithm, allowing to include user input and enhanced prominent line detection. This still relies on a total number of N particles, with each one of them representing a different hypothesis of the camera position:

- In the beginning of each time step or iteration k, the weights $w_k^{(i)}$ are all initialised with a value of 1/N.
- The user can now enter a prior by selecting two or more points over the image to form lines, possibly describing the edge of an object as in Section 5.2.1.
- Any new start and end points used to describe lines are added to a map according to Equation (5.28) in Section 5.3.3.
- The particles $\mathbf{x}_k^{(i)}$ are generated according to $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ in Equation (5.30). For the purposes of camera localisation, each particle $\mathbf{x}_k^{(i)}$ is generated from a previous particle $\mathbf{x}_{k-1}^{(i)}$ which has been perturbed by randomly generated linear and angular velocities.

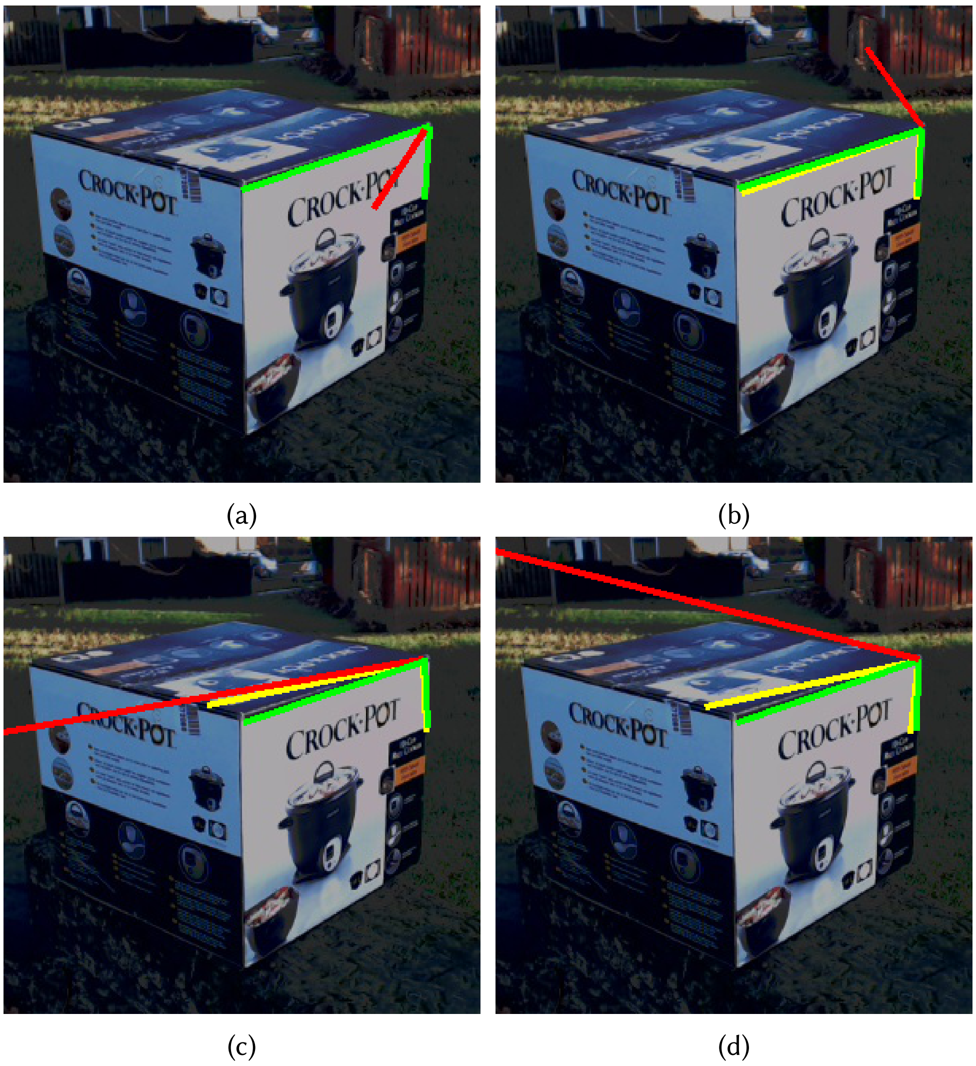
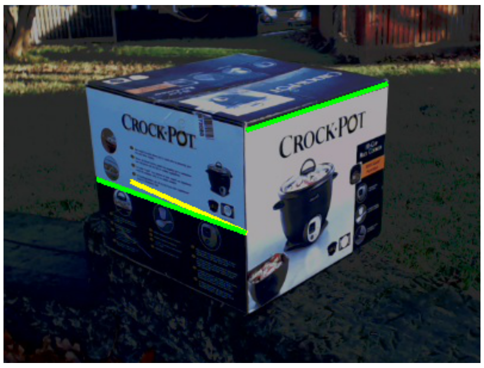
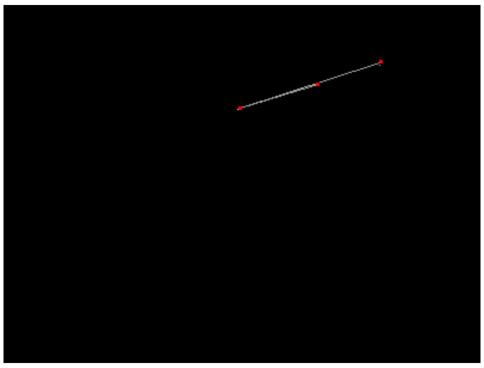


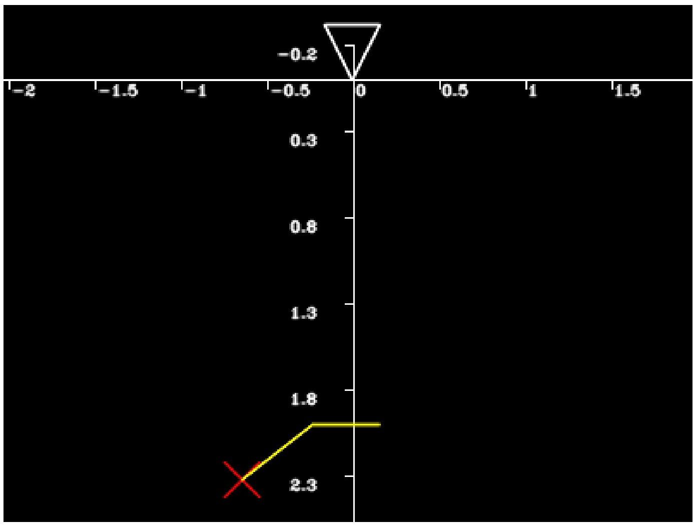
Figure 5.13: Projection Plane with different θ_p and ϕ_p values. (a) shows two lines in green which were previously selected by an user, a red line shows a perpendicular line formed from the past lines. (b) shows a projection plane with $\theta_p = 0^\circ$ and $\phi_p = 14^\circ$ visible with yellow lines, (c) shows another plane with values $\theta_p = 36^\circ$ and $\phi_p = 0^\circ$. Finally (d) depicts the plane with $\theta_p = 36^\circ$ and $\phi_p = 14^\circ$. The perpendicular line has an arbitrary fixed length but this changes due its projection in the image.



(a)

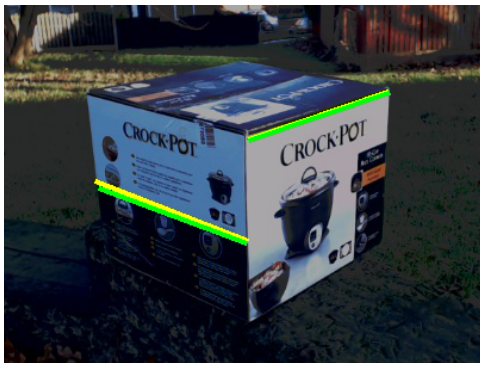


(b)

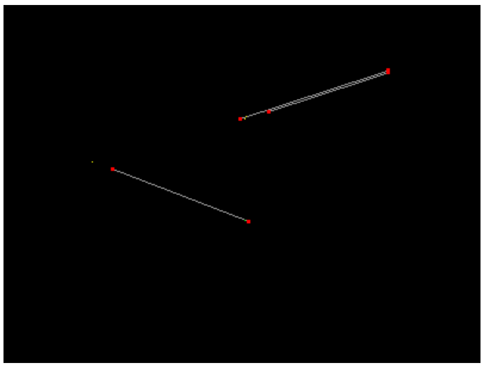


(c)

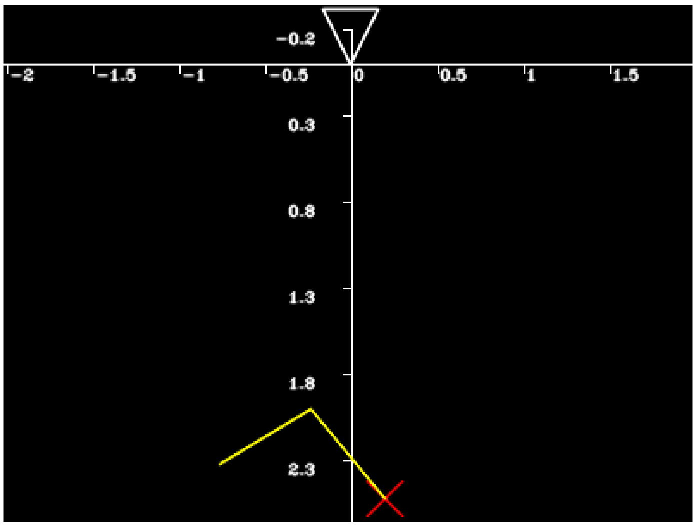
Figure 5.14: User feature depth changes as seen from the XZ plane. (a) shows green lines depicting the initial user line selection, the yellow line represents a feature that has changed depth on one of its ends. Even with only this modification the yellow line shows a change in its perspective, becoming unable to fit in the object edge as it did in the initial user selection. (b) shows line detection using Hough-Bresenham transform of Section 5.3.2, note that the line whose depth has been changed does not have a gradient and hence it is not detected. (c) shows the projection in the XZ planes of the lines, with the first line changed in depth. Numbers in (c) are not representative of any unit as the algorithm works with its own internal units.



(a)



(b)



(c)

Figure 5.15: User feature parameter changes as seen from the XZ plane. (a) shows green lines depicting the initial user line selection, whereas the overlapped yellow lines represent the same lines with user changes in $x_{f_{iw}}$, $y_{f_{iw}}$ and $z_{f_{iw}}$. (b) shows line detection using Hough-Bresenham transform of Section 5.3.2. (c) shows the projection in the XZ planes of the lines, an user has adjusted both of the lines depicting more closely the topology of the box. Numbers in (c) are not representative of any unit as the algorithm works with its own internal units.

- For each particle $\mathbf{x}_k^{(i)}$ or camera location hypothesis, the start and end points from the user prior are projected back onto the image using Equation (5.29). These points are used to produce different combinations of θ_H and ρ_H depending on the camera location described by its particle.

- For each particle $\mathbf{x}_k^{(i)}$ its weight $w_k^{(i)}$ is obtained following Equation (5.17), which uses the likelihood $p(\mathbf{z}_k|\mathbf{x}_k)$. This likelihood is obtained for each particle by using the Hough-Bresenham algorithm described in Section 5.3.2, which counts the amount of votes from the individual combinations of θ_H and ρ_H that each particle holds.

- Depending on the amount of votes for each particle, an user can decide whether or not to adjust the line parameters $x_{f_{iw}}$, $y_{f_{iw}}$ and $z_{f_{iw}}$ after camera motion. This actively requires the user to pause the particle filter algorithm, in order to adjust the lines so that the edges fit better onto the object of interest. The algorithm can be resumed shortly afterwards.

- Resampling is performed, adding all the weights $w_k^{(i)}$ and then normalising them so that their total sum is equal to 1. Afterwards the array in Equation (5.18) is formed together with the array from Equation (5.20). This allows to perform systematic resampling as described in Section 5.1.3 (Figure 5.1) after reordering terms using the weight values. This marks also the start of a new cycle.

The next section implements all of these steps, allowing to see the performance of particle filtering coupled with the Hough-Bresenham algorithm, including active user intervention. This is also compared against the EKF algorithm seen in Chapter 3.

5.4 Implementation Results

This section presents the results obtained from particle filtering aided with the Hough-Bresenham algorithm, in which an user is able to select edges from an object (a box) allowing to perform camera localisation. The particle filter has been set with a total number of $N = 90$ particles and the Hough-Bresenham algorithm has been set with a pixel intensity threshold of 40, a maximum of 1 pixel of discontinuity and a minimum line length of 40.

The implementation works in real-time by using a source video avoiding using frames individually after each time step. It is worth mentioning that this does not consider meaningful units for mapping, as the main interest is to ensure stability within the implementation. Therefore, the particle filtering algorithm also works using its own internal units.

For this scenario the camera was hand-held performing approximately smooth motion patterns as follows: yaw left rotation, yaw right rotation, pitch tilting down, forward translation, yaw right rotation, yaw left rotation, rotation around the object (counter clockwise), stand still position, slow rotation around the object (counter clockwise), pitch tilting up, pitch tilting down and some erratic motion. It must be noted that as soon as the particles were observed not to properly fit the object's edges, the video was paused in order to adjust the line parameters as in Figure 5.15.

The novelty of this approach relies on object selection according to user preference, for this case objects with edges present themselves as good candidates. This is related to another novelty introduced in this chapter, which joined the weighting requirement from the particle filter with the voting capabilities of the Hough accumulator, further improved with the Bresenham algorithm to detect lines of interest.

Compared to the baseline implementation of SLAM with inverse depth parametrisation in Chapter 3, and the approach aided by active contours in Chapter 4, this implementation does not present instabilities caused by loss of observations. This is as the particles themselves scatter when all the hypothesis remain with the same weight, i.e. when no gradients are detected (Figure 5.16).

By introducing interactivity in this implementation an user is able to modify the parameters $x_{f_{iw}}, y_{f_{iw}}$ and $z_{f_{iw}}$ after camera motion in Equation (5.28), which allows to create a better fit to the object's edges or to correct initial assumptions. The particle filter still looks for the best match, using the feature as a reference for camera localisation.

Naturally, these estimates will possess errors whilst an user is modifying the line parameters. Once a proper combination is found the camera position estimate should improve and even when this is not conclusive for pose estimation, it demonstrates algorithm stability. This is even after extended periods of time,

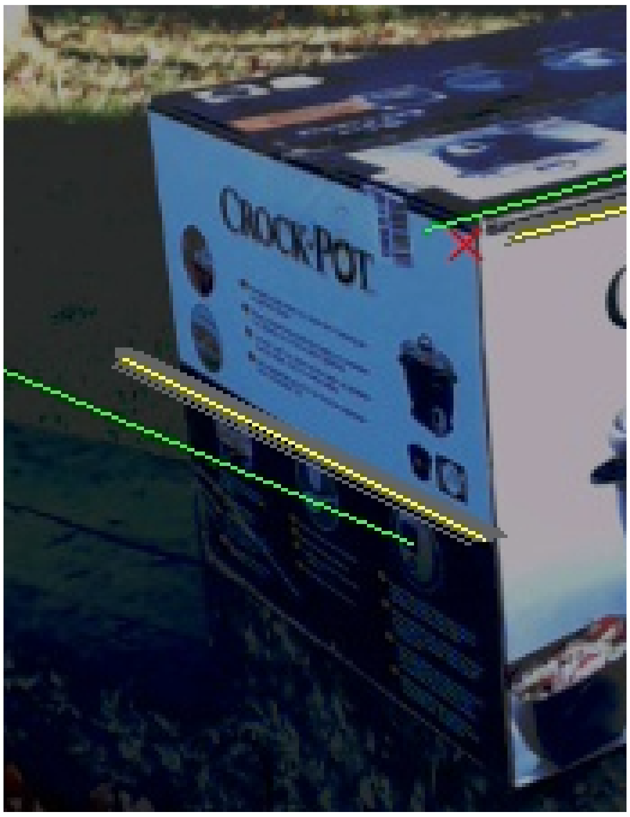


Figure 5.16: Scattered particles. One of the properties of particle filtering is that when no hypothesis has more votes than another, the newly resampled particles present scattering (Grey lines covered by the yellow line). This allows more resilience to erratic motion, since the scattered particles cover a wider area allowing to retrieve the object's edge gradient observation. Once a hypothesis with high weight is found, the particles quickly pile up towards the most weighted one.

which also accounts for object parameter correction (Figures 5.17, 5.18 and 5.19). In the same graphs it can be seen that compared to its EKF counterpart the camera estimates tend to jump less. This is since adding and deleting features affects uncertainty (Figures 5.20 and 5.21). After a while the uncertainty grows big enough that the system cannot recover, whereas this does not occur in the presented implementation using the particle filter. When there is the case that observations from the object's edges are totally lost, an user can simply reposition the lines. However, particle filtering produces noisier estimates and also requires a higher computational cost than EKF, but the gains reflect greatly in algorithm stability and flexibility in hypotheses weighting or voting.

5.5 Concluding Remarks

Particle filtering with a novel line voting algorithm has been presented, which also makes use of interactivity for high-level object tracking. Line detection involves using the Hough transform, which is then further improved by means of a Bresenham algorithm. From this approach the following conclusions are drawn:

- Particle filtering was chosen for its feasibility to obtain estimates which involve many dimensions as seen in Section 5.1. In this investigation it is seen that it behaves more stably than other approaches which rely on the EKF, as it is not prone to matrix inversion problems and avoids linearisation.

- The Hough transform is a useful algorithm for detecting lines by voting all pixels in the image, which are then put into an accumulator allowing to discern edges as seen in Section 5.2. This is ideal in particle filtering, as its flexibility in hypotheses allows resampling particles, in the same way as they do when the most supported hypothesis is carried over to the next iteration with many copies of it.

- Interactivity is introduced by an user setting an initial likelihood in the form of a line, which can be used for high-level object tracking and detection. The Hough transform fills the Hough accumulator with votes according to prominent lines in the image, these also serve to find the most supported

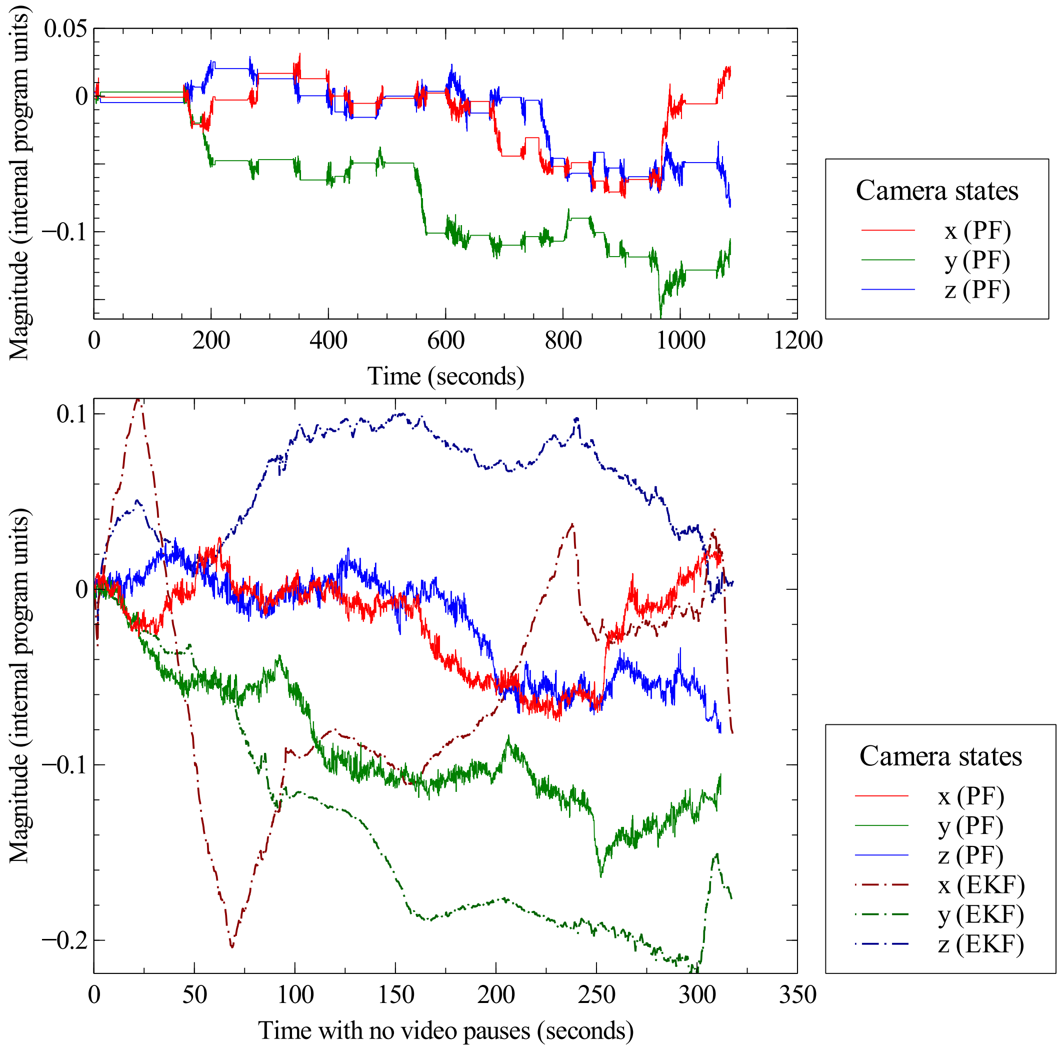
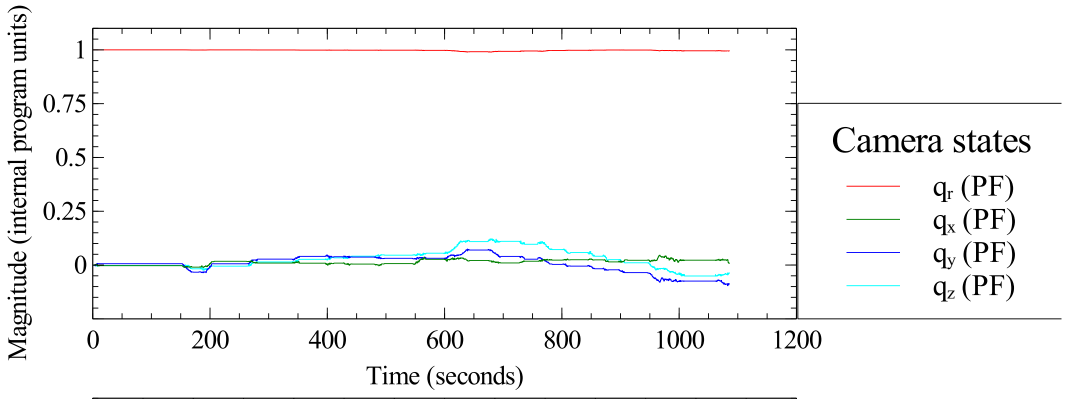


Figure 5.17: EKF SLAM baseline inverse depth parametrisation and particle filter, camera translation states. Top graph depicts translation states with included time pauses in particle filter, hence the straight lines in between the estimates. Bottom graph removes these pauses leaving only translation values. EKF estimates are also shown in the bottom graph, jumping in more magnitude compared to the estimates obtained through particle filtering.



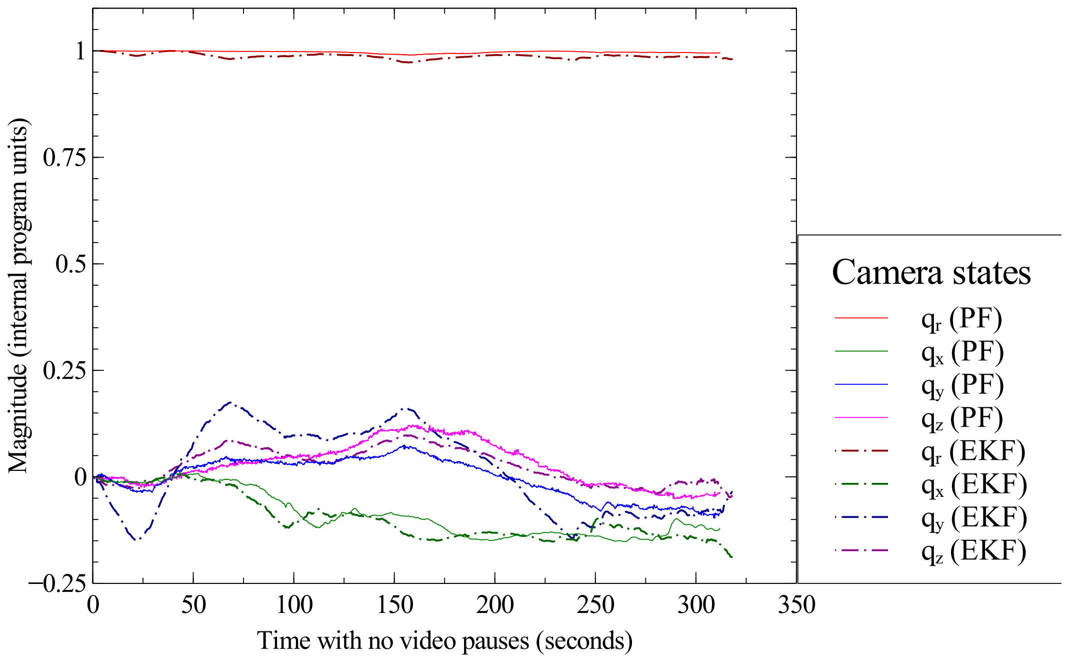


Figure 5.18: EKF SLAM baseline inverse depth parametrisation and particle filter, camera quaternion states. Top graph depicts quaternion states with included time pauses, hence the straight lines in between the estimates. Bottom graph removes these pauses leaving only quaternion estimates. EKF estimates are also shown in the bottom graph, in which the estimates from q_x and q_y drift considerably compared to the estimates obtained through particle filtering.

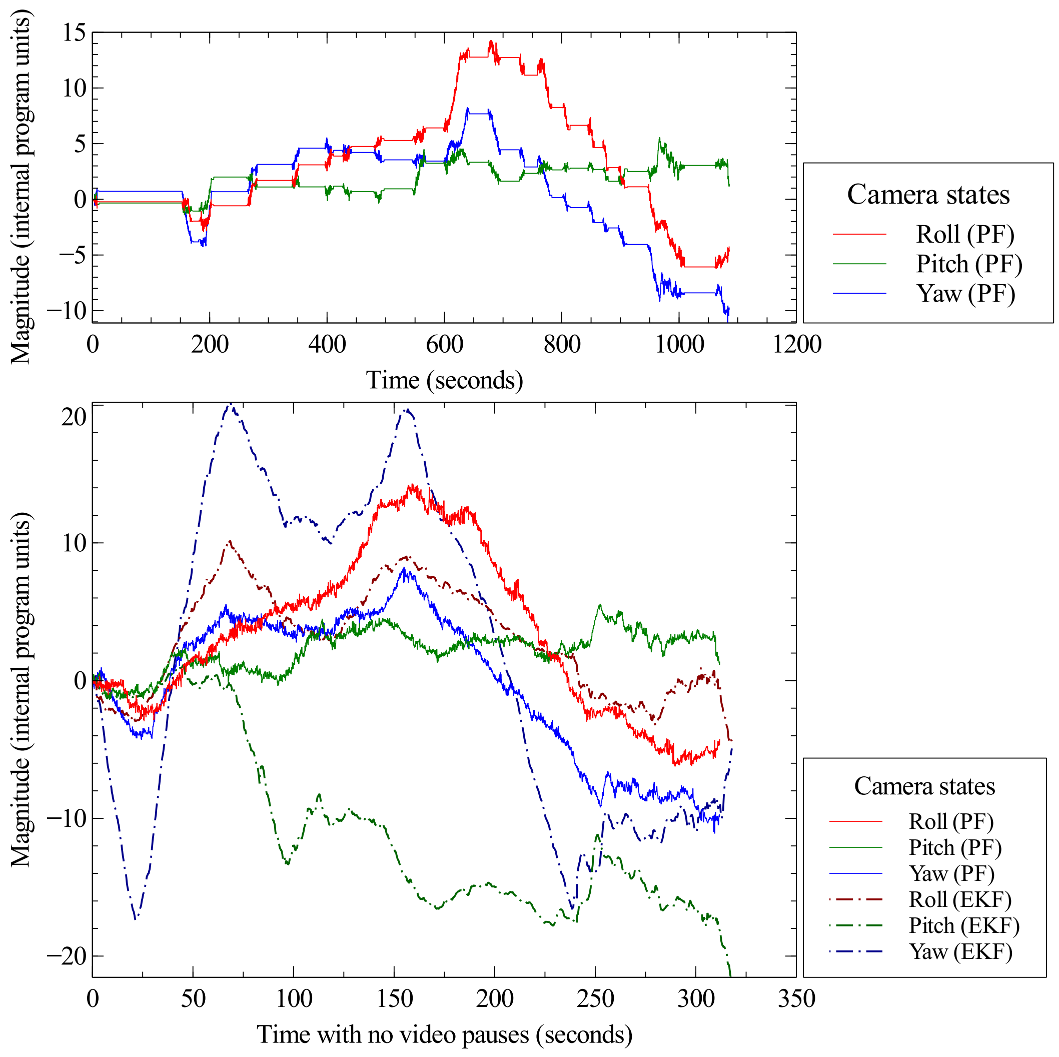


Figure 5.19: EKF SLAM baseline inverse depth parametrisation and particle filter, camera orientation from quaternion states. Top graph depicts camera orientation from quaternion estimates with included time pauses, hence the straight lines in between the estimates. Bottom graph removes these pauses leaving only camera orientations from quaternion values. EKF estimates are also shown in the bottom graph, in which the estimates from pitch and yaw drift considerably compared to the estimates obtained through particle filtering. Note that the roll remains almost the same in both cases, which is comparable to the results obtained from the EKF SLAM baseline inverse depth parametrisation aided by active contours (Figure 4.18).

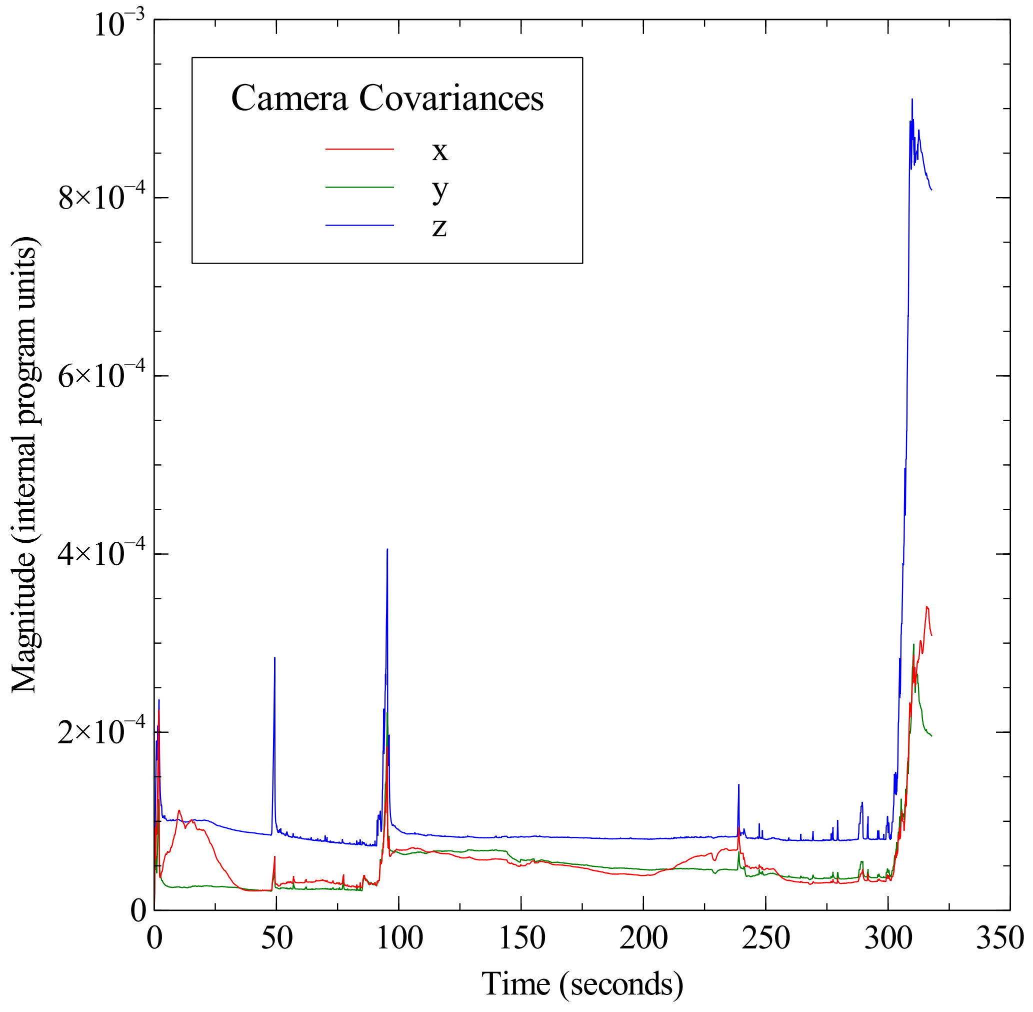


Figure 5.20: Inverse depth parametrisation EKF covariance for camera translation states using particle filter video. Peaks represent whenever new features are added, yet at a certain point the algorithm is not able to reduce uncertainty. This leads to failure in properly update the EKF covariance matrix around 300s and hence the overshoot. Covariance magnitude numbers are not representative of any unit as the algorithm works with its own internal units.

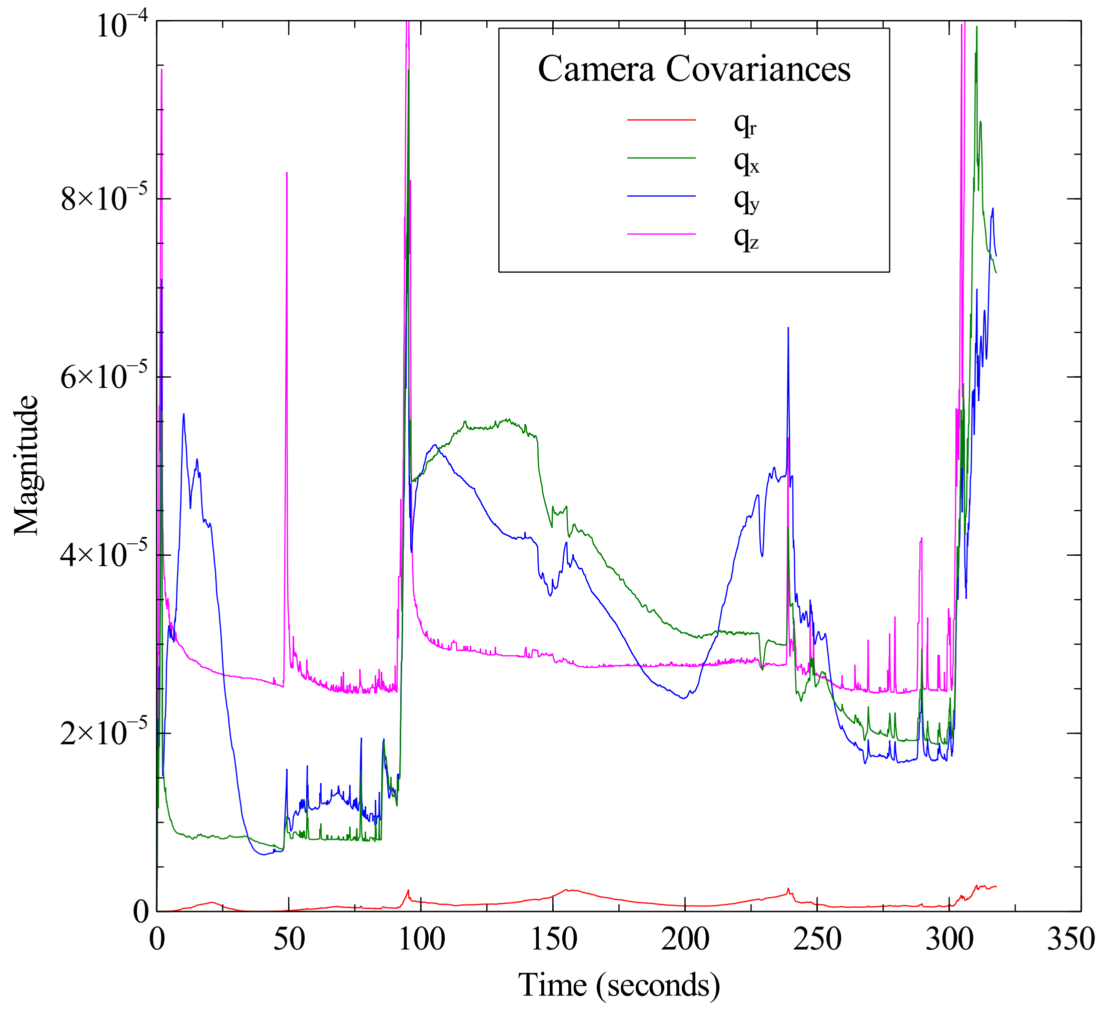


Figure 5.21: Inverse depth parametrisation EKF covariance for camera quaternion states using particle filter video. Peaks represent whenever a new feature is being added, yet at certain point the algorithm is not able to reduce uncertainty. This leads to failure in properly update the EKF covariance matrix around 300s, hence the overshoot in values.

hypothesis for camera pose. However, the Hough transform does not account for line discontinuities. As such, the accumulator might give a high vote count to lines which are formed from stray image pixels. The Hough-Bresenham algorithm solves this issue by checking for line discontinuities, discarding those that do not follow continuity pixel by pixel as seen in Section 5.3. Therefore, high-level object tracking through particle filtering is also improved.

- The results obtained from this novel implementation of particle filtering, aided by Hough-Bresenham line detection yields improved stability. This is even considering algorithm pauses in order to correct high-level object line properties as seen in Section 5.4. Even when the lock-on by the particle filtering is lost, it is possible to recover tracking through user input. This is in part because it is possible to rely on fewer features compared to many points in other SLAM implementations.

Overall, particle filtering presents long term stability and good computational performance. This is in part thanks to high-level objects which allow to reduce computational load, by focusing on more abstract features from an object of interest. This abstraction is done using lines, which rely on both line extraction and user input. Interactivity therefore allows to focus detection onto lines of interest, whereas a novel Hough-Bresenham transform allows to discern continuous lines from those containing only scattered pixels.

Still, this investigation really only scratches the surface regarding interactivity and high-level objects in SLAM. This is mainly due to the fact that abstraction can be presented in many forms, which at the same time can be exploited in many ways. Particle filtering offers good performance using these abstractions, as it allows to estimate many camera pose states whilst focusing hypotheses into objects rather than many localised points.

EKF on the other hand has proven to be very situational and dependent on controlled conditions, otherwise it is very easy to reach divergence. Many investigations have focused efforts on improving stability, as this is a priority in estimation algorithms. However, particle filtering offers a viable solution to estimation considering a focus into high-level objects. This was the main aim of this thesis, which is to open a new branch within vision-SLAM focused on feature abstraction aided by interactivity.

Chapter 6

Conclusions and Future Work

Chapters 1 and 2 focused on exploring Simultaneous Localisation And Mapping (SLAM) with particular focus on its vision-SLAM branch in order to find works that relied on active user input in order to improve localisation. The result of this yielded no direct method to include interactivity, as most of the investigations try to push the envelope in regards to feature and camera estimation, dense mapping, feature re-association after periods of missed observations, indoor and outdoor applications. Many other works are derived from these investigations as SLAM is by itself a multidisciplinary research subject.

In spite of this, no real effort has been made to include active user intervention within the SLAM algorithm, ignoring any real-time feature input or selection. This can be explained by a continuous push towards automated approaches in engineering. However, there are works that improve over the SLAM concept by improving on feature representation. These include approaches taking complex cues from the environment such as lines, planes, ceilings, walls or even riverine areas using curve fitting techniques, also more complex objects can be used if offline databases are used for their recognition.

However, fully autonomous algorithms often require high computing resources which limit real-time operation. This is also present in SLAM implementations, in which also unaccounted situations outside of algorithm programming might lead to malfunction or task interruption. An hybrid approach involving autonomous operation and high-level decision making can make it possible to overcome these problems, by introducing user interactivity in SLAM. Particularly in vision-SLAM this presents other advantages, since persons often segment objects better than machines. This ability comes from years of experience in contact objects, allowing to offload feature selection and recognition to an operator.

In order to introduce active user input into vision-SLAM first its inner workings are explored, as all the information required to performed is assumed known from considerable amounts of past investigations. This also allows to see how to introduce interactivity in the algorithm, from which two approaches are proposed. The first considers active contours (snakes) due to its continuous deformation towards a local minima on the image, which can represent an object. The second uses particle filtering in which a person actively modifies the properties of an object of interest, with particles as different camera pose hypotheses.

Chapter 3 focused on exploring a baseline implementation of SLAM with inverse depth parametrisation as there is no full compilation for it. This shows a huge dependence on an Extended Kalman Filter (EKF), which is often used for SLAM implementations as it offers a compelling framework for it, by relating robot and feature uncertainties using a covariance matrix.

The covariance matrix that holds all the feature and camera uncertainties is updated by the EKF algorithm, after obtaining observations from concurrently added localised features in the form of points. These are affected by image imperfections which are dealt with using a Brown-Conrady model which is capable of dealing with radial distortions, caused by wide angle lenses often recommended for SLAM due to its increased field of view. This model also offers good invertible properties, making it possible to remove and add deformations from the localised features. Using points has the caveat of requiring dense acquisition in order to provide an idea of the surroundings, increasing computational requirements.

However, EKF in SLAM is an algorithm prone to instabilities due to the need for complex matrix operations in the covariance matrix. As features are added, removed and accumulated over time the covariance matrix becomes more difficult to invert. Eventually this causes numerical errors leading to the algorithm unable to recover from drift. This problem becomes more accentuated when all observations are lost, due to small but sudden motion changes. Therefore, maintaining the same features for as long as possible might be a good solution to avoid these issues.

Chapter 4 introduced a novel approach for feature abstraction using active contours driven by Gradient Vector Flow (GVF). However, this is a computationally expensive tool to use as it depends on heavy image processing. General Purpose computing on Graphics Processing Units (GPGPU) greatly alleviates and accelerates image manipulation tasks, which translates real-time active contour deformation for tracking objects according to camera motion. This is possible as GVF relies on independent operations applied over each image pixel, which is ideal for huge parallelism gains.

As a picture contains many gradients it was decided to use high contrast objects for this implementation. These allow the snake not to become distracted and ensure good locking onto an object for tracking. Compared to the implementation in Chapter 3, this approach did require less features and did not continuously add and remove them. This improved on the instabilities caused by having a sizeable covariance matrix.

Other important improvements were noted as well, which include observation stability and the inclusion of user input in the form of description or semantics. The former is possible as the active contour deformations smoothed measurements, avoiding the loss of observations that otherwise were caused by moving the camera in opposite direction. The latter allows to input information right after object selection, which can be used for descriptive or semantic purposes in a map avoiding the need to perform dense feature acquisition. Despite this, instabilities were still observed due to the use of the EKF as an estimator for camera motion, moderately after long periods of time and thus it was decided to drastically move onto particle filtering.

Finally, Chapter 5 presented a radical change in the direction of this investigation but still kept the idea of introducing interactivity in SLAM. The main aim was to use particle filtering as its capabilities allow for long term runs, independence to the number of dimensions or camera states to estimate and flexibility in hypothesis (particle) weighting. This algorithm does not rely on complex matrix operations nor linearisation, which avoids the numerical problems seen in the implementations from Chapters 3 and 4 at the cost of requiring more computational resources.

The Hough transform is a well known method for extracting lines from an image, by means of votes and an accumulator containing all the bins. Votes and particle weighting share an inherent relationship and thus it can be used as a likelihood for camera estimation. This leads to introduce interactivity within the algorithm, as an user can now describe objects by means of its edges. Particles then represent different hypotheses pertaining to camera position, with each one of them projecting a modified line from the original given input. The line containing most votes is the most likely and thus the most weighted particle.

The Hough transform in its baseline implementation is able to extract lines from all the pixels image. However, this also means that only two points are needed to produce them and thus even noise within the image will yield invalid votes. Therefore, Hough transform is improved by means of using a Bresenham algorithm. The latter is used to draw lines over latices using only integer values, but here it has been modified to produce a novel Hough-Bresenham algorithm. This allows to detect prominent lines in an image but only accounting for those where discontinuities are not present, filtering as well pixels that are produced by noise or isolated from a prominent line.

All of this has been accelerated using GPGPU technologies in order to perform real-time camera location. User input is further refined to also include three dimensional repositioning of the initially placed lines, so that they can be corrected and properly fit after camera motion since initial depth assumptions will yield invalid lines. Compared to EKF which turned out to be a situational approach prone to drift, this implementation reached long term stability with moderate computational requirements. Still, whereas the main aim of investigation was to introduce interactivity in SLAM, it only begins to dive into interactive feature abstraction in SLAM as there is not a unique way to attain it.

6.1 Future Work

Based on the stability results presented in Chapter 5, this method shown to be very promising for future investigation. With some key aspects to be considered:

- The nature of particle filtering allows for high parallelism, therefore GPGPU technologies can be used to greatly accelerate the performance of this estimator. The most crucial step in this approach is the sorting of all the weights, but assuming that camera location follows an unimodal distribution this can be sorted by using atomic operations in GPUs. These allow to perform certain serial computations that are often limited to only CPUs. Therefore, if the algorithm is efficiently coded it could be put onto drones with a remote operator using a portable device allowing input for object description.

- Object abstraction can be further improved, as both Hough transform and Bresenham algorithm can be used to detect and draw more than lines. They have been also used to detect more complex shapes such as circles. Therefore, as long as there is a possibility to count pixels from gradients it is possible to obtain votes from them.

- Describing objects through user input can be further exploited. This includes not only drawing lines overlaid on top of a camera feed but rather more intuitive methods can be explored, e.g. following Computer Assisted Design (CAD) software guidelines. These are often based on geometric properties that allow to create symmetrical objects based on very simple shapes. After an object has been created in this way only its edges are required, these will be used in conjunction with the Hough-Bresenham algorithm to provide likelihoods.

- Fully automation based on feature abstraction is not denied. An artificial intelligence can be created so that it can detect gradients belonging to objects, therefore only asking an user for descriptive or semantic input. This information can be used on holographic wearable devices such as the Hololens*, which would allow to interactively map the surroundings of a person. This would allow to create a map that is based merely on objects and does not rely on dense representations, therefore being more useful for tasks based on objectives.

This investigation managed to open a door to introduce users into a fully automated approach like SLAM, making it part of the algorithm. Many further possibilities can be considered if augmented reality is taken into account, including recreational, work or search and rescue tasks. Therefore, it is concluded that there is a considerable amount of applications that will be discovered as time passes.

* *https://www.microsoft.com/microsoft-hololens/en-us*

Chapter 7

Appendix

Videos for the implementations can be seen in the following URLs:

- For Chapter 3 *https://www.youtube.com/watch?v=OO7b31zfM1U*, EKF with inverse depth parametrisation.
- For Chapter 4 *https://www.youtube.com/watch?v=N3ts6Px0z7g*, EKF with inverse depth parametrisation aided by active contours. Also *https://www.youtube.com/watch?v=w_9YCNpnfpE*, EKF only using the same source video.
- For Chapter 5 *https://www.youtube.com/watch?v=T4YqLrprnhA*, particle filtering with Hough-Bresenham algorithm. Also *https://www.youtube.com/watch?v=u495A9_en-w*, EKF only using the same source video.

Bibliography

Evan Ackerman and Erico Guizzo. irobot brings visual mapping and navigation to the roomba 980, 2015. URL *http://spectrum.ieee.org/automaton/robotics/home-robots/irobot-brings-visual-mapping-and-navigation-to-the-roomba-980*. [Online; accessed 2-Jun-2016].

Sunghwan Ahn, Minyong Choi, Jinwoo Choi, and Wan Kyun Chung. Data association using visual object recognition for EKF-SLAM in home environment. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2588–2594. IEEE, 2006.

Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J. Davison. KAZE features. In *Computer Vision (ECCV) 2012*, pages 214–227. Springer, 2012.

Abbas M. Ali and Md Jan Nordin. SIFT based monocular SLAM with multi-clouds features for indoor navigation. In *TENCON 2010-2010 IEEE Region 10 Conference*, pages 2326–2331, 2010.

H. Andreasson, T. Duckett, and A.J. Lilienthal. A Minimalistic Approach to Appearance-Based Visual SLAM. *IEEE Transactions on Robotics*, 24(5):991–1001, October 2008.

Henrik Andreasson, Tom Duckett, and Achim Lilienthal. Mini-SLAM: Minimalistic visual SLAM in large-scale environments based on a new interpretation of image similarity. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4096–4101. IEEE, 2007.

George B Arfken, Hans-Jurgen Weber, and Frank E Harris. *Mathematical methods for physicists: a comprehensive guide.* Elsevier, Amsterdam; Boston, 2013.

Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, 2006.

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008.

Paul Berner. Technical concepts: Orientation, rotation, velocity and acceleration, and the srm, 2008. URL *http://www.sedris.org/wg8home/Documents/WG80485.pdf*. [Online; accessed 12-May-2016].

Gabriele Bleser and Didier Stricker. Using the marginalised particle filter for real-time visual-inertial sensor fusion. In *Mixed and Augmented Reality, 2008. ISMAR 2008. 7th IEEE/ACM International Symposium on*, pages 3–12. IEEE, 2008.

Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A benchmark. *IEEE Transactions on Image Processing*, 24(12):5706 – 5722, 2015.

Duane C Brown. Decentering distortion of lenses. *Photogrammetric Engineering.*, 32(3):444–462, May 1966.

Luca Carlone and Andrea Censi. From Angular Manifolds to the Integer Lattice: Guaranteed Orientation Estimation With Application to Pose Graph Optimization. *IEEE Transactions on Robotics*, 30(2):475–492, April 2014.

Tat-Jen Cham and James M. Rehg. A multiple hypothesis approach to figure tracking. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, 1999.

Denis Chekhlov, Andrew P. Gee, Andrew Calway, and Walterio Mayol-Cuevas. Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual SLAM. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–4. IEEE Computer Society, 2007.

Jinwoo Choi, Sunghwan Ahn, Minyong Choi, and Wan Kyun Chung. Metric SLAM in home environment with visual objects and sonar features. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4048–4053. IEEE, 2006.

Chen Chwan-Hsen, Ze Yuan, and Chan Yung-Pyng. Sift based monocular SLAM with inverse depth parameterization for robot localization. In *Proc. IEEE Workshop on Advanced Robotics and Its Social Impacts ARSO 2007.*, pages 1–6, December 2007.

J. Civera, A.J. Davison, and J. Montiel. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, 24(5):932–945, October 2008.

Javier Civera, Oscar G. Grasa, Andrew J. Davison, and J. M. M. 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, May 2010.

Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D. Tardós, and J. M. M. Montiel. Towards semantic SLAM using a monocular camera. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS*, pages 1277–1284, 2011.

E Claridge. Derivation of Bresenham's line algorithm, n.d. URL *http://www.cs.bham.ac.uk/~vvk201/Teach/Graphics/Bresenham_derivation.pdf*. [Online; accessed 11-March-2016].

T. H. J. Collett and Bruce A. MacDonald. Developer oriented visualisation of a robot program. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 49–56. ACM, 2006.

M.G. Cox. The numerical evaluation of b-splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, April 2008.

Andrew J. Davison. Real time simultaneous localisation and mapping with a single camera. In *Proc. Ninth IEEE International Conference on Computer Vision*, pages 1403–1410, 2003.

Andrew J. Davison and Nobuyuki Kita. 3D simultaneous localisation and map-building using active vision for a robot moving on undulating terrain. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, volume 1, pages 384–391, 2001.

Andrew J. Davison and David W. Murray. Mobile robot localisation using active vision. In *Proc. 5th European Conference on Computer Vision*, volume 1407, pages 809–825. Springer, 1998.

Andrew J. Davison and David W. Murray. Simultaneous localization and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):865–880, 2002.

Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, June 2007.

Hendrik Deusch, Stephan Reuter, and Klaus Dietmayer. The Labeled Multi-Bernoulli SLAM Filter. *IEEE Signal Processing Letters*, 22(10):1561–1565, October 2015.

Frédéric Devernay and Olivier D. Faugeras. Automatic calibration and removal of distortion from scenes of structured environments. In *SPIE's 1995 International Symposium on Optical Science, Engineering, and Instrumentation*, pages 62–72. International Society for Optics and Photonics, 1995.

James Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Tech Rep. Stanford University*, 2006.

M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241, 2001.

Randal Douc and Olivier Cappé. Comparison of resampling schemes for pinproceedings filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64–69. IEEE, 2005.

Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7): 932–946, 2002.

Markéta Dubská, Adam Herout, and Jirí Havel. PClines line detection using parallel coordinates. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1489–1494. IEEE, 2011.

Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

R. Porter E. Rosten and T. Drummond. Faster and better: A machine learning approach to corner detection. In *IEEE Trans. on Pattern Analysis and Machine Intelligence*, volume 32, pages 105–119, January 2010.

Ethan Eade and Tom Drummond. Scalable monocular SLAM. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 469–476, 2006.

Eigen. Eigen is a C++ template library for linear algebra, n.d. URL *http://eigen.tuxfamily.org*. [Online; accessed 11-March-2016].

Pantelis Elinas, Robert Sim, and James J. Little. σSLAM: Stereo vision SLAM using the Rao-Blackwellised particle filter and a novel mixture proposal distribution. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1564–1570. IEEE, 2006.

Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Computer Vision ECCV 2014*, pages 834–849. Springer, 2014.

Pablo F. Alcantarilla, Jesús Nuevo, and Adrien Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *British Machine Vision Conference (BMVC)*, September 2013.

Alex Flint, Christopher Mei, Ian Reid, and David Murray. Growing semantically meaningful models for visual SLAM. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 467–474. IEEE, 2010.

Terrence Fong, Clayton Kunz, Laura M. Hiatt, and Magda Bugajska. The human-robot interaction operating system. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 41–48. ACM, 2006. URL *http://dl.acm.org/citation.cfm?id=1121251*.

A.P. Gee, D. Chekhlov, A. Calway, and W. Mayol-Cuevas. Discovering Higher Level Structure in Visual SLAM. *IEEE Transactions on Robotics*, 24(5):980–990, October 2008.

Izrail Moiseevitch Gelfand and S Fomin. *Calculus of variations*. PRENTICE-HALL INTERNATIONAL, INC., October 2000.

Peter Gemeiner, Wolfgang Ponweiser, Peter Einramhof, and Markus Vincze. Real-time SLAM with a high-speed CMOS camera. In *Proc. 14th International Conference on Image Analysis and Processing ICIAP 2007.*, pages 297–302, 2007.

Arturo Gil, Oscar Reinoso, Oscar Martinez Mozos, Cyrill Stachniss, and Wolfram Burgard. Improving data association in vision-based SLAM. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2076–2081. IEEE, 2006.

Peng Gong, Yuksel Ozan Basciftci, and Fusun Ozguner. A Parallel Resampling Algorithm for Particle Filtering on Shared-Memory Architectures. In *Parallel and Distributed Processing Symposium Workshops And PhD Forum (IPDPSW)*, pages 1477–1483. IEEE, May 2012.

Neil J. Gordon, David J. Salmond, and Adrian FM Smith. Novel approach to non-linear/non-Gaussian Bayesian state estimation. In *Proc. IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113, 1993.

Scott Green, Mark Billinghurst, XiaoQi Chen, and Geoffrey Chase. Human-Robot Collaboration: A Literature Review and Augmented Reality Approach in Design. *International Journal of Advanced Robotic Systems*, pages 1–18, 2007.

H.-M. Gross, H.-J. Böhme, Christof Schröter, Steffen Mueller, Alexander König, Ch Martin, Matthias Merten, and Andreas Bley. Shopbot: Progress in developing an interactive mobile shopping assistant for everyday use. In *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on*. IEEE, 2008.

Fredrik Gustafsson. Particle Filter Theory and Practice with Positioning Applications. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):53–81, 2010.

Ji-Hyeong Han, Seung-Jae Lee, and Jong-Hwan Kim. Behavior Hierarchy-Based Affordance Map for Recognition of Human Intention and Its Application to Human-Robot Interaction. *IEEE Transactions on Human-Machine Systems*, pages 1–15, 2016.

Adam Herout, Markéta Dubská, and Jirí Havel. *Real-Time Detection of Lines and Grids*. SpringerBriefs in Computer Science. Springer London, London, 2013. ISBN 978-1-4471-4413-7, 978-1-4471-4414-4.

Jeroen D. Hol, Thomas B. Schon, and Fredrik Gustafsson. On resampling algorithms for particle filters. In *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*, pages 79–82. IEEE, 2006.

S.A. Holmes, G. Klein, and D.W. Murray. An O(N^2) Square Root Unscented Kalman Filter for Visual Simultaneous Localization and Mapping. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(7):1251–1263, July 2009.

Steven Holmes, Georg Klein, and David W. Murray. A square root unscented Kalman filter for visual monoSLAM. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3710–3716. IEEE, 2008.

Berthold KP Horn and Brian G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1):185–203, 1981.

Seo-Yeon Hwang and Jae-Bok Song. Monocular Vision-Based SLAM in Indoor Environment Using Corner, Lamp, and Door Features From Upward-Looking Camera. *IEEE Transactions on Industrial Electronics*, 58(10):4804–4812, October 2011.

Lim Hyon and Lee Young Sam. Real-time single camera SLAM using fiducial markers. In *Proc. ICROS-SICE International Joint Conference 2009*, Fukuoka International Congress Center, Japan, August 2009.

Sun Yong Kim Iickho Song, Jinsoo Bae. *Advanced Theory of Signal Detection: Weak Signal Detection in Generalized Observations (Signals and Communication Technology)*. Springer, 2002.

Jim Ivins and John Porrill. Everything you always wanted to know about snakes (but were afraid to ask). *Artificial Intelligence*, 2000, 1995.

Woo Yeon Jeong and Kyoung Mu Lee. Visual SLAM with Line and Corner Features. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2570 – 2575, Beijing, China., October 2006. IEEE.

Michael L. Johnson. *Essential Numerical Computer Methods*. Academic Press, 1st edition, 2010. ISBN 978-0-12-384997-7.

Jongdae Jung, Seung-Mok Lee, and Hyun Myung. Indoor Mobile Robot Localization and Mapping Based on Ambient Magnetic Fields and Aiding Radio Sources. *IEEE Transactions on Instrumentation and Measurement*, 64(7):1922–1934, July 2015.

Michael Kass and Andrew Witkin. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

Robert Keller. Example: Laplace equation & central difference, n.d. URL *http://www.cs.hmc.edu/~keller/courses/cs156/s98/slides/052.html*. [Online; accessed 11-March-2016].

Jong-Hwan Kim, Seung-Hwan Choi, In-Won Park, and Sheir Afgen Zaheer. Intelligence Technology for Robots That Think [Application Notes]. *IEEE Computational Intelligence Magazine*, 8(3):70–84, August 2013.

Jungho Kim, Kuk-Jin Yoon, Jun-Sik Kim, and Inso Kwe. Visual SLAM by single-camera catadioptric stereo. In *SICE-ICASE, 2006. International Joint Conference*, pages 2005–2009. IEEE, 2006. URL *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4109016*.

Alexander Kleiner, Christian Dornhege, and Sun Dali. Mapping disaster areas jointly: RFID-Coordinated SLAM by Hurnans and Robots. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6. IEEE, 2007.

Kurt Konolige and James Bowman. Towards lifelong visual maps. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 1156–1163. IEEE, 2009.

Markus Konrad. Parallel Computing for Digital Signal Processing on Mobile Device GPUs, March 2014. URL *https://www.researchgate.net/publication/262406322_Parallel_Computing_for_Digital_Signal_Processing_on_Mobile_Device_GPUs*. [Online; accessed 11-March-2016].

V. Krishnan. Planar curve evolution using B-Splines, 2013. URL *https://sites.google.com/site/skrishnanv/Home/research-1*. [Online; accessed 11-March-2016].

Abhijit Kundu, K. Madhava Krishna, and C. V. Jawahar. Realtime multibody visual SLAM with a smoothly moving monocular camera. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2080–2087. IEEE, 2011.

Toby HW Lam and Raymond ST Lee. Visual tracking by using Kalman Gradient Vector Flow (KGVF) snakes. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 557–563, 2004.

Bernard Lapeyre. Introduction to Monte Carlo methods, 2007. URL *http://cermics.enpc.fr/~bl/Halmstad/monte-carlo/lecture-1.pdf*. [Online; accessed 6-Aug-2016].

Chee Sing Lee, Sharad Nagappa, Narcis Palomeras, Daniel E. Clark, and Joaquim Salvi. SLAM with SC-PHD Filters: An Underwater Vehicle Application. *IEEE Robotics & Automation Magazine*, 21(2):38–45, June 2014.

Seung-Mok Lee, Jongdae Jung, Shin Kim, In-Joo Kim, and Hyun Myung. DV-SLAM (Dual-Sensor-Based Vector-Field SLAM) and Observability Analysis. *IEEE Transactions on Industrial Electronics*, 62(2):1101–1112, February 2015.

Sukhan Lee, Seongsoo Lee, Jeihun Lee, Dongju Moon, Eunyoung Kim, and Jeonghyun Seo. Robust recognition and pose estimation of 3d objects based on evidence fusion in a sequence of images. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3773–3779. IEEE, 2007.

Richard Lemaire and Simon Lacroix. Monocular-vision based SLAM using Line Segments. In *IEEE International Conference on Robotics and Automation, 2007*, pages 2791–2796, Rome, Italy, April 2007. IEEE.

John J. Leonard and Hugh F. Durrant-Whyte. Simultaneous Map Building and Localization for Autonomous Mobile Robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems 1991*, pages 1442–1447, Osaka, Japan, November 1991. IEEE.

Stefan Leutenegger, Margarita Chli, and Roland Yves Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

Yangming Li, Shuai Li, Quanjun Song, Hai Liu, and Max Q.-H. Meng. Fast and Robust Data Association Using Posterior Based Approximate Joint Compatibility Test. *IEEE Transactions on Industrial Informatics*, 10(1):331–339, February 2014.

Lixiong Liu and Alan C. Bovik. Active contours with neighborhood-extending and noise-smoothing gradient vector flow external force. *EURASIP Journal on Image and Video Processing*, 2012(1):1–6, 2012.

Stephanie Lowry, Gordon Wyeth, and Michael Milford. Odometry-driven inference to link multiple exemplars of a location. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 534–539. IEEE, 2013.

Ren C. Luo and Chun Chi Lai. Multisensor Fusion-Based Concurrent Environment Mapping and Moving Object Detection for Intelligent Service Robotics. *IEEE Transactions on Industrial Electronics*, 61(8):4043–4051, August 2014.

Dubská et all Markéta. Real-Time Detection of Lines using Parallel Coordinates and OpenGL. *Proceedings of the 27th Spring Conference on Computer Graphics*, pages 149 –155, 2011.

John H. Mathews and Kurtis K. Fink. *Numerical Methods Using Matlab*. Prentice-Hall Inc, 2004.

Michael Milford, Gordon Wyeth, and David Prasser. RatSLAM on the edge: revealing a coherent representation from an overloaded rat brain. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4060–4065. IEEE, 2006.

M.J. Milford and G.F. Wyeth. Mapping a Suburb With a Single Camera Using a Biologically Inspired SLAM System. *IEEE Transactions on Robotics*, 24(5):1038–1053, October 2008.

Paul Milgram, Shijun Zhai, David Drascic, and Julius J. Grodski. Applications of augmented reality for human-robot communication. In *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 3, pages 1467–1472. IEEE, 1993.

Mohammad Hossein Mirabdollah and Barbel Mertsching. Monocular SLAM: using trapezoids to model landmark uncertainties. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 482–488. IEEE, 2012.

Armaghan Moemeni and Eric Tatham. Inertial-visual pose tracking using optical flow-aided particle filtering. In *Computational Intelligence for Multimedia, Signal and Vision Processing (CIMSIVP), 2014 IEEE Symposium on*, pages 1–8. IEEE, 2014.

Michael Montemerlo and Sebastian Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. In *Proc. IEEE International Conference on Robotics and Automation, Proceedings. ICRA*, volume 2, pages 1985–1991, 2003.

J. M. M. Montiel, Javier Civera, and Andrew J. Davison. Inverse depth parametrization for monocular SLAM. *IEEE Transactions on Robotics*, pages 932–945, October 2008.

Diluka Moratuwage, Danwei Wang, Akshay Rao, Namal Senarathne, and Han Wang. RFS Collaborative Multivehicle SLAM: SLAM in Dynamic High-Clutter Environments. *IEEE Robotics & Automation Magazine*, 21(2):53–59, June 2014.

R. Munguia and A. Grau. Camera localization and mapping using delayed feature initialization and inverse depth parametrization. *IEEE Conf. Emerging Technologies and Factory Automation*, pages 981–988, September 2007.

Paul Newman, David Cole, and Kin Ho. Outdoor SLAM using visual appearance and laser ranging. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1180–1187. IEEE, 2006. URL *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1641869*.

Kazunori Ohno, Takafumi Nomura, and Satoshi Tadokoro. Real-time robot trajectory estimation and 3d map construction using 3d camera. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5279–5285. IEEE, 2006.

OpenCV. OpenCV open source computer vision, n.d. URL *http://opencv.org/*. [Online; accessed 11-March-2016].

L.M. Paz, P. Pinies, J.D. Tardos, and J. Neira. Large-Scale 6-DOF SLAM With Stereo-in-Hand. *IEEE Transactions on Robotics*, 24(5):946–957, October 2008.

L. Pedraza, D. Rodriguez-Losada, F. Matia, G. Dissanayake, and J.V. Miro. BS-SLAM: shaping the world. In *Proceedings of Robotics: Science and Systems*, June 2007.

L. Pedraza, D. Rodriguez-Losada, F. Matia, G. Dissanayake, and J.V. Miro. Extending the limits of feature-based SLAM with b-splines. *IEEE Transactions on Robotics*, 25(2):353–366, April 2009.

Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(7):629–639, 1990.

Natan Peterfreund. The velocity snake. In *Proc. IEEE Nonrigid and Articulated Motion Workshop*, pages 70–79, 1997.

Natan Peterfreund. Robust tracking of position and velocity with Kalman snakes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(6):564–569, 1999.

Pedro Piniés, Todd Lupton, Salah Sukkarieh, and Juan D. Tardós. Inertial aiding of inverse depth SLAM using a monocular camera. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2797–2802. IEEE, 2007.

Vivek Pradeep, Gerard Medioni, and James Weiland. Visual loop closing using multi-resolution SIFT grids in metric-topological SLAM. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1438–1445. IEEE, 2009.

Mark Pupilli and Andrew Calway. Real-Time Camera Tracking Using a Particle Filter. In *BMVC*, 2005. URL *http://www.cse.iitk.ac.in/users/vision/dipakmj/papers/2000377.pdf*.

Mark Pupilli and Andrew Calway. Real-time visual SLAM with resilience to erratic motion. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1244–1249. IEEE, 2006.

Shehrzad Qureshi. Computer vision acceleration using GPUs, 2011. URL *http://developer.amd.com/wordpress/media/2013/06/2162_final.pdf*. [Online; accessed 11-March-2016].

Fabio T. Ramos, Juan Nieto, and Hugh F. Durrant-Whyte. Recognising and modelling landmarks to close loops in outdoor SLAM. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2036–2041. IEEE, 2007.

Ananth Ranganathan and Jongwoo Lim. Visual place categorization in maps. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3982–3989. IEEE, 2011.

Dushyant Rao, Soon-Jo Chung, and Seth Hutchinson. CurveSLAM: an approach for vision-based navigation without point features. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4198–4204, 2012.

Oscar Reinoso, Luis Miguel Jimenez, Lorenzo Fernández, and Luis Payá. Appearance-based approach to hybrid metric-topological simultaneous localisation and mapping. *IET Intelligent Transport Systems*, 8(8):688–699, December 2014.

Gerhard Reitmayr, Ethan Eade, and Tom W. Drummond. Semi-automatic annotations in unknown environments. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 67–70. IEEE, 2007.

David F Rogers. *An introduction to NURBS with historical perspective.* Morgan Kaufmann Publishers, San Francisco, 2001. ISBN 9781558606692 1558606696. URL *http://www.sciencedirect.com/science/book/9781558606692*.

Roguebasin. Bresenham's line algorithm, n.d. URL *http://www.roguebasin.com/index.php?title=Bresenham%27s_Line_Algorithm*. [Online; accessed 11-March-2016].

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision ICCV*, pages 2564–2571. IEEE, 2011.

Simo Särkkä. *Bayesian filtering and smoothing.* Cambridge University Press, 2013. URL *http://becs.aalto.fi/~ssarkka/pub/cup_book_online_20131111.pdf*.

Simo Särkkä, Jouni Hartikainen, Isambi Sailon, and Mbalawata Heikki Haario. Computing posterior inference on parameters of stochastic differential equations via non-linear Gaussian filtering and adaptive MCMC, n.d. URL *https://users.aalto.fi/~ssarkka/pub/cdgf-param.pdf*. [Online; accessed 6-Aug-2016].

Hanno Scharr. *Optimal Operators in Digital Image Processing.* PhD thesis, Rupertus Carola University of Heidelberg, May 2000. URL *http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:bsz:16-opus-9622*. Table B.11.

David Schleicher, Luis Miguel Bergasa, Rafael Barea, Elena Lopez, and Manuel Ocana. Real-time simultaneous localization and mapping using a wide-angle stereo camera and adaptive patches. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2090–2095, 2006.

Jean C. Scholtz. Human-robot interactions: Creating synergistic cyber forces. In *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 177–184. Springer, 2002.

Christof Schroeter and Horst-Michael Gross. A sensor-independent approach to RBPF SLAM-Map Match SLAM applied to Visual Mapping. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2078–2083. IEEE, 2008.

Jianbo Shi and Carlo Tomasi. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

B. G. Sileshi, C. Ferrer, and J. Oliver. Particle filters and resampling techniques: Importance in computational complexity analysis. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, pages 319–325. IEEE, 2013a.

B. G. Sileshi, C. Ferrer, and J. Oliver. Particle filters and resampling techniques: Importance in computational complexity analysis. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*, pages 319–325. IEEE, 2013b.

Fabio Silveira Vidal, Andre de Oliveira Palmerim Barcelos, and Paulo Fernando Ferreira Rosa. SLAM solution based on particle filter with outliers filtering in dynamic environments. In *Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on*, pages 644–649. IEEE, 2015.

Robert Sim, Pantelis Elinas, Matt Griffin, Alex Shyr, and James J. Little. Design and analysis of a framework for real-time vision-based SLAM using Rao-Blackwellised particle filters. In *Computer and Robot Vision, 2006. The 3rd Canadian Conference on*, pages 21–21. IEEE, 2006.

Graham Singer. The history of the modern graphics processor, 2013. URL *http://www.techspot.com/article/650-history-of-the-gpu/*. [Online; accessed 11-March-2016].

Per Skoglar and David Törnqvist. Simultaneous camera orientation estimation and road target tracking. In *Information Fusion (FUSION), 2012 15th International Conference on*, pages 802–807. IEEE, 2012.

Iryna Skrypnyk and David G. Lowe. Scene modelling, recognition and tracking with invariant image features. In *Proc. Third IEEE and ACM International Symposium on Mixed and Augmented Reality.*, pages 110–119, 2004.

Erik Smistad. GPU gradient vector flow using OpenCL, 2013. URL *http://www.thebigblob.com/gpu-based-gradient-vector-flow-using-opencl/*. [Online; accessed 11-March-2016].

Erik Smistad, Anne C. Elster, and Frank Lindseth. Real-time gradient vector flow on GPUs using OpenCL. *Real-Time Image Processing*, June 2012.

Christopher E. Smith. Fast tracking of natural textures using fractal snakes. In *Proc. IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2175–2181, 2010.

Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proc. Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, pages 267–288, Corvallis, Oregon, 1986. AUAI Press.

OpenCV Sobel. OpenCV sobel derivatives, n.d. URL *http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html*. [Online; accessed 11-March-2016].

Joan Sola, André Monin, Michel Devy, and Teresa Vidal-Calleja. Fusing monocular information in multicamera SLAM. *Robotics, IEEE Transactions on*, 24(5):958–968, 2008.

V. Srikrishnan and S. Chaudhuri. Stabilization of parametric active contours using a tangential redistribution term. *IEEE Transactions on Image Processing*, 18(8): 1859–1872, August 2009.

Olivier Stasse, Andrew J. Davison, Ramzi Sellaouti, and Kazuhito Yokoi. Real-time 3d SLAM for humanoid robot considering pattern generator information. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 348–355. IEEE, 2006.

Bor-Yiing Su. *Parallel Application Library for Object Recognition*. PhD thesis, University of California, Berkeley, 2012. URL *http://digitalassets.lib.berkeley.edu/etd/ucb/text/Su_berkeley_0028E_12815.pdf*.

Ganesh Sundaramoorthi, Jeremy D. Jackson, Anthony Yezzi, and Andrea C. Mennucci. Tracking with Sobolev active contours. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 674–680, 2006.

Mehmet Süzen. Particle approximation to probability density functions: Dirac delta function representation, 2014. URL *http://science-memo.blogspot.co.uk/2014/01/particle-approximation-to-probability.html*. [Online; accessed 12-Aug-2016].

Jean-Philippe Tardif, Yanis Pavlidis, and Kostas Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2531–2538. IEEE, 2008.

D. Terzopoulos and Richard Szeliski. Tracking with Kalman snakes. In *Active Vision*, pages 3–20. MIT Press, 1992.

Masahiro Tomono. Monocular slam using a Rao-Blackwellised particle filter with exhaustive pose space search. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2421–2426. IEEE, 2007.

Panagiota Tsarouchi, Sotiris Makris, and George Chryssolouris. Human robot interaction review and challenges on task planning and programming, 2016. URL *http://dx.doi.org/10.1080/0951192X.2015.1130251*. [International Journal of Computer Integrated Manufacturing, Online; accessed 27-April-2016].

Teresa Vidal-Calleja, Andrew J. Davison, Juan Andrade-Cetto, and David W. Murray. Active control for single camera SLAM. In *Proc. IEEE International Conference on Robotics and Automation ICRA*, pages 1930–1936, 2006.

Diego Viejo and Miguel Cazorla. 3d plane-based egomotion for SLAM on semi-structured environment. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2761–2766. IEEE, 2007.

Yingxu Wang, Edmund T. Rolls, Newton Howard, Victor Raskin, Witold Kinsner, Fionn Murtagh, Virendrakumar C. Bhavsar, Shushma Patel, Dilip Patel, and Duane F. Shell. Cognitive informatics: From information revolution to intelligence revolution. *International Journal of Software Science and Computational Intelligence*, 7(2):52–71, 2015.

Wikipedia. Fundamental lemma of calculus of variations, n.d. URL *https://en.wikipedia.org/wiki/Calculus_of_variations*. [Online; accessed 21-Jul-2016].

C. L. Winter. Normalized mahalanobis distance for comparing process-based stochastic models. *Stochastic Environmental Research and Risk Assessment*, 24 (6):917–923, March 2010.

Hao Wu and Xuan Zhang. Active contours with extended neighborhood generalized gradient vector flow external force. In *Proc. 5th International Congress on Image and Signal Processing*, pages 639–642, 2012.

Chenyang Xu and Jerry L. Prince. Generalized gradient vector flow external forces for active contours. *Signal Processing*, 71(2):131–139, 1998a.

Chenyang Xu and Jerry L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998b.

Paul Zarchan and Howard Musoff. *Fundamentals of Kalman filtering a practical approach*. American Institute of Aeronautics and Astronautics, Inc., 3rd edition, 2009. ISBN 978-1-60086-718-7.

Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. IEEE, 1999.