UNIVERSITY OF NAVARRA

ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES

DONOSTIA-SAN SEBASTIÁN

A Stochastic Parallel Method for Real Time Monocular SLAM Applied to Augmented Reality

DISSERTATION
submitted for the
Degree of Doctor of Philosophy
of the University of Navarra by

Jairo R. Sánchez Tapia

Dec, 2010

A mis abuelos.

Agradecimientos

Lo lógico hubiese sido que la parte que más me costara escribir de la Tesis fuera la que la gente fuese a leer con más atención (el capítulo que desarrolla el método propuesto, por supuesto). Por eso resulta un tanto desconcertante que me haya costado tanto escribir esta parte…

Me gustaría empezar agradeciendo a Alejo Avello, Jordi Viñolas y Luis Matey la confianza que depositaron en mi al aceptarme como doctorando en el área de simulación del CEIT, así como a todo el personal de administración y servicios por facilitar tanto mi trabajo. Del mismo modo, agradezco a la Universidad de Navarra, y especialmente a la Escuela de Ingenieros TECNUN el haber aceptado y renovado año tras año mi matrícula de doctorado, así como la formación que me ha dado durante este tiempo. No me olvido de la Facultad de Informática de la UPV/EHU, donde empecé mi andadura universitaria, y muy especialmente a su actual Vicedecano y amigo Alex García-Alonso por su apoyo incondicional.

Agradezco también a mi tutor de prácticas, director de proyecto, director de Tesis y amigo Diego Borro toda su ayuda. Han sido muchos años hasta llegar aquí, en los que además de trabajo, hemos compartido otras muchas cosas.

La verdad es que como todo doctorando sabe (o sabrá), los últimos meses de esta peregrinación se hacen muy duros. El concepto de tiempo se desvanece, y la definición de fin de semana pasa a ser "esos días en los que sabes que nadie te va a molestar en el despacho". Sin embargo, dicen que con pan y vino se anda el camino, o en mi caso con Carlos y con Gaizka, que además de compartir este último verano conmigo, también me han acompañado durante todo el viaje. Gracias y V's.

Me llevo un especial recuerdo de los tiempos de *La Crew*. En aquella época asistí y participé en la creación de conceptos tan innovadores como la marronita, las torrinas o las simulaciones del mediodía, que dificilmente serán olvidados. A todos vosotros os deseo una vida llena de películas aburridas.

Como lo prometido es deuda, aquí van los nombres de todos los que colaboraron en el proyecto *Una Teja por Una Tesis*: Olatz, Txurra, Ane A., Julián, Ainara B., Hugo, Goretti, Tomek, Aiert, Diego, Mildred, Maite, Gorka, CIBI, Ilaria, Borja, Txemi, Eva, Alex, Gaizka, Imanol, Álvaro, Iñaki B., Denix, Josune, Aitor A., Javi B., Nerea E., Alberto y Fran.

También me gustaría mencionar a todas las criaturas que han compartido mazmorra conmigo, aunque algunos aparezcan repetidos: Aitor "*no me da ninguna pena*" Cazón, Raúl "*susurros*" de la Riva, Carlos "*Ignacio*" Buchart, Hugo "*estoy muy loco*" Álvarez, Gaizka "*no me lies*" San Vicente, Goretti "*tengo cena*" Etxegaray, Ilaria "*modo gota*" Pasciuto, Luis "*apestati*" Unzueta, Diego "*que vergüenza*" Borro, Maite "*cállate Carlos*" López, Iker "visitación" Aguinaga, Ignacio M., Aitor O., Alex V., Yaiza V., Oskar M., Javi Barandiarán, Iñaki G., Jokin S., Iñaki M., Denix, Imanol H., Álvaro B., Luis A., Gorka V., Borja O., Aitor A., Nerea E., Virginia A., Javi Bores, Ibon E., Xabi S., Imanol E., Pedro A., Fran O. y Meteo R.

Entre estas líneas también hay hueco para los personajes de mi bizarra cuadrilla, a.k.a.: Xabi, Pelopo, Iker, Olano, Gonti, Peña, Gorka, Julen, Lope, Maiz, y los Duques Inaxio y Etx.

And last but not least, agradezco a mi familia, y muy especialmente a mis padres Emilio y Maria Basilia, y a mi hermano Emilio José todo su apoyo. Por supuesto también meto aquí a Jose Mari y a Montse (aunque me pongan pez para comer), a Juan, Ane y a Jon. De todas formas, en este apartado la mención honorífica se la lleva Eva por todo lo que ha aguantado, sobre todo durante las últimas semanas que he "no-estado".

A todos vosotros y a los que he olvidado poner,

Qatlho'!

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Vectors are represented with a hat $\vec{a}$, matrices in boldface capital letter $\mathbf{A}$, scalars in italic face a and sets in uppercase $\mathtt{S}$. Unless otherwise stated, subindices are used to reference the components of a vector $\vec{a} = [a_1, a_2, \ldots, a_n]^\top$ or for indexing the elements of a set $\mathtt{S} = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$. Superindices are used to refer a sample drawn from a random variable $x^{(i)}$.

$\vec{M}$:	A world point (4-vector)
$\vec{m}$:	A point in the image plane (3-vector)
$\vec{0}_n$:	The zero vector (n-vector)
$\mathbf{I}_n$:	The $n \times n$ identity matrix
$\vec{x}_k$:	State vector at instant k of a space-state system
$\hat{x}_k$:	An estimator of the vector $\vec{x}_k$
$\mathtt{X}_k$:	The set of state vectors of a space-state system in instant k
$\mathtt{X}_{i:k}$:	The set of state vectors of a space-state system from instant i to instant k
$\vec{y}_k$:	Measurement vector at instant k of a state-space system
$\mathtt{Y}_k$:	The set of measurement vectors of a space-state system up to instant k

Abstract

In augmented reality applications, the position and orientation of the observer must be estimated in order to create a virtual camera that renders virtual objects aligned with the real scene. There are a wide variety of motion sensors available in the market, however, these sensors are usually expensive and impractical. In contrast, computer vision techniques can be used to estimate the camera pose using only the images provided by a single camera if the 3D structure of the captured scene is known beforehand. When it is unknown, some solutions use external markers, however, they require to modify the scene, which is not always possible.

Simultaneous Localization and Mapping (SLAM) techniques can deal with completely unknown scenes, simultaneously estimating the camera pose and the 3D structure. Traditionally, this problem is solved using nonlinear minimization techniques that are very accurate but hardly used in real time. In this way, this thesis presents a highly parallelizable random sampling approach based on Monte Carlo simulations that fits very well on the graphics hardware. As demonstrated in the text, the proposed algorithm achieves the same precision as nonlinear optimization, getting real time performance running on commodity graphics hardware.

Along this document, the details of the proposed SLAM algorithm are analyzed as well as its implementation in a GPU. Moreover, an overview of the existing techniques is done, comparing the proposed method with the traditional approach.

Resumen

En las aplicaciones de realidad aumentada es necesario estimar la posición y la orientación del observador para poder configurar una cámara virtual que dibuje objetos virtuales alineados con los reales. En el mercado existe una gran variedad de sensores de movimiento destinados a este propósito. Sin embargo, normalmente resultan caros o poco prácticos. En respuesta a estos sensores, algunas técnicas de visión por computador permiten estimar la posición de la cámara utilizando únicamente las imágenes capturadas, si se conoce de antemano la estructura 3D de la escena. Si no se conoce, existen métodos que utilizan marcadores externos. Sin embargo, requieren intervenir el entorno, cosa que no es siempre posible.

Las técnicas de *Simultaneous Localization and Mapping* (SLAM) son capaces de hacer frente a escenas de las que no se conoce nada, pudiendo estimar la posición de la cámara al mismo tiempo que la estructura 3D de la escena. Normalmente, estos métodos se basan en técnicas de optimización no lineal que, siendo muy precisas, resultan difíciles de implementar en tiempo real. Sobre esta premisa, esta Tesis propone una solución altamente paralelizable basada en el método de Monte Carlo diseñada para ser ejecutada en el hardware gráfico del PC. Como se demuestra, la precisión que se obtiene es comparable a la de los métodos clásicos de optimización no lineal, consiguiendo tiempo real utilizando hardware gráfico de consumo.

A lo largo del presente documento, se detalla el método de SLAM propuesto, así como su implementación en GPU. Además, se expone un resumen de las técnicas existentes en este campo, comparándolas con el método propuesto.

Part I

Introduction

Chapter 1
Introduction

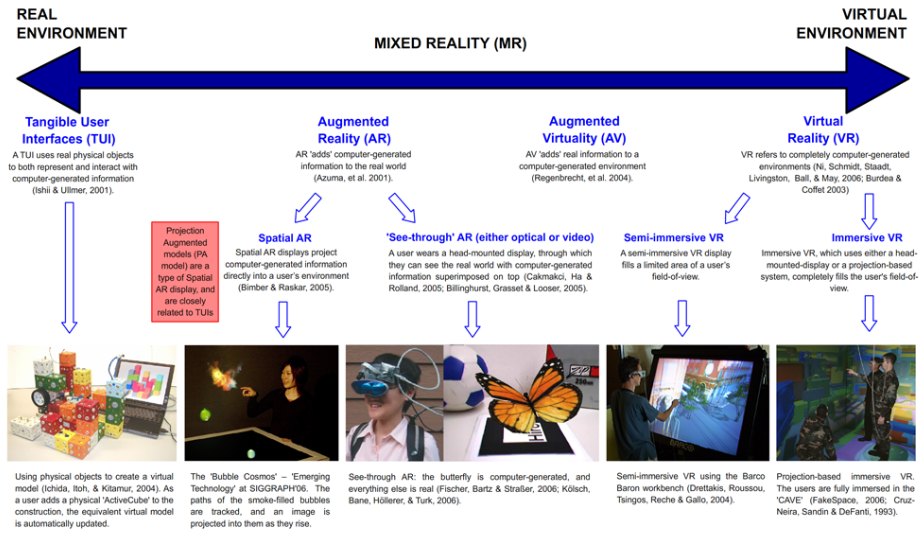
Part of this chapter has been presented in:

> *Barandiaran, J., Moreno, I., Ridruejo, F. J., Sánchez, J. R., Borro, D., and Matey, L. "Estudios y Aplicaciones de Realidad Aumentada en Dispositivos Móviles". In Conferencia Española de Informática Gráfica (CEIG'05), pp. 241–244. Granada, Spain. 2005.*

1.1 Motivation

Facing the success of the virtual reality in industrial and entertainment applications, augmented reality is emerging as a new way to visualize information. It consists on combining virtual objects with real images taken from the physical world, aligning them along a video sequence, creating the feeling that both real and virtual objects coexist. This combination enriches the information that the user perceives, and depending on the amount of virtual objects added, (Milgram and Kishino, 1994) propose the Reality-Virtuality continuum shown in Figure 1.1, spanning from a real environment to a completely virtual world.

The elements that compound an augmented reality system are a camera that captures the real world, a computer and a display where the virtual objects and the images are rendered. The main challenge is to track the position and the orientation of the observer. Using this information, a virtual camera can be configured, so that the virtual objects drawin with it align with the real scene. However, there are other problems like illumination estimation and occlusions



A TUI uses real physical objects to both represent and interact with computer-generated information (Ishii & Ullmer, 2001).

AR 'adds' computer-generated information to the real world (Azuma, et al. 2001).

AV 'adds' real information to a computer-generated environment (Regenbrecht, et al. 2004).

VR refers to completely computer-generated environments (Ni, Schmidt, Staadt, Livingston, Ball, & May, 2006; Burdea & Coffet 2003)

Projection Augmented models (PA model) are a type of Spatial AR display, and are closely related to TUIs

Spatial AR displays project computer-generated information directly into a user's environment (Bimber & Raskar, 2005).

A user wears a head-mounted display, through which they can see the real world with computer-generated information superimposed on top (Cakmakci, Ha & Rolland, 2005; Billinghurst, Grasset & Looser, 2005).

A semi-immersive VR display fills a limited area of a user's field-of-view.

Immersive VR, which uses either a head-mounted-display or a projection-based system, completely fills the user's field-of-view.

Using physical objects to create a virtual model (Ichida, Itoh, & Kitamur, 2004). As a user adds a physical 'ActiveCube' to the construction, the equivalent virtual model is automatically updated.

The 'Bubble Cosmos' – 'Emerging Technology' at SIGGRAPH'06. The paths of the smoke-filled bubbles are tracked, and an image is projected into them as they rise.

See-through AR: the butterfly is computer-generated, and everything else is real (Fischer, Bartz & Straßer, 2006; Kölsch, Bane, Höllerer, & Turk, 2006).

Semi-immersive VR using the Barco Baron workbench (Drettakis, Roussou, Tsingos, Reche & Gallo, 2004).

Projection-based immersive VR. The users are fully immersed in the 'CAVE' (FakeSpace, 2006; Cruz-Neira, Sandin & DeFanti, 1993).

Figure 1.1: Based on the Milgram Reality-Virtuality continuum. [1]

that have to be solved in order to get a realistic augmentation.

Augmented reality started becoming popular in the nineties when the term was first coined by (Caudell and Mizell, 1992). However, the idea was not new. Ivan Sutherland built a prototype in the sixties consisting of a head mounted display that is considered the first work on augmented reality (Sutherland, 1968).

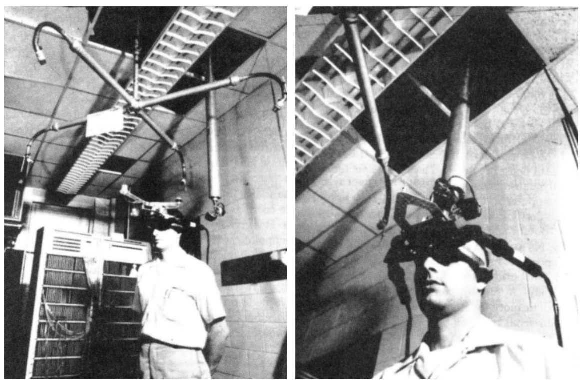


Figure 1.2: The tracking systems used by Sutherland. (Sutherland, 1968)

[1]This image is licensed under the Creative Commons Attribution-ShareAlike 3.0 License.

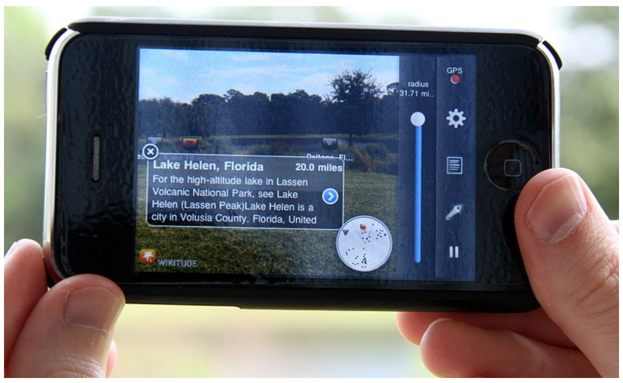
(a) Apple iPhone. (©Apple Inc.) (b) Google Nexus One. (©Google)

Figure 1.3: Smartphones equipped with motion sensors.

As illustrated in Figure 1.2, he tried both mechanical and ultrasonic sensors in order to track the position of the head.

Recently, thanks to the introduction of smartphones, augmented reality is starting to break into the masses. These devices (Figure 1.3) have a variety of sensors like GPS, accelerometers, compasses, etc. that combined with the camera can measure the pose of the device precisely using limited amounts of computational power.

An example of a smartphone application is Wikitude. It displays information about the location where the user is using a geolocated online database. The tracking is done using the data coming from the GPS, the compass and the accelerometers and the output is displayed overlaid with the image given by the camera (Figure 1.4). It runs in Apple iOS, Android and Symbian platforms.



Lake Helen, Florida 20.0 miles
For the high-altitude lake in Lassen Volcanic National Park, see Lake Helen (Lassen Peak)Lake Helen is a city in Volusia County, Florida, United

Figure 1.4: Screenshot of Wikitude.

The gaming industry has also used this technology in some products. Sony has launched an augmented reality game for its PSP platform developed by Novarama called Invizimals. As shown in Figure 1.5, the tracking is done using a square marker very similar to the markers used in ARToolkit (Kato and Billinghurst, 1999). In contrast to the previous example, there are not any sensors that measure directly the position or the orientation of the camera. Instead, using computer vision techniques, the marker is segmented from the image so that the pose can be recovered using some geometric restrictions applied to the shape of the marker. This solution is very fast and precise, and only uses the images provided by the camera. However, it needs to modify the scene with the marker, which is not possible in some other applications.

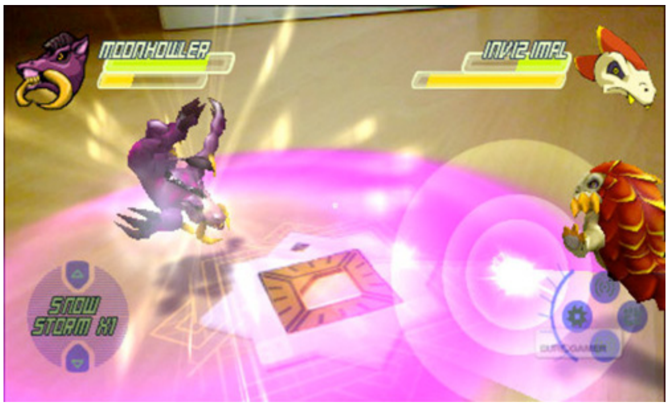


Figure 1.5: Screenshot of Invizimals (©Sony Corp.).

Other different type of marker based optical tracking includes the work of (Henderson and Feiner, 2009). A system oriented to industrial maintenance tasks is proposed and its usability is analyzed. It tracks the viewpoint of the worker using passive reflective markers that are illuminated by infrared light. These markers are detected by a set of cameras that determine the position and the orientation of the user. The infrared light ease the detection of the markers, benefiting the robustness of the system. As shown in Figure 1.6, this allows to display virtual objects in a Head Mounted Display (HMD) that the user wears.

In addition to these applications, there are other fields like medicine, cinema or architecture that can also benefit from this technology and its derivatives, providing it a promising future.

However, the examples shown above have the drawback of needing special hardware or external markers in order to track the pose of the camera. In contrast,

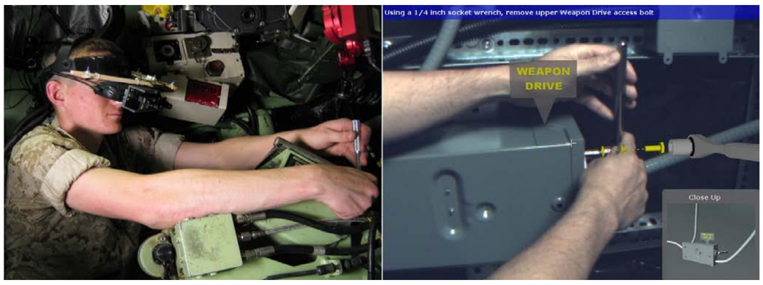

Using a 1/4 inch socket wrench, remove upper Weapon Drive access bolt
WEAPON DRIVE
Close Up

Figure 1.6: ARMAR maintenance application (Henderson and Feiner, 2009).

markerless monocular systems are able to augment the scene using only images without requiring any external markers. These methods exploit the effect known as motion parallax derived from the apparent displacement of the objects due to the motion of the camera. Using computer vision techniques this displacement can be measured, and applying some geometric restrictions, the camera pose can be automatically recovered. The main drawback is that the accuracy achieved by these solutions cannot be compared with marker based systems. Moreover, existing methods are very slow, and in general they can hardly run in real time.

In recent years, the use of high level programming languages for the Graphics Processor Unit (GPU) is being popularized in many scientific areas. The advantage of using the GPU rather than the CPU, is that the first has several computing units that can run independently and simultaneously the same task on different data. These devices work as data-streaming processors (Kapasi et al., 2003) following the SIMD (Single Instruction Multiple Data) computational model.

Thanks to the gaming industry, these devices are present in almost any desktop computer, providing them massively parallel processors with a very large computational power. This fact gives the opportunity to use very complex algorithms in user oriented applications, that previously could only be used in scientific environments.

Anyway, each computing unit has less power than a single CPU, so the benefit of using the GPU comes when the size of the input data is large enough. For this reason, it is necessary to adapt existing algorithms and data structures to fit this programming model.

Regarding to augmented reality, as stated before, the complexity of monocular tracking systems makes them difficult to use in real time applications. For this reason, it would be very useful to take the advantage of GPUs in this field. However, existing monocular tracking algorithms are not suitable to be implemented on the GPU because of their batch nature. In this way, this thesis studies the monocular tracking problem and proposes a method that can achieve a high level of accuracy in real time using efficiently the graphics hardware. By doing so, the algorithm tracks the camera without leaving the computer unusable for other tasks.

1.2 Contributions

The goal of this thesis is to track the motion of the camera in a video sequence and to obtain a 3D reconstruction of the captured scene, focused on augmented reality applications, without using any external sensors and without having any markers or knowledge about the captured scene. In order to get real time performance, the graphics hardware has been used. In this context, instead of adapting an existing method, a fully parallelizable algorithm for both camera tracking and 3D structure reconstruction based on Monte Carlo simulations is proposed, being the main contributions:

- The design and implementation of a GPU friendly 3D motion tracker that can estimate both position and orientation of a video sequence frame by frame using only the captured images. A GPU implementation is analyzed in order to validate its precision and its performance using a handheld camera, demonstrating that it can work in real time and without the need of user interaction.

- A 3D reconstruction algorithm that can obtain the 3D structure of the captured scene at the same time that the motion of the camera is estimated. It is also executed in the GPU adjusting in real time the position of the 3D points as new images become available. Moreover, new points are continuously added in order to reconstruct unexplored areas while the rest are recursively used by the motion estimator.

- A computationally efficient 3D point remapping technique based on a simplified version of SIFT descriptors. The remapping is done using temporal coherence between the projected structure and image measurements comparing the SIFT descriptors.

- A local optimization framework that can be used to solve efficiently in the GPU any parameter estimation problem driven by an energy function. Providing an initial guess about the solution, the parameters are found using the graphics hardware in a very efficient way. These problems are very common in computer vision and are usually the bottleneck of real time systems.

1.3 Thesis Outline

The content of this document is divided in five chapters. Chapter 1 has introduced the motivation of the present work. Chapter 2 will introduce some preliminaries and previous works done by other authors in markerless monocular tracking and reconstruction. Chapter 3 describes the proposed method including some experimental results in order to validate it. Chapter 4 deals with the GPU implementation details, presenting a performance analysis that includes a comparison with other methods. Finally, Chapter 5 presents the conclusions derived from the work and proposes future research lines.

Additionally, various appendices have been included clarifying some concepts needed to understand this document. Appendix A introduces some concepts on projective geometry, including a brief description of a self-calibration technique. Appendix B explains the basics about epipolar geometry. Appendix C introduces the architecture of the GPU viewed as a general purpose computing device and finally Appendix D enumerates the list of publications generated during the development of this thesis.

Chapter 2
Background

This chapter describes the background on camera tracking and the mathematical tools needed for understanding this thesis. Some aspects about the camera model and its parameterization are exposed and the state of the art in camera calibration and tracking is analyzed.

Part of this chapter has been presented in:

Sánchez, J. R. and Borro, D. "Automatic Augmented Video Creation for Markerless Environments". In Poster Proceedings of the 2nd International Conference on Computer Vision Theory and Applications (VISAPP'07), pp. 519–522. Barcelona, Spain. 2007.

Sánchez, J. R. and Borro, D. "Non Invasive 3D Tracking for Augmented Video Applications". In IEEE Virtual Reality 2007 Conference, Workshop "Trends and Issues in Tracking for Virtual Environments", pp. 22–27. Charlotte, NC, USA. 2007.

2.1 Camera Geometry

The camera is a device that captures a three dimensional scene and projects it into the image plane. It has two major parts, the lens and the sensor, and depending on the properties of them an analytical model describing the image formation process can be inferred.

As the main part of an augmented reality system, it is important to know accurately the parameters that define the camera. The quality of the augmentation perception depends directly on the similarity between the real camera and the virtual camera that is used to render the virtual objects.

The pin-hole camera is the simplest representation of the perspective projection model and can be used to describe the behavior of a real camera. It assumes a camera with no lenses whose aperture is described by a single point, called the center of projection, and a plane where the image is formed. Light rays passing through the hole form an inverted image of the object as shown in Figure 2.1. The validity of this approximation is directly related to the quality of the camera, since it does not include some geometric distortions induced by the lenses of the camera.

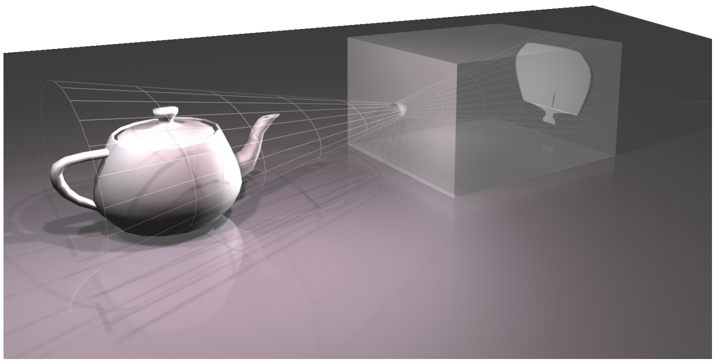


Figure 2.1: Light rays passing through the pin-hole.

Because of its simplicity, it has been widely used in computer vision applications to describe the relationship between a 3D point and its corresponding 2D projection onto the image plane. The model assumes a perspective projection, i.e., objects seem smaller as their distance from the camera increases.

Renaissance painters found themselves interested by this fact and started studying the perspective projection in order to reproduce realistic paintings of the world that they were observing. They used a device called *perspective machine* in order to calculate the effects of the perspective before having its mathematical model. These machines follow the same principle as the pin-hole cameras as can be seen in Figure 2.2.

More formally, a pin-hole camera is described by an image plane situated at a non-zero distance f of the center of projection $\vec{C}$, also known as optical center that corresponds with the pin-hole. The image (x, y) of a world point (X, Y, Z) is the intersection of the line joining the optical center and the point with the image

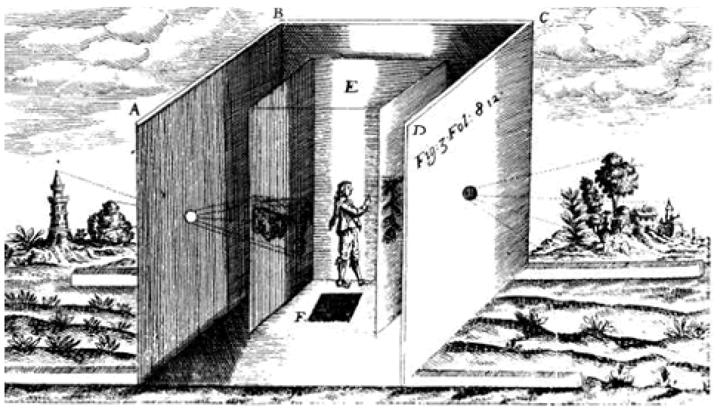


Figure 2.2: Athanasius Kircher's camera obscura. [1]

plane. As shown in Figure 2.3, if the optical center is assumed to be behind the camera, the formed image is no longer inverted. There is a special point called principal point, defined as the intersection between the ray passing through the optical center perpendicular to the image plane.

If an orthogonal coordinate system is fixed at the optical center with the z axis perpendicular to the image plane, the projection of a point can be obtained by similar triangles:

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}. \tag{2.1}$$

These equations can be linearized if points are represented in projective space using its homogeneous representation. An introduction to projective geometry can be found in the Appendix A. In this case, Equation 2.1 can be expressed by a 4×3 matrix **P** called the projection matrix:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{P}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2.2}$$

In the simplest case, when the focal length is $f = 1$, the camera is said to be

[1]Drawing from "Ars Magna Lucis Et Umbra", 1646.

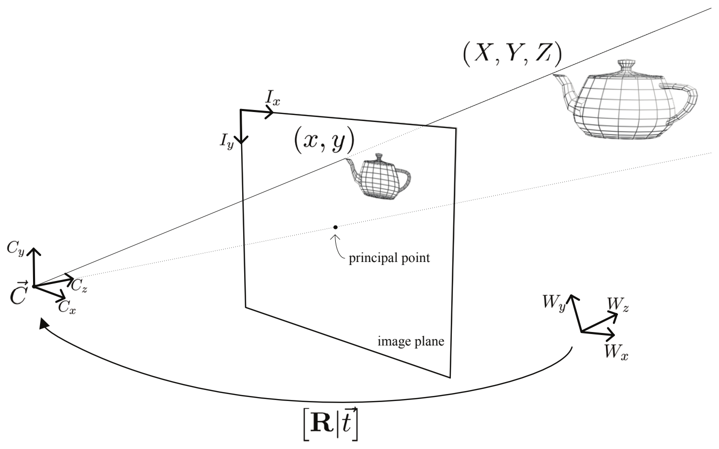


Figure 2.3: Perspective projection model.

normalized or centered on its canonical position. In this situation the normalized pin-hole projection function Π is defined as:

$$\Pi : \mathbb{P}^3 \rightarrow \mathbb{P}^2, \quad \vec{m} = \Pi\left(\vec{M}\right). \tag{2.3}$$

However, the images obtained from a digital camera are described by a matrix of pixels whose coordinate system is located on its top left corner. Therefore, to get the final pixel coordinates (u, v) an additional transformation is required. As shown in Figure 2.4, this transformation also depends on the size and the shape of the pixels, which is not always square, and on the position of the sensor with respect to the lens. Keeping this in consideration, the projection function of a digital camera can be expressed as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \underbrace{\begin{bmatrix} f/s_x & (tan\alpha)\frac{f}{s_y} & p_x \\ 0 & f/s_y & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{P}_N} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{2.4}$$

where $\mathbf{P}_N$ is the projection matrix of the normalized camera, $\mathbf{K}$ is the calibration matrix of the real camera, (s_x, s_y) is the dimension of the pixel and $\vec{p} = (p_x, p_y)^\top$ is the principal point in pixel coordinates. For the rest of this thesis, a simplified notation of the calibration matrix will be used:

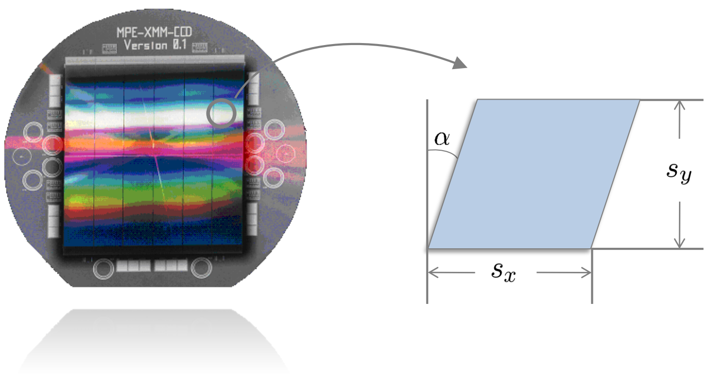


Figure 2.4: The pixels of a CCD sensor are not square.

$$\mathbf{K} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.5}$$

In most cases, pixels are almost rectangular, so the skew parameter s can be assumed to be 0. It is also a very common assumption to consider the principal point centered on the image. In this scenario, a minimal description of a camera can be parameterized only with f_x and f_y.

The entries of the calibration matrix are called intrinsic parameters because they only depend on the internal properties of the camera. In a video sequence, they remain fixed in all frames if the focus does not change and zoom does not apply.

There are other effects like radial and tangential distortions that cannot be modeled linearly. Tangential distortion is caused by imperfect centering of the lens and usually assumed to be zero. Radial distortions are produced by using a lens instead of a pin-hole. It has a noticeable effect in cameras with shorter focal lengths affecting to the straightness of lines in the image making them curve, and they can be eliminated warping the image. Let (u, v) be the pixel coordinates of a point in the input image and $(\hat{u}, \hat{v})$ its corrected coordinates. The warped image can be obtained using Brown's distortion model (Brown, 1966):

$$\begin{aligned} \hat{u} &= p_x + (u - px)\left(1 + k_1 r + k_2 r^2 + k_3 r^3 + \dots\right) \\ \hat{v} &= p_y + (v - py)\left(1 + k_1 r + k_2 r^2 + k_3 r^3 + \dots\right) \end{aligned} \tag{2.6}$$

where $r^2 = (u - p_x)^2 + (v - p_y)^2$ and $\{k_1, k_2, k_3, \ldots\}$ are the coefficients of the Taylor expansion of an arbitrary radial displacement function $L(r)$.

The quality of the sensor can also distort the final image. Cheap cameras sometimes use the rolling shutter acquisition technique that implies that not all parts of the image are captured at the same time. This can introduce some artifacts induced by the camera movement that are hardly avoided.

2.2 Camera Motion

As stated before, Equation 2.4 defines the projection of a point expressed in a coordinate frame aligned with the projection center of the camera. However, a 3D scene is usually expressed in terms of a different world coordinate frame. These coordinate systems are related via a Euclidean transformation expressed as a rotation and a translation. When the camera moves, the coordinates of its projection center relative to the camera center changes, and this motion can be expressed as a set of concatenated transformations.

Since these parameters depend on the camera position and not on its internal behavior, they are called extrinsic parameters. Adding this transformation to the projection model described in Equation 2.4, the projection of a point (X, Y, Z) expressed in an arbitrary world coordinate frame is given by

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \vec{t} \\ \vec{0}_3^\top & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.7}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\vec{t}$ is a translation 3-vector that relates the world coordinate frame to the camera coordinate frame as shown in Figure 2.3. In a video sequence with a non-static camera, extrinsic parameters vary in each frame.

2.3 Camera Calibration

Camera calibration is the process of recovering the intrinsic parameters of a camera. Basically, there are two ways to calibrate a camera from image sequences:

pattern based calibration and self (or auto) calibration. A complete survey of camera calibration can be found in (Hemayed, 2003).

Pattern based calibration methods use objects with known geometry to carry out the process. Some examples of these calibration objects are shown in Figure 2.5. Matching the images with the geometric information available from the patterns, it is possible to find the internal parameters of the camera very fast and with a very high accuracy level. Examples of this type of calibration are (Tsai, 1987) and (Zhang, 2000). However, these methods cannot be applied always, especially when using pre-recorded video sequences where it becomes impossible to add the marker to the scene.

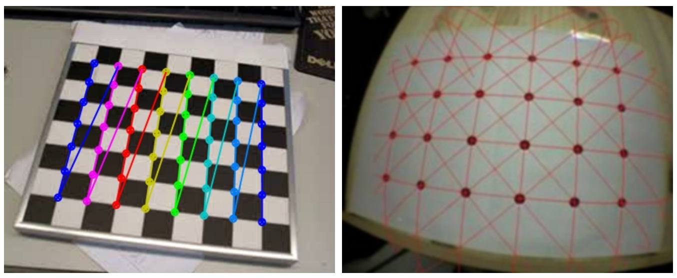

(a) Chessboard pattern. (b) Dot pattern used in ArToolkit.

Figure 2.5: Two examples of camera calibration patterns.

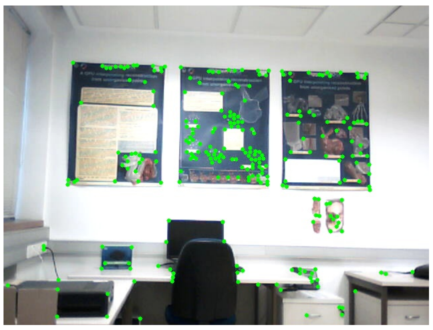
In contrast, self calibration methods obtain the parameters from captured images, without having any calibration object on the scene. These methods are useful when the camera is not accessible or when the environment cannot be altered with external markers. They obtain these parameters by exploiting some constraints that exist in multiview geometry (Faugeras et al., 1992) (Hartley, 1997) or imposing restrictions on the motion of the camera (Hartley, 1994b) (Zhong and Hung, 2006). Other approaches to camera calibration are able to perform the reconstruction from a single view (Wang et al., 2005) if the length ratios of line segments are known.

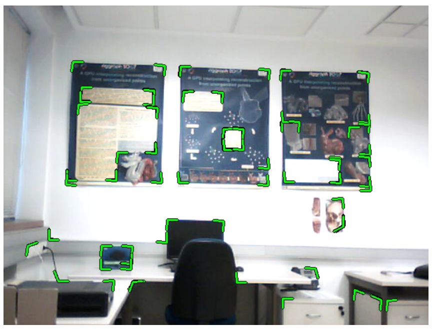
Self calibration methods usually involves two steps: affine reconstruction and metric upgrade. Appendix A develops an approximation to affine reconstruction oriented to self calibration methods. The only assumption done is that parallel lines exist in the images used to perform the calibration.

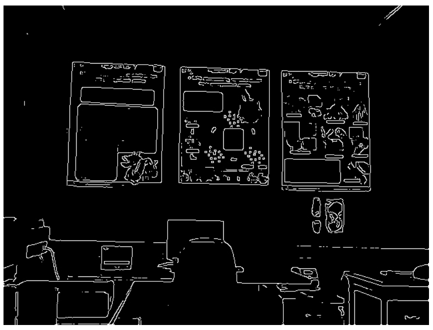
2.4 Motion Recovery

The recovery of the motion of a camera is reduced to obtain its extrinsic parameters for each frame. Although they can be obtained using motion sensors, this section is only concerned on the methods that obtain these parameters using only the images captured.

Each image of a video sequence has a lot of information contained in it. However, its raw representation as a matrix of color pixels needs to be processed in order to extract higher level information that can be used to calculate the extrinsic parameters. There are several alternatives that are used in the literature, such as corners, lines, curves, blobs, junctions or more complex representations (Figure 2.6). All these image features are related to the projections of the objects that are captured by the camera, and they can be used to recover both camera motion and 3D structure of the scene.


(a) Feature points.


(b) Junctions.


(c) Edges.

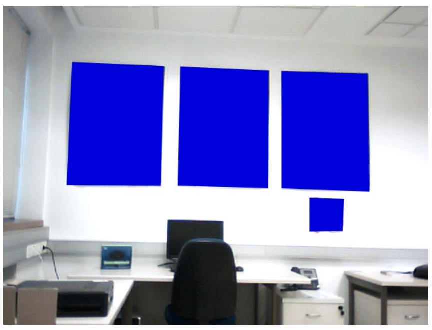

(d) Blobs corresponding to the posters.

Figure 2.6: Different features extracted from an image.

In this way, Structure from Motion (SFM) algorithms correspond to a family of methods that obtain the 3D structure of a scene and the trajectory of the camera that captured it, using the motion information available from the images.

SFM methods began to be developed in the eighties when Longuet-Higgins published the eight point algorithm (Longuet-Higgins, 1981). His method is based on the multiple view geometry framework introduced in Appendix B. Using a minimum of two images, by means of a linear relationship existing between the features extracted from them, the extrinsic parameters can recovered at the same time the structure of the scene is obtained. However the equation systems involved in the process are very sensitive to noise, and in general the solution obtained cannot be used for augmented reality applications. Trying to avoid this problem, SFM algorithms have evolved in two main directions, i.e., batch optimizations and recursive estimations. The first option is very accurate but not suitable for real time operation. The second option is fast but sometimes not as accurate as one would wish.

Optimization based SFM techniques estimate the parameters of the camera and 3D structure minimizing some cost function that relates them with the measurements taken from images (points, lines, etc.). Having an initial guess about the scene structure and the motion of the camera, 3D points can be projected using Equation 2.7 and set a cost function as the difference between these projections and measurements taken from the images. Then, using minimization techniques, the optimal scene reconstruction and camera motion can be recovered, in terms of the cost function.

On the other hand, probabilistic methods can be used as well to recover the scene structure and the motion parameters using Bayesian estimators. These methods have been widely used in the robot literature in order to solve the Simultaneous Localization and Mapping (SLAM) problem. External measures, like video images or laser triangulation sensors, are used to build a virtual representation of the world where the robot navigates. This reconstruction is recursively used to locate the robot while new landmarks are acquired at the same time. When the system works with a single camera, it is known as monocular SLAM. In fact, SLAM can be viewed as a class of SFM problem and vice versa, so existing methods for robot SLAM can be used to solve the SFM problem.

2.4.1 Batch Optimization

One of the most used approaches to solve the SFM problem has been to model it as a minimization problem. In these solutions, the objective is to find a parameter vector $\vec{x}$ that having an observation $\vec{y}$ minimizes the residual vector $\|\vec{e}\|$ in:

$$\vec{y} = f(\vec{x}) + \|\vec{e}\| \,. \tag{2.8}$$

In the simplest case, the parameter vector $\vec{x}$ contains the motion parameters and the 3D structure, the observation vector $\vec{y}$ is composed by image measurements corresponding to the projections of the 3D points and $f()$ is the function that models the image formation process. Following this formulation and assuming the pin-hole model, the objective function can be expressed as:

$$\underset{i}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{j=1}^{n} \left\| \Pi\left(\mathbf{R}_i \vec{M}_j + \vec{t}_i\right) - \vec{m}_j^i \right\|, \tag{2.9}$$

where $\mathbf{R}_i$ and $\vec{t}_i$ are the extrinsic parameters of the frame i, $\vec{M}_j$ is the point j of the 3D structure, $\vec{m}_j^i$ is its corresponding measurement in the image i and Π is the projection function given in Equation 2.3.

The solution obtained with this minimization is jointly optimal with respect to the 3D structure and the motion parameters. This technique is known as Bundle Adjustment (BA) (Triggs et al., 2000) where the name “bundle” refers to the rays going from the camera centers to the 3D points. It has been widely used in the field of photogrammetry (Brown, 1976) and the computer vision community has successfully adopted it in the last years.

Since the projection function Π in Equation 2.9 is nonlinear, the problem is solved using nonlinear minimization methods where the more common approach is the Levenberg-Marquardt (LM) (Levenberg, 1944) (Marquardt, 1963) algorithm. It is an iterative procedure that, starting from an initial guess for the parameter vector, converges into the local minima combining the Gauss-Newton method with the gradient descent approach. However, some authors (Lourakis and Argyros, 2005) argue that LM is not the best choice for solving the BA problem.

Usually, the initial parameter vector is obtained with multiple view geometry methods. As stated, the solution is very sensitive to noise, but it is reasonably close to the solution, and therefore it is a good starting point for the minimization.

The problem of this approach is that the parameter space is huge. With a normal parameterization, a total of 6 parameters per view and 3 per point are used. However, it presents a very sparse structure that can be exploited in order to get reasonable speeds. Some public implementations like (Lourakis and Argyros, 2009) take advantage of this sparseness allowing fast adjustments, but hardly in real time.

Earlier works like (Hartley, 1994a), compute a stratified Euclidean reconstruction using an uncalibrated sequence of images. BA is applied to the initial projective reconstruction which is then upgraded to a quasi-affine representation. From this reconstruction the calibration matrix is computed and the Euclidean representation is found performing a second simpler optimization step. However, the whole video sequence is used for the optimization, forcing therefore to be executed in post processing and making it unusable in real time applications.

One recent use of BA for real time SLAM can be found in (Klein and Murray, 2007). In their PTAM algorithm they use two computing threads, one for tracking and the other for mapping, running on different cores of the CPU. The mapping thread uses a local BA over the last five selected keyframes in order to adjust the reconstruction. The objective function that they use is not the classical reprojection error function given in Equation 2.9. In contrast, they use the more robust Tukey biweight objective function (Huber, 1981), also known as bisquare.

There are other works that have successfully used BA in large scale reconstruction problems. For example, (Agarwal et al., 2009) have reconstructed entire cities using BA but they need prohibitive amounts of time. For example, the reconstruction of the city Rome took 21 hours in a cluster with 496 cores using 150k images. The final result of this reconstruction is shown in Figure 2.7. It is done using unstructured images downloaded from the Internet, instead of video sequences.

Authors, like (Kahl, 2005), use the L_∞ norm in order to reformulate the problem as a convex optimization. These types of problems can be efficiently solved using second order cone programming techniques. However, as the author states, the method is very sensitive to outliers.

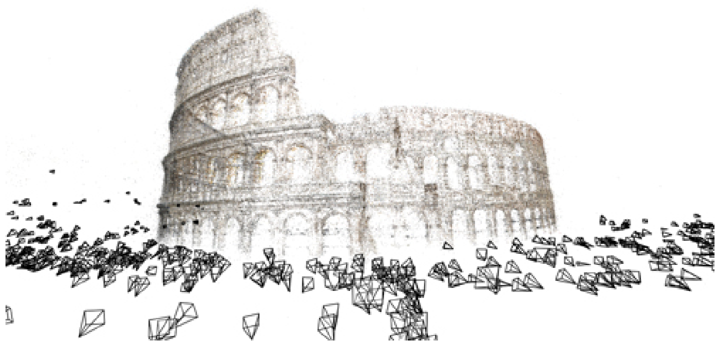


Figure 2.7: Dense reconstruction of the Colosseum (Agarwal et al., 2009).

2.4.2 Bayesian Estimation

In contrast to batch optimization methods, probabilistic techniques compute the reconstruction in an online way using recursive Bayesian estimators. In this formulation, the problem is usually represented as a state-space model. This paradigm, taken from control engineering, models the system as a state vector $\vec{x}_k$ and a measurement vector $\vec{y}_k$ caused as a result of the current state. In control theory, state-space models also include some input parameters that guide the evolution of the system. However, as they are not going to be used in this text, they are omitted in the explanations without loss of generality.

The dynamics of a state-space model is usually described by two functions, i.e., the state model and the observation model. The state model describes how the state changes over time. It is given as a function that depends on the current state of the system and a random variable, known as process noise, that represents the uncertainty of the transition model:

$$\vec{x}_{k+1} = f\left(\vec{x}_k, \vec{u}_k\right). \tag{2.10}$$

The observation model is a function that describes the measurements taken from the output of the system depending on the current state. It is given as a function that depends on the current state and a random variable, known as measurement noise, that represents the noise introduced by the sensors:

$$\vec{y}_k = h\left(\vec{x}_k, \vec{v}_k\right). \tag{2.11}$$

The challenge is to estimate the state of the system, having only access to its outputs. Suppose that at any instant k a set of observations $\mathtt{Y}_k = \{\vec{y}_1, \ldots, \vec{y}_k\}$ can be measured from the output of the system. From a Bayesian viewpoint, the problem is formulated as the conditional probability of the current state given the entire set of observations: $P\left(\vec{x}_k | \mathtt{Y}_k\right)$. The key of these algorithms is that the current state probability can be expanded using the Bayes's rule as:

$$P(\vec{x}_k|\mathtt{Y}_k) = \frac{P(\vec{y}_k|\vec{x}_k)P(\vec{x}_k|\mathtt{Y}_{k-1})}{P(\vec{y}_k|\mathtt{Y}_{k-1})}. \tag{2.12}$$

Equation 2.12 states the posterior probability of the system, i.e., the filtered state after when measurements $\vec{y}_k$ are available. Since Equation 2.10 describes a Markov chain, the prior probability of the state can also be estimated using the Chapman-Kolmogorov equation:

$$P(\vec{x}_k|\mathtt{Y}_{k-1}) = \int P(\vec{x}_k|\vec{x}_{k-1})P(\vec{x}_{k-1}|\mathtt{Y}_{k-1})d\vec{x}_{k-1}. \tag{2.13}$$

In the SLAM field, usually the state $\vec{x}_k$ is composed by the camera pose and the 3D reconstruction, and the observation $\vec{y}_k$ is composed by image measurements. As seen, under this estimation approach, the parameters of the state-space model are considered random variables that are expressed by means of their probability distributions. Now the problem is how to approximate them.

If both model noise and measurement noise are considered additive Gaussian, with covariances $\mathbf{Q}$ and $\mathbf{R}$, and the functions f and h are linear, the optimal estimate (in a least squares sense) is given by the Kalman filter (Kalman, 1960) (Welch and Bishop, 2001). Following this assumption, Equations 2.10 and 2.11 can be rewritten as:

$$\begin{aligned} \vec{x}_{k+1} &= \mathbf{A}\vec{x}_k + \vec{u}_k \\ \vec{y}_k &= \mathbf{H}\vec{x}_k + \vec{v}_k. \end{aligned} \tag{2.14}$$

The Kalman filter follows a prediction-correction scheme having two steps, i.e., state prediction (time update) and state update (measurement update), providing equations for solving the prior an the posterior means and covariances of the distributions given in Equations 2.12 and 2.13. The initial state must be provided as a Gaussian random variable with known mean and covariance. Under this assumption and because of the linearity of Equation 2.14, the Gaussian property of the state and measurements is preserved.

In the state prediction step, the prior mean and covariance of the state in time $k+1$ are predicted. It is only a projection of the state based on the previous prediction $\hat{x}_k$ and the dynamics of the system given by Equation 2.14, and it can be expressed as the conditional probability of the state in $k+1$ given the measurements up to instant k:

$$P\left(\vec{x}_{k+1}|\mathtt{Y}_k\right) \sim N\left(\hat{x}_{k+1}^{(-)}, \mathbf{P}_{k+1}^{(-)}\right), \tag{2.15}$$

where

$$\begin{aligned} \hat{x}_{k+1}^{(-)} &= \mathbf{A}\hat{x}_k \\ \mathbf{P}_{k+1}^{(-)} &= \mathbf{A}P_k\mathbf{A}^\top + \mathbf{Q}. \end{aligned} \tag{2.16}$$

When measurements are available, the posterior mean and covariance of the state can be calculated in the state update step using the feedback provided by these measurements. In this way, the state gets defined by the distribution

$$P\left(\vec{x}_{k+1}|\mathtt{Y}_{k+1}\right) \sim N\left(\hat{x}_{k+1}, \mathbf{P}_{k+1}\right), \tag{2.17}$$

where

$$\begin{aligned} \hat{x}^{k+1} &= \hat{x}_{k+1}^{(-)} + \mathbf{G}_{k+1}\left(\vec{y}_k - \mathbf{H}\hat{x}_{k+1}^{(-)}\right) \\ \mathbf{P}_{k+1} &= \left(\mathbf{I} - \mathbf{G}_{k+1}\mathbf{H}\right)\mathbf{P}_{k+1}^{(-)}. \end{aligned} \tag{2.18}$$

The matrix $\mathbf{G}$ is called the Kalman Gain, and it is chosen as the one that minimizes the covariance of the estimation $\mathbf{P}_{k+1}$:

$$\mathbf{G}_{k+1} = \mathbf{P}_{k+1}^{(-)}\mathbf{H}^\top\left(\mathbf{H}\mathbf{P}_{k+1}^{(-)}\mathbf{H}^\top + \mathbf{R}\right)^{-1}. \tag{2.19}$$

In the case that the state or the observation models are nonlinear, there is an estimator called the Extended Kalman Filter (EKF) that provides a similar framework. The idea is to linearize the system using the Taylor expansion of the functions f and h in Equations 2.10 and 2.11. Let

$$\begin{aligned}\tilde{x}_{k+1} &= f\left(\hat{x}_k, \vec{0}\right)\\ \tilde{y}_{k+1} &= h\left(\tilde{x}_{k+1}, \vec{0}\right)\end{aligned} \tag{2.20}$$

be the approximate *a priori* state and measurement vectors, assuming zero noise. If only the first order of the expansion is used, the state and the observation models can be rewritten as:

$$\begin{aligned}\vec{x}_{k+1} &\approx \tilde{x}_{k+1} + \mathbf{J}_{\vec{x}_k}\left(\vec{x}_k - \hat{x}_k\right) + \mathbf{J}_{\vec{u}_k}\vec{u}_k\\ \vec{y}_k &\approx \tilde{y}_k + \mathbf{J}_{\vec{y}_k}\left(\vec{x}_k - \tilde{x}_k\right) + \mathbf{J}_{\vec{v}_k}\vec{v}_k,\end{aligned} \tag{2.21}$$

where $\hat{x}_k$ is the *a posteriori* state obtained from the previous estimate, $\mathbf{J}_{\vec{x}_k}$ is the Jacobian matrix of partial derivatives of f with respect to $\vec{x}$ at instant k, $\mathbf{J}_{\vec{u}_k}$ is the Jacobian of f with respect to $\vec{u}$ at instant k, $\mathbf{J}_{\vec{y}_k}$ is the Jacobian of h with respect to $\vec{x}$ at instant k and $\mathbf{J}_{\vec{v}_k}$ is the Jacobian of h with respect to $\vec{v}$ at instant k. Having the system linearized, the distributions can be approximated using equations 2.15 and 2.18.

One of the first successful approaches in real time monocular SLAM using the EKF was the work of (Davison, 2003). Their state vector $\vec{x}_k$ is composed by the current camera pose, its velocity and by the 3D features in the map. For the camera parameterization they use a 3-vector for the position and a quaternion for the orientation. The camera velocity is expressed as a linear velocity 3-vector and an angular velocity 3-vector (exponential map). Points are represented by its world position using the full Cartesian representation. However, when they are first seen, they are added to the map as a line pointing from the center of the camera. In the following frames, these lines are uniformly sampled in order to get a distribution about the 3D point depth. When the ratio of the standard deviation of depth to depth estimate drops below a threshold, the points are upgraded to their full Cartesian representation. As image measurements they use 2D image patches of 9×9 to 15×15 pixels that are matched in subsequent frames using normalized Sum of Squared Difference correlation operator (SSD). In this case, the measurement model is nonlinear, so for the state estimation they use a full covariance EKF assuming a constant linear and angular velocity transition model. Other earlier authors like (Broida et al., 1990) use an iterative version of the EKF for tracking and reconstructing an object using a single camera. The work of (Azarbayejani and Pentland, 1995) successfully introduces in the state vector the focal length of the camera, allowing the estimation even without prior knowledge of the camera calibration parameters. Moreover, they use a minimal parameterization for representing 3D points. In this case, points are represented as the depth value α that they have in the first frame. This representation reduces to the third part the parameter space needed to represent 3D points, however as demonstrated later by (Chiuso et al., 2002), it becomes invalid in long video sequences or when inserting 3D points after the first frame.

However, the flaw of the EKF is that the Gaussian property of the variables is no longer preserved. Moreover, if the functions are very locally nonlinear, the approximation using the Taylor expansion introduces a lot of bias in the estimation process. Trying to avoid this, in (Julier and Uhlmann, 1997) an alternative approach is proposed called Unscented Kalman Filter (UKF) that preserves the normal distributions throughout the nonlinear transformations. The technique consists of sampling a set of points around the mean and transform them using the nonlinear functions of the model. Using the transformed points, the mean and the covariance of the variables is recovered. These points are selected using a deterministic sampling technique called unscented transform.

This filter has been used successfully in monocular SLAM applications in works like (Chekhlov et al., 2006). They use a simple constant position model, having a smaller state vector and making the filter more responsive to erratic camera motions. The structure is represented following the same approach of Davison in (Davison, 2003).

All variants of the Kalman Filter assume that the model follows a Gaussian distribution. In order to get a more general solution, Monte Carlo methods have been found to be effective in SLAM applications. The Monte Carlo simulation paradigm was first proposed by (Meteopolis and Ulam, 1949). It consists of approaching the probability distribution of the model using a set of discrete samples generated randomly from the domain of the problem. Monte Carlo simulations are typically used in problems where it is not possible to calculate the exact solution from a deterministic algorithm. Theoretically, if the number of samples goes to infinity, the approximation to the probability distribution becomes exact.

In the case of sequential problems, like SLAM, the family of Sequential Monte Carlo methods (SMC) also known as Particle Filters (PF) (Gordon et al., 1993) are used. The sequential nature of the problem can be used to ease the sampling step using a state transition model, as in the case of Kalman Filter based methods. In this scenario, a set of weighted samples are maintained representing the posterior distribution of the camera trajectory so that the probability density function can be approximated as:

$$P(\vec{x}_k|\mathtt{Y}_k) \approx \sum_i w_k^{(i)} \delta\left(\vec{x}_k - \vec{x}_k^{(i)}\right) \tag{2.22}$$

where $\delta()$ is the Dirac Delta and $\{\vec{x}_k^{(i)}, w_k^i\}$ is the sample i and its associated weight. Each sample represents an hypothesis about the camera trajectory. Since it is not possible to draw samples directly from the target distribution $P(\vec{x}_k|\mathtt{Y}_k)$, samples are drawn from another arbitrary distribution called importance density $Q(\vec{x}_k|\mathtt{Y}_k)$ such that:

$$P(\vec{x}_k|\mathtt{Y}_k) > 0 \Rightarrow Q(\vec{x}_k|\mathtt{Y}_k) > 0. \tag{2.23}$$

In this case, the associated weights in Equation 2.22 are updated every time step using the principle of Sequential Importance Sampling (SIS) proposed by (Liu and Chen, 1998):

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{P\left(\vec{y}_k|\vec{x}_k^{(i)}\right) P\left(\vec{x}_k^{(i)}|\vec{x}_{k-1}^{(i)}\right)}{Q\left(\vec{x}_k^{(i)}|\mathtt{X}_{k-1}, \vec{y}_k\right)}. \tag{2.24}$$

This sampling technique has the problem that particle weights are inevitably carried to zero. This is because since the search space is not discrete, particles will have weights less than one, so repeated multiplications will lead them to very small values. In order to avoid this problem, particles with higher weights are replicated and lower weighted particles are removed. This technique is known as Sequential Importance Resampling (SIR).

An example is the work of (Qian and Chellappa, 2004). They use SMC methods for recovering the motion of the camera using the epipolar constraint to form the likelihood. The scene structure is recovered using the motion estimates as a set of depth values using SMC methods as well. They represent the camera pose as the three rotation angles and the translation direction using spherical coordinates. The magnitude is later estimated in a post process step through triangulation. Although results are good, it has two drawbacks: all the structure must be visible in the initial frame and the real time operation is not achieved.

Authors like (Pupilli and Calway, 2006) use a PF approach in order to obtain the probability density of the camera motion. They model the state of the system as a vector containing the pose of the camera and the depth of the mapped features. However, due to real time requirements, they only use the particle filtering for estimating the camera pose. For updating the map they use a UKF allowing real time operation.

Summarizing, a PF gets completely defined by the importance distribution $Q(\vec{x}_k|\Upsilon_k)$ and by the marginal of the observation density $P\left(\vec{y}_k|\vec{x}_k\right)$. It is very common to choose the first to be proportional to the dynamics of the system, and depending on the observation density used, the filter can be classified in two variants:

- **Bottom-up approach** or *data driven*: The marginal $P\left(\vec{y}_k|\vec{x}_k^{(i)}\right)$ depends on the features measured in the images. The projections of 3D points obtained using the camera parameters defined by the sample are compared with those features, and the probability is set as their similarity. This strategy can be reduced to four steps:
  1. **Image Analysis:** Measurements are taken from the input images.
  2. **Prediction:** The current state is predicted using the dynamics of the system and its previous configuration.
  3. **Synthesis:** The predicted model is projected, thus obtaining an image representation.
  4. **Evaluation:** The model is evaluated depending on the matches between the measurements taken and the predicted image.
- **Top-down approach** or *model driven*: The model of the system guides the measurement, comparing a previously learned description of the 3D points with the image areas defined by their projections. Therefore, this approach does not depend on any feature detection method, but the visual appearance of 3D points needs to be robustly described. This strategy can also be reduced to four steps:
  1. **Prediction:** The current state is predicted using the dynamics of the system and its previous configuration.
  2. **Synthesis:** The predicted model is projected, thus obtaining an image representation.
  3. **Image Analysis:** A description of the areas defined by the projection of the model is extracted.
  4. **Evaluation:** The model is evaluated depending on the similarities between the extracted description and the predicted image.

2.5 Discussion

It has be seen in this chapter that there are a lot of alternatives to solve the motion and the structure parameters of a scene using a video sequence. At the moment, cannot be said that one of them is the best solution for every estimation problem. The concrete application together with its accuracy and performance requirements will determine the appropriate method to be used.

Having that the target application of this thesis is an augmented reality system, some accuracy aspects can be sacrificed for the benefit of speed. After all, the human eye tolerates pretty well certain levels of error, but needs a high frame rate in order not to loose the feeling of smooth animation.

Part II

Proposal

Chapter 3

Proposed Method

Existing SLAM methods are inherently iterative. Since the target platform of this thesis is a parallel system, iterative methods are not the best candidates for exploiting its capabilities, because they usually have dependencies between the iterations that cannot be solved efficiently. In this context, instead of adapting a standard solution like BA or Kalman Filter, this chapter develops a fully parallelizable algorithm for both camera tracking and 3D structure reconstruction based on Monte Carlo simulations. More specifically, a SMC method is adapted taking into account the restrictions imposed by the parallel programming model.

The basic operation of a SMC method is to sample and evaluate possible solutions taken from a target distribution. Taking into account that each one of those samples can be evaluated independently from others, the solution is very adequate for parallel systems and has very good scalability. In contrast, it is very computationally intensive being unsuitable for current serial CPUs, even multicores.

Following these argumentations, this chapter describes a full SLAM method using SMC, using only a video sequence as input. Some experiments are provided in order to demonstrate its validity.

A synthesis of this chapter and Chapter 4 has been presented in:

> *Sánchez, J. R., Álvarez, H., and Borro, D. "Towards Real Time 3D Tracking and Reconstruction on a GPU Using Monte Carlo Simulations". In International Symposium on Mixed and Augmented Reality (ISMAR'10), pp. 185–192. Seoul, Korea. 2010.*

Sánchez, J. R., Álvarez, H., and Borro, D. "GFT: GPU Fast Triangulation of 3D Points". In Computer Vision and Graphics (ICCVG'10), volume 6375 of Lecture Notes in Computer Science, pp. 235–242. Warsaw, Poland. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. "GPU Optimizer : A 3D reconstruction on the GPU using Monte Carlo simulations". In Poster proceedings of the 5th International Conference on Computer Vision Theory and Applications (VISAPP'10), pp. 443–446. Angers, France. 2010.

3.1 Overview

The pipeline of the SLAM method used in this work (shown in Figure 3.1) consists of three different main modules:

- A **Feature Tracking** module, that analyzes the input images looking for salient areas that give useful information about the motion of the camera and the structure of the underlying scene.
- A **3D Motion Tracking** module, that using the measurements taken in the images estimates the extrinsic parameters of the camera for each frame.
- A **Scene Reconstruction** module, that extends the structure information about the scene that the system has, adding new 3D points and adjusting already mapped ones. This is done taking into account the motion model predicted by the motion tracker and the measurements given by the feature tracker.

The scheme proposed here is a bottom-up approach, but the pose and structure estimation algorithms are suitable for top-down approaches as well. Under erratic motions, top-down systems perform better since they do not need to detect interest points into the images. However, they rely on the accuracy of the predictive model making them unstable if it is not well approximated.

The method proposed here applies to a sequence of sorted images. The camera that captured the sequence must be weakly calibrated, i.e., only the two focal lengths are needed and it is assumed that there is no lens distortion. The calibration matrix used has the form of Equation 2.5. The principal point is assumed to be centered and the skew is considered null. This assumption can be done if the used

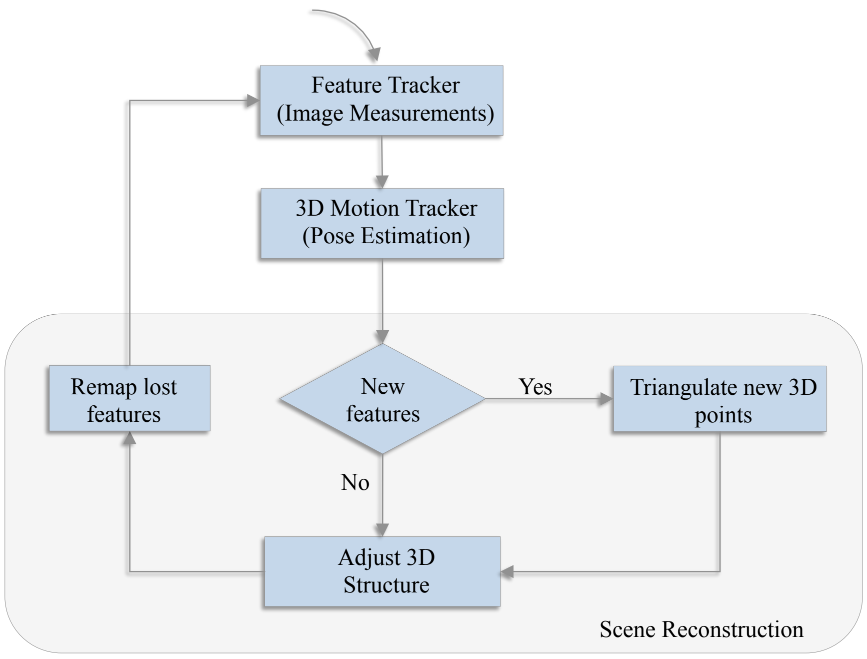


Figure 3.1: SLAM Pipeline.

camera has not a lens with a very short focal length, which is the case in almost any non-professional camera.

From now on, a frame k is represented as a set of matched features $\mathtt{Y}_k = \{\vec{y}_1^k, \ldots, \vec{y}_n^k\}$ so that a feature $\vec{y}_j$ corresponds to a single point in each image of the sequence, i.e., a 2D feature point. The set of frames from instant i up to instant k is represented as $\mathtt{Y}_{i:k}$. The 3D structure is represented as a set of 3D points $\mathtt{Z} = \{\vec{z}_1, \ldots, \vec{z}_n\}$ such that each point has associated a feature point which represents its measured projections into the images. The 3D motion tracker estimates the pose using these 3D-2D point matches. Drawing on the previous pose, random samples for the current camera configuration are generated and weighted using the Euclidean distance between measured features and the estimated projections of the 3D points. The pose in frame k is represented as a rotation quaternion $\vec{q}_k$ and a translation vector $\vec{t}_k$ with an associated weight w_k. The pin-hole camera model is assumed, so the relation between a 2D feature in the frame k and its associated 3D point is given by Equation 2.7.

In the structure estimation part, the 3D map is initialized using epipolar geometry (see Appendix B). When a great displacement is detected in the image, the camera pose relative to the first frame is estimated using the 8 point algorithm (Longuet-Higgins, 1981). Then, the 3D structure is built using linear triangulation. This method produces a very noisy structure map, but it is quickly improved by the structure optimizer.

Every time a new frame becomes available, the 3D structure is adjusted taking into account the measurements extracted. For this task a SMC model is used as well. In this case, each point in the map is sampled around its neighborhood and projected using the pose of the previously tracked frames. These projections are compared with tracked 2D feature points so that the sample that minimizes the sum of Euclidean distances is chosen as the new location of the 3D point.

Summarizing, the objective is to find the pose for every frame k and the reconstruction of every feature point i that satisfy:

$$\vec{y}_i^k = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_u(\vec{z}_i, k) \\ f_v(\vec{z}_i, k) \end{bmatrix} = \Pi\big(\mathbf{R}_k \vec{z}_i + \vec{t}_k\big) \tag{3.1}$$

Following sections analyze in detail these steps. A study about the feature tracking is done in Section 3.2 detailing some image measurement techniques. The initialization of the structure map, including the triangulation of new points is described in Section 3.3. Section 3.4 deals with the 3D pose estimation. Section 3.5 covers the structure adjustment step, and finally Section 3.6 explains how lost points can be remapped. Some experiments demonstrating the suitability of the method are also described in Section 3.7.

3.2 Image Measurements

As stated in Section 2.4 the images need to be processed in order to extract useful information from them. The main challenge is to identify a set of characteristics directly related to the 3D structure and the motion of the scene, allowing to recover those parameters from them.

Among the alternatives, in this thesis a feature point representation has been chosen because of its simplicity. Figure 3.2 shows an overview of the image measurement module implemented, composed by a detector, a descriptor and a tracker. Following sections describe these parts in detail.

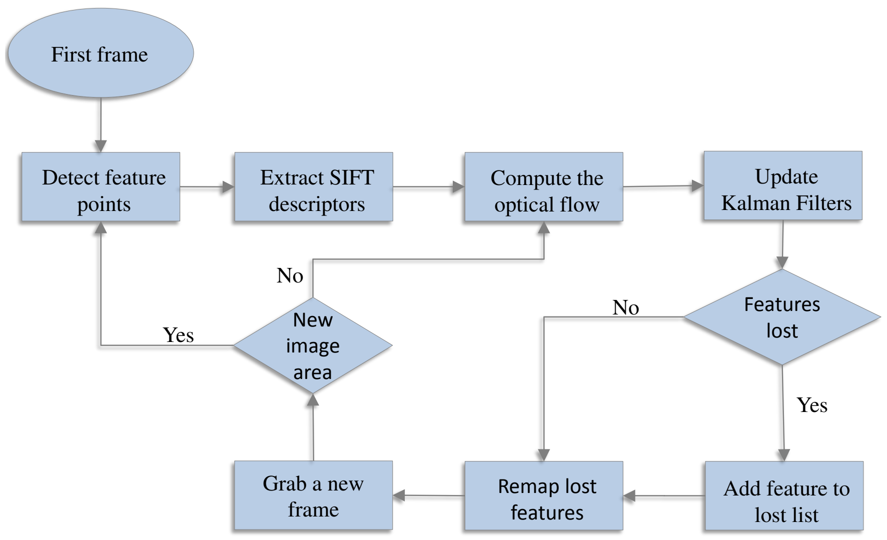


Figure 3.2: Description of the feature tracker.

3.2.1 Feature Detection

A feature point can be defined as a salient point which can be easily identified in the image. It can be represented as a pixel inside the image or as an estimated position with its respective covariance. Some desirable properties of a feature point are:

- It should be well defined, preferably with a set of mathematical restrictions.
- It should be locally differentiable, in the sense that there is no any feature point in its vicinity that has a similar definition.
- It should be invariant to as many transformations as possible, including camera rotations, translations, scales and illumination changes.

There are many feature point detectors in the literature. The most common are those corresponding to the family of corner detectors, that define a corner as a point having high intensity variations when moving around its vicinity. This is normally computed as the response of the pixel to the SSD operator between a patch around the pixel and its vicinity. High values indicate that the point has a low self-similarity, thus corresponding to a corner. The simplest implementation is the Moravec corner detector (Moravec, 1977) that computes the self similarity shifting a patch in eight directions (horizontals, verticals and diagonals). The point is considered a corner if all shiftings produce a correlation value above a predefined threshold. The similarity operator given a window W centered at the pixel (x, y) with a displacement $(\Delta x, \Delta y)$ is defined as:

$$c(x, y) = \sum_{(u,v)\in W} [I(u, v) - I(u + \Delta x, v + \Delta y)]^2 \tag{3.2}$$

where $I(x, y)$ is the intensity of the pixel with coordinates x, y. However this operator only finds corners in the direction of the shifted patches, not being invariant to rotations. Harris and Stephens (Harris and Stephens, 1988) improved this operator by calculating the dominant intensity variations instead of using shifted patches. This is done approximating the shifted image using the first order Taylor expansion:

$$I(u + \Delta x, v + \Delta y) \approx I(u, v) + \frac{\partial I(u, v)}{\partial x}\Delta x + \frac{\partial I(u, v)}{\partial y}\Delta y. \tag{3.3}$$

Substituting in Equation 3.2:

$$c(x, y) = \sum_{(u,v)\in W} \left[\frac{\partial I(u, v)}{\partial x}\Delta x + \frac{\partial I(u, v)}{\partial y}\Delta y\right]^2 \tag{3.4}$$

which can be rewritten in a matrix form:

$$\begin{aligned} c(x, y) &= [\Delta x \; \Delta y] \begin{bmatrix} \sum_{(u,v)\in W} \left(\frac{\partial I(u, v)}{\partial x}\right)^2 & \sum_{(u,v)\in W} \left(\frac{\partial I(u, v)}{\partial x}\frac{\partial I(u, v)}{\partial y}\right) \\ \sum_{(u,v)\in W} \left(\frac{\partial I(u, v)}{\partial x}\frac{\partial I(u, v)}{\partial y}\right) & \sum_{(u,v)\in W} \left(\frac{\partial I(u, v)}{\partial y}\right)^2 \end{bmatrix} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix} \\ &= [\Delta x \; \Delta y]\mathbf{M} \begin{bmatrix}\Delta x \\ \Delta y\end{bmatrix}. \end{aligned} \tag{3.5}$$

The matrix $\mathbf{M}$ represents the intensity variation in the vicinity of the interest point. Thus, its eigenvectors denote the directions of maximum intensity variations. Figure 3.3 shows some possible behaviors of the Equation 3.5. The presence of large eigenvalues means that the point corresponds to a corner, in contrast to smooth regions that are represented by small eigenvalues. Due to performance reasons, Harris detector does not compute the eigenvalues of all pixels. Instead the strength of a corner is assigned as:

$$M_c(x,y) = \lambda_1\lambda_2 - \kappa\,(\lambda_1+\lambda_2)^2 = \det(\mathbf{M}) - \kappa\ \mathrm{trace}^2(\mathbf{M}) \tag{3.6}$$

where λ_1 and λ_2 are the eigenvalues of $\mathbf{M}$ and κ is a sensitivity parameter.

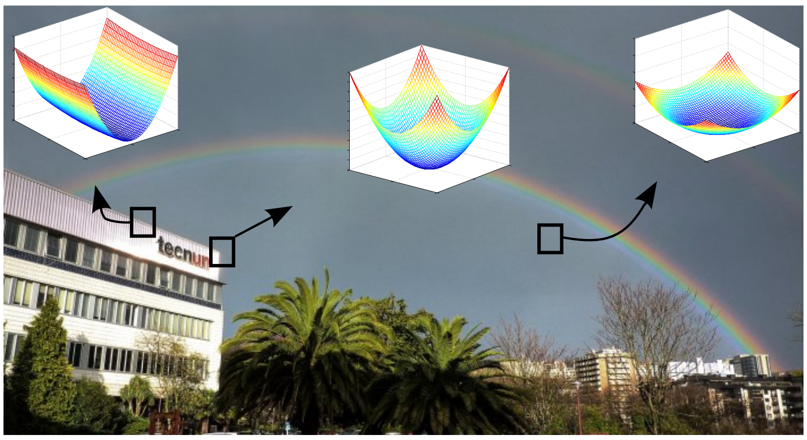


Figure 3.3: Different responses of the Harris detector [1].

In this work, the corner strength is measured as the minimum eigenvalue of the matrix $\mathbf{M}$, which corresponds to the Shi and Tomasi detector (Shi and Tomasi, 1994). Moreover a technique known as non-maxima suppression is applied, so that only the strongest point in a 3×3 local neighborhood remains. In this way, it is not possible to have two consecutive pixels marked as corners. The feature detection is carried out every frame, but only in the image regions that have no features on it, i.e., in new areas. Figure 3.6a shows the obtained corners applying this method.

3.2.2 Feature Description

The representation of a feature point can be extended adding a description of the image region around its position. It is usually done measuring the local image

[1]Photography courtesy of Carlos Buchart.

gradients using a patch centered at the feature point. This information is stored in a vector called feature descriptor that should have the next properties:

- **Highly distinctive:** The descriptor of different feature points should be different.
- **Easy to compare:** A metric should be defined in order to measure the difference between two feature descriptors. Usually the Euclidean distance is sufficient.
- **Invariant:** The descriptor of a feature point should be preserved even after image transformations like rotations, translations or scales. Invariance against illumination changes is also desirable.

There are several descriptors in the literature such as SIFT (Lowe, 1999) (Lowe, 2004), SURF (Bay et al., 2008) or FERNS (Ozuysal et al., 2010), but in this thesis, a simplified version of SIFT descriptors are used.

The SIFT method was originally designed for object recognition tasks, but it can be very useful for tracking where a feature must be located and matched along different images. In addition to the feature descriptor, it also includes a feature detector. In object recognition tasks, a SIFT based algorithm can locate an object in a cluttered image, having only a database with the SIFT features of the object sought.

In the original implementation of the SIFT algorithm, feature points are located using the Difference of Gaussians (DOG) technique. It is done subtracting images produced by convolving the original image with different Gaussian filters at different scales. It can be seen as a band-pass filter. Each pixel in a scale level is compared with its eight neighbors. Pixels that are locally maxima or minima are then compared with its 18 neighbors in the next and previous scales levels. Those pixels that are locally maxima or minima are considered feature points. The scale where the feature was detected is stored as the scale of the feature and will be used later in order to calculate the descriptor of the feature. This process can be viewed more clearly in Figure 3.4.

In order to achieve invariance against rotations, the dominant direction and magnitude of the intensity gradient is calculated for each feature point. The gradients of all the pixels around the feature are calculated, and their magnitudes are weighted using a Gaussian whose variance depends on the scale of the feature. The gradients of the pixels are discretized in 36 directions (10 degrees

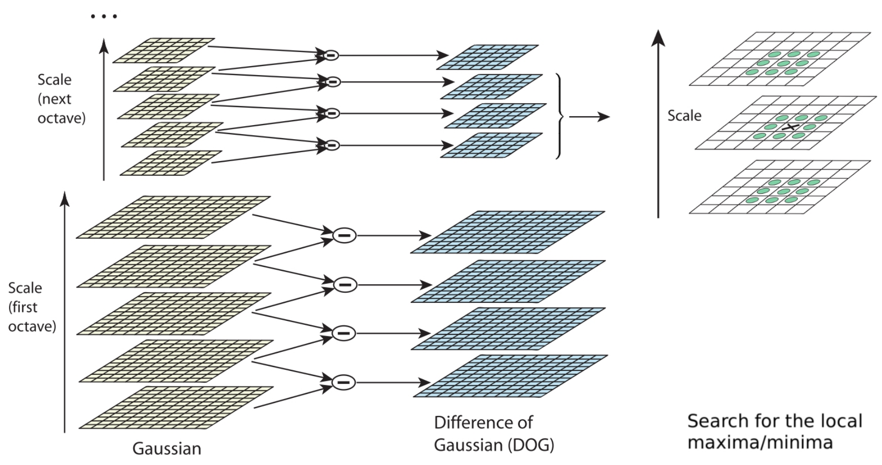

Scale (next octave)
Scale (first octave)
Gaussian
Difference of Gaussian (DOG)
Scale
Search for the local maxima/minima

Figure 3.4: Feature point detection using DOG (Lowe, 2004).

per direction) building a histogram with their weighted magnitudes. The dominant direction of the feature is set as the direction corresponding to the highest value in the histogram. If there are two dominant directions, both magnitudes are stored replicating the feature point.

Using the dominant direction, a descriptor containing the gradient information of the neighborhood of the feature is calculated. In the original method a 16×16 window around the feature is used in the scale image where it was detected. This window is subdivided in four 4×4 regions so that for each region a histogram of eight gradient orientations is calculated. In order to achieve the rotational invariance, the dominant direction is subtracted to all the computed gradients. The descriptor is built as the concatenation of the four histograms giving a total of 128 elements per feature point. Finally, it is normalized to unit length in order to minimize the effects of the illumination.

SIFT feature detection and descriptor computation require a lot of processing time and they are not suitable for real time applications. For that reason, some simplifications have been adopted in order to get real time performance based on the indications of (Wagner et al., 2010). First, the original DOG has been replaced by the faster Shi and Tomasi detector. This implies that the invariance against scale changes is lost, so the dominant direction of the features must be calculated using a fixed Gaussian in order to weight the gradients of neighbor pixels. Moreover, only one descriptor for each feature is retained, independently of not detecting a strong gradient orientation. Finally, in order to accelerate the SIFT descriptor computation, all possible kernel orientations are precomputed in order to accelerate the online processing.

The main drawback of these restrictions is that the matching quality will decrease if the system suffers a strong movement combined with a notable scale change. However, a SLAM application can afford these restrictions since, in contrast to object detection applications, it can take the advantage of the spatial coherence given by the sequence of images. However, if the tracking is completely lost, the spatial coherence cannot be used. In this case, the scale change is not a problem since these losses are due to sudden movements that do not involve large scale changes.

Figure 3.5 shows the time needed to compute the descriptors in a test sequence using an Intel Core2Duo@3.0GHz. It runs quite fast, because the implementation uses multiple threads and scales well in a multiple-core CPU, needing less that 10 ms to compute about 1000 descriptors.

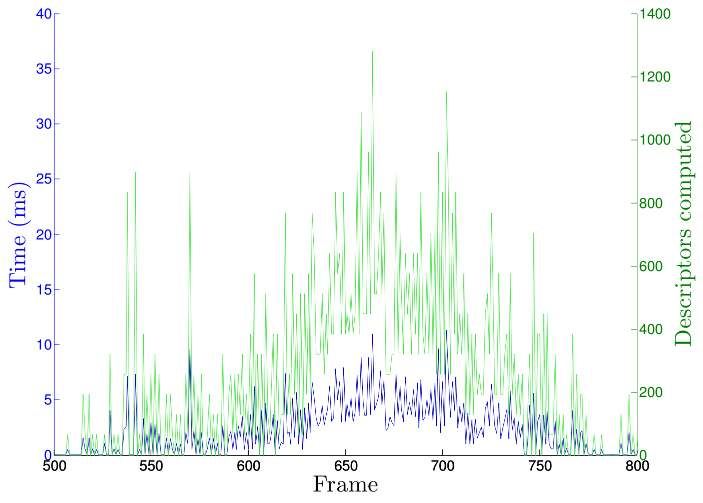


Figure 3.5: SIFT computation time.

3.2.3 Feature Tracking

Given the position of a feature point in a reference image, feature tracking refers to the process of determining its position in the rest of the images of the video sequence.

Although feature points in different images can be matched using their SIFT descriptors, in this work they are not used for this purpose. Instead, the optical flow of the features is computed using the Lukas-Kanade algorithm in two pyramidal reductions (Bouguet, 2000). The optical flow is defined as the apparent motion of the features along the video sequence and it is computed as the least squares solution of:

$$\epsilon(\Delta x, \Delta y) = \sum_{u,v \in w} [I(u,v) - J(u + \Delta x, v + \Delta y)]^2, \tag{3.7}$$

where $I(x, y)$ and $J(x, y)$ are consecutive images in the video sequence and $(\Delta x, \Delta y)$ is the optical flow of the image in the window w. The algorithm first runs the optical flow computation in the lowest resolution image of the pyramid. Then, the results are propagated to the upper levels as an initial guess of the optical flow. Finally, in the highest level (the original image) the refined optical flow is obtained. As can be seen, the optical flow computation is very similar to the similarity operator given in Equation 3.2. Indeed, the optical flow is the displacement that minimizes the SSD of a patch in the two images. Figure 3.6b shows the optical flow vectors associated to a set of features.

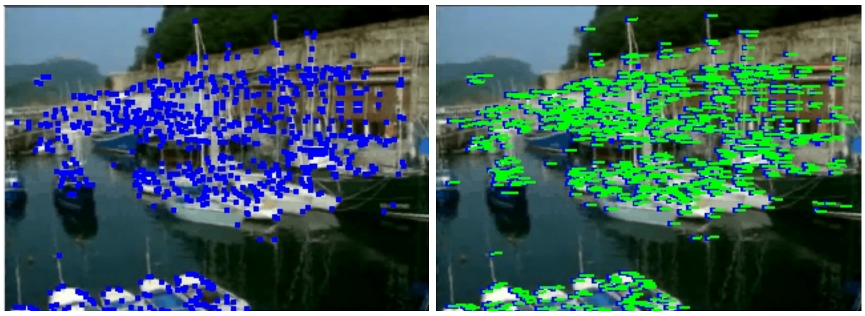

(a) Feature points detected by the Shi-Tomasi method.
(b) Optical flow of the sequence.

Figure 3.6: Measurements taking on the images

If the optical flow algorithm fails tracking a feature point, it is marked as lost. In this case, the stored SIFT descriptor is compared with the descriptors of the new features in order to relocalize the lost feature. The best match is selected as the one that minimizes the Euclidean distance between those descriptors. Moreover, the match is only accepted if the difference between the matching score of the best candidate with respect to the second, is greater than a threshold. As will be detailed in Section 3.6, in each frame only the feature points having associated a 3D point whose projection using the actual pose is inside the image are subject to be relocalized. Moreover, only points lying inside a patch around the projection are considered.

In order to make the tracking robust to outliers, the coherency of the optical flow is checked using a separate Kalman Filter for each feature point. The filter provides an estimation about the position of the features assuming a constant velocity motion model. If this prediction differs from the measurements given by the optical flow, the point is considered an outlier.

Taking into account the constant velocity model, the state vector for a feature point is represented as its position and velocity. Let $\vec{y}^k = \left(u^k, v^k\right)^\top$ be a point in the frame k moving with a velocity of $(\Delta x, \Delta y)$. The subscripts indicating the point index have been ommited for readability. Following Equation 2.14, the transition function of the filter is given by:

$$\begin{bmatrix} \hat{u}^{k+1} \\ \hat{v}^{k+1} \\ \Delta x \\ \Delta y \end{bmatrix} = \mathbf{A} \begin{bmatrix} u^k \\ v^k \\ \Delta x \\ \Delta y \end{bmatrix} + \vec{\omega} \tag{3.8}$$

where $\vec{\omega}$ follows a 4-dimensional Gaussian distribution with covariance $\mathbf{Q}$. The transition matrix is defined as:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Consecuently, the observation function can be defined as:

$$\begin{bmatrix} u^{k+1} \\ v^{k+1} \end{bmatrix} = \mathbf{H} \begin{bmatrix} \hat{u}^{k+1} \\ \hat{v}^{k+1} \\ \Delta x \\ \Delta y \end{bmatrix} + \vec{\mu} \tag{3.9}$$

where $\vec{\mu}$ follows a 2-dimensional Gaussian distribution with covariance $\mathbf{R}$ and the measurement matrix is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Considering the state vector as a Gaussian variable and taking into account that equations 3.8 and 3.9 are linear, the expected position of the feature and its covariance can be estimated using the equations 2.16 and 2.18.

The covariance $\mathbf{Q}$ in Equation 3.8 is modeling the uncertainty of the constant velocity model. In this way, it is set small but different to zero in order to allow small velocity variations. The covariance $\mathbf{R}$ in Equation 3.9, following the results of (Kanazawa and Kanatani, 2001), is set as a diagonal matrix with 2 pixels of uncertainty.

However, as shown in Figure 3.7, some outliers can appear in form of false corners. These situations are not detected by Kalman Filters because the motion pattern is coherent with a constant velocity model. In fact, they can only be detected using higher level geometric constraints, like the epipolar geometry for example.

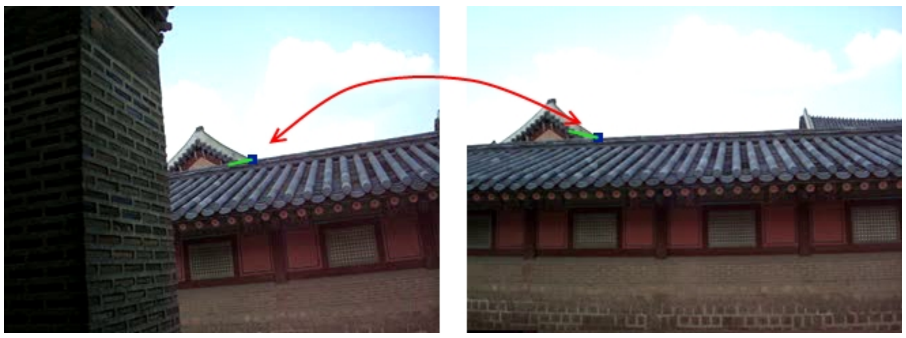
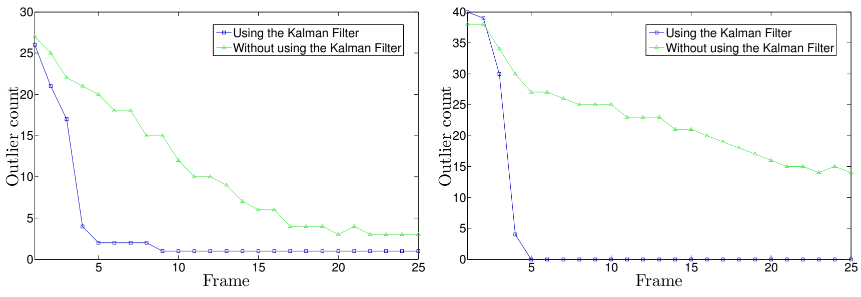


Figure 3.7: A false corner indectectable by the Kalman Filter

Anyway, those situations are not considered here as long as they are detected in the 3D reconstruction stage in Section 3.3.

In contrast, as shown in Figure 3.8, the outliers induced by incorrect feature detections are rapidly detected. The plots correspond to the detection process in a synthetic sequence which has been perturbed with different levels of additive Gaussian noise.



(a) Additive noise of $\sigma = 70$ in each color channel.

(b) Additive noise of $\sigma = 90$ in each color channel.

Figure 3.8: Outliers detected by the Kalman Filter

3.3 Initial Structure

Before starting the pose estimation, an initial 3D structure is needed. There are several ways to obtain it, depending on the amount of user interaction required.

Klein and Murray (Klein and Murray, 2007) use the five point algorithm (Stewénius et al., 2006) that can find the Essential matrix using only five point correspondences. Their initialization is only applicable to online video sequences, since the user is required to translate the camera in one direction. The initial and the final frames of the translation are used to estimate the epipolar geometry of the sequence and the initial map is built after a BA step.

In this thesis, two initialization alternatives are used. In the first, the user is required to select four points in the image defining a square. Having these points, the initial pose is calculated using the Direct Linear Transform (DLT) method (Hartley and Zisserman, 2004). This initialization has the advantage of being very accurate, but needs the user to cooperate and it is only practical for offline video sequences.

The second alternative is less restricted and the user is not required to cooperate. When a great pixel displacement is detected, the Essential matrix is calculated using the 8 point algorithm (Longuet-Higgins, 1981) together with RANSAC (Fischler and Bolles, 1981) and adjusted using LM.

Having two frames of the video sequence described by two sets of matched features $\mathtt{Y}_k$ and $\mathtt{Y}_l$, the relation between them can be described by the Fundamental matrix, or in the case of calibrated images by the Essential matrix:

$$\vec{y}_i^{l\top} \mathbf{E} \vec{y}_i^k = 0 \tag{3.10}$$

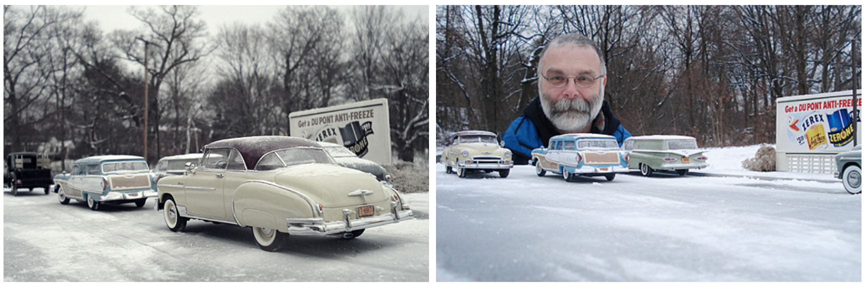
This relation can be used to write a linear homogeneous system of equations in the terms of $\mathbf{E}$. Since the Essential matrix is defined up to an arbitrary scale factor, having eight point matches is sufficient to solve it. Normally, the feature tracker has more than eight point matches available, so the problem is overdetermined. In order to minimize the effects of incorrect correspondences, this fact can be exploited using the RANSAC algorithm. It randomly selects groups of eight points and calculates the corresponding Essential matrix for each group. The residual of each matrix is calculated using Equation 3.10 with all the matches and the correct matrix is selected as the one that minimizes the number of outliers.

From this matrix and aligning the origin of the coordinate system in the first frame, the camera pose of the second frame can be recovered (see Appendix B). This first pose estimation allows to calculate a prior 3D reconstruction using linear triangulation, that later is adjusted when more views become available.

If scale information is not supplied, the obtained reconstruction is only valid up to an unknown scale factor, which means that only relative distances can be measured. This problem is present in every reconstruction algorithm that works only with images because of the scale ambiguity inherent to perspective cameras. This effect is illustrated in Figure 3.9. The image on the left seems to be a photo taken in the fifties. However, looking at the image on the right, it becomes clear that the photo is taken from a scale model.

The obtained initial reconstruction is translated so that the origin is at the centroid of the point cloud and scaled so that the Root Mean Squared (RMS) distance of the points to the origin is $\sqrt{3}$. In this way, the mean norm of every point is going to be near to one, achieving a better numerical stability in the whole system. After this transformation, the computed camera pose must be transformed as well in order to be adjusted to the new 3D point cloud.

At the rest of the sequence, new points are added to the map as long as new 2D



(a) Looks real. (b) But it is not.

Figure 3.9: Scale ambiguity shown in photographs. From Michael Paul Smith's collection. [2]

features are detected. These new points are computed using linear triangulation as well, but in this case the pose data come from the pose estimator instead of epipolar geometry. However, as stated in Section 3.2, some outliers can appear as false corners. In order to detect them, before running the triangulation, the epipolar constraint is checked using the corresponding projections. Since the pose of the camera is known, the Essential matrix can be directly computed from it.

As seen in Figure 3.10, the triangulation is strongly influenced by the angle between the rays, being the worst case moving forward or backward. In order to minimize the uncertainty of the reconstruction, a constraint is imposed on the length of the baseline of the frames used for the triangulation (Cornelis, 2004).

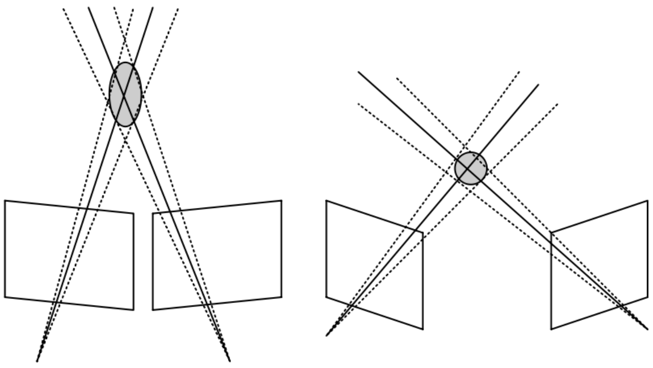


Figure 3.10: Uncertainty region triangulating points from images.

[2] `http://www.flickr.com/photos/24796741@N05/`

Let $\vec{x}_{k(i)}$ be the *i*th row of the camera matrix for the frame *k*. Let u^k and v^k be the *x,y* coordinates of the feature point $\vec{y}_i^k$. The corresponding 3D point $\vec{z}_i$ is obtained solving:

$$\begin{bmatrix} \vec{x}_{k(3)}u^k - \vec{x}_{k(1)} \\ \vec{x}_{k(3)}v^k - \vec{x}_{k(2)} \\ \vec{x}_{l(3)}u^l - \vec{x}_{l(1)} \\ \vec{x}_{l(3)}v^l - \vec{x}_{l(2)} \end{bmatrix} \vec{z}_i = \vec{0} \tag{3.11}$$

This stage is implemented in the CPU since it runs very fast, even when many points are triangulated.

3.4 Pose Estimation

The pose is estimated each frame using the 2D features given by the feature tracker described in Section 3.2. Since these features might contain noise and outliers, the direct least squares solution using the DLT method can lead to very large errors in the estimation. There are several solutions to avoid this problem, but because of the parallelism requirements of this thesis, a SMC solution has been chosen.

In order to facilitate the sampling task, the pose is parameterized using a rotation quaternion and a translation 3-vector instead of its 4×4 matrix representation. Although the minimal representation using Euler angles can be used, quaternions are safer because they do not suffer from gimbal lock. Thus, the pose in frame k is represented by a 7-vector in the form $\vec{x}_k = \left[\vec{q}_k, \vec{t}_k\right]^T$, so that the set of camera poses up to k is given by $\mathtt{X}_{1:k}$.

The goal is to get for each frame k the state vector $\vec{x}_k$ that best adjusts to the image observations $\mathtt{Y}_k$ given the 3D structure of the captured scene $\mathtt{Z}$. As stated in Equation 2.22, the distribution $P(\vec{x}_k|\mathtt{Y}_{1:k}, \mathtt{Z})$ is approximated as a set of weighted hypotheses $S = \left\{\left(\vec{x}_k^{(1)}, w_k^{(1)}\right), \ldots, \left(\vec{x}_k^{(m)}, w_k^{(m)}\right)\right\}$ where each weight can be calculated using Equation 2.24. Taking into account that the system described here follows a Markov chain model, the proposal density $Q\left(\vec{x}_k^{(i)}|\mathtt{X}_{k-1}, \mathtt{Y}_k\right)$ can be chosen to be $P(\vec{x}_k|\vec{x}_{k-1})$. Substituting in Equation 2.24 the weights of each sample can be updated as follows:

$$w_k^{(i)} \propto w_{k-1}^{(i)} P(\mathtt{Y}_k|\vec{x}_k^{(i)}, \mathtt{Z}). \tag{3.12}$$

Each hypothesis represents a possible evolution of the model and they can be directly drawn from the transition function that describes the state-space model. Similarly, the observation density $P(\mathtt{Y}_k|\vec{x}_k^{(i)}, \mathtt{Z})$ can be evaluated using the stochastic observation model given in Equation 2.11. Figure 3.11 shows a summary of the steps involved in the pose estimation.

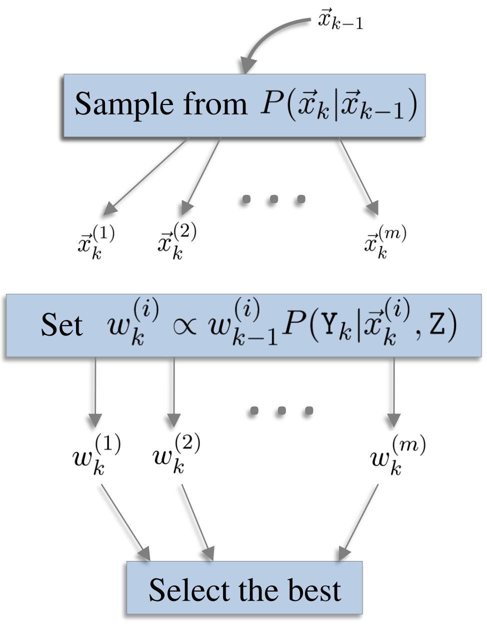

$\vec{x}_{k-1}$
Sample from $P(\vec{x}_k|\vec{x}_{k-1})$
$\vec{x}_k^{(1)}$
$\vec{x}_k^{(2)}$
$\vec{x}_k^{(m)}$
Set $w_k^{(i)} \propto w_{k-1}^{(i)} P(\mathtt{Y}_k|\vec{x}_k^{(i)}, \mathtt{Z})$
$w_k^{(1)}$
$w_k^{(2)}$
$w_k^{(m)}$
Select the best

Figure 3.11: Description of the pose estimation algorithm.

In order to work efficiently in the GPU platform, some modifications have been done to the sampling and weighting steps. Following sections describe in detail how these distributions are approximated.

3.4.1 Pose Sampling

In each frame k a set of samples $A = \left\{\vec{x}_k^{(1)}, \ldots, \vec{x}_k^{(m)}\right\}$ is drawn from the proposal $P(\vec{x}_k|\vec{x}_{k-1})$. As discussed in Section 2.4.2, there are various alternatives to the transition function like constant velocity or constant acceleration that guide the sampling procedure to specific high probability areas. However, in order to avoid problems with erratic motion patterns introduced by handheld cameras, hypotheses are generated using the random walk sampling model, also known as constant position sampling.

In this way, new samples are generated as random rotations and translations from the previous pose $\vec{x}_{k-1}$, so the sampling function is:

$$\vec{x}_k^{(i)} = f\left(\vec{x}_{k-1}, \vec{\omega}_i, \vec{\tau}_i\right) = \left[\vec{q}_{k-1} + \vec{\omega}_i, \vec{t}_{k-1} + \vec{\tau}_i\right] \tag{3.13}$$

where $\vec{\omega}_i$ and $\vec{\tau}_i$ are samples taken from random variables following any distribution. This is an important difference with respect to Kalman filters, since the Gaussianity is not mandatory.

In this case, the transition function is linear, but this filter can work with nonlinear functions as well without any special consideration. Note that the perturbation model applied to the quaternion is a simple vector addition (and a posterior normalization) instead of a quaternion composition. Compared to a normal quaternion composition, this perturbation model has a major impact on the rotation axis rather than on its magnitude. Experimentally it has been concluded that it describes better the behavior of a handheld camera, and in the same time, it is the simplest way to update its orientation.

Due to purely practical reasons, $\vec{\omega}_i$ and $\vec{\tau}_i$ have been chosen to be drawn from normally distributed random variables $\Omega \sim N\left(\vec{0}, \Sigma_{rot}\right)$ and $T \sim N\left(\vec{0}, \Sigma_{trans}\right)$ so that the proposal $P(\vec{x}_k|\vec{x}_{k-1})$ follows a Gaussian distribution as well. In order to delimit the sampling space, the covariance matrices Σ_{rot} and Σ_{trans} should be carefully chosen. These covariances are modeling the uncertainty of the pose parameters, so they need to be large enough to cover all the space of possible camera configurations. However, a large region would lead to imprecise estimations introducing jitter into the system, while a small region would introduce drift in the estimation.

Since rotations between frames are very small even with erratic motions, the covariance matrix Σ_{rot} is fixed for all frames. Note that an assumption of 10 degrees per frame means a total of 250 degrees per second at 25 fps. Faster rotations would lead to excessive blur that would make the sequence non-trackeable.

However, the magnitude of the translation can vary a lot between frames, so the covariance matrix Σ_{trans} is updated every frame. To calculate this matrix, it is assumed that the camera only translates locally. In this way, taking Equation 3.1, the projection of a 3D point can be approximated as:

$$\vec{y}_i^k \approx \Pi\big(\mathbf{R}_{k-1}\vec{z}_i + \hat{t}_k\big) \tag{3.14}$$

where the vector $\hat{t}_k$ is the only variable in this translational camera model. Since there are no rotations contributing to the pixel displacement of the projections of the scene points, the translation obtained from this model can be considered as the limit of the uncertainty region of the real translation. In order to estimate this translational model, it can be expressed linearly using the Taylor expansion of Equation 3.14:

$$\vec{y}_i^k \approx \vec{y}_i^{k-1} + \mathbf{J}_{k-1}\left(\hat{t}_k - \vec{t}_{k-1}\right) + \mathrm{HOT} \tag{3.15}$$

where $\mathbf{J}_{k-1}$ is the Jacobian matrix of Equation 3.14 evaluated at $\vec{z}_i$ with respect to translation and HOT are the Higher Order Terms of the Taylor expansion. As can be seen, this linearized model is quite similar to the measurement function used by the Extended Kalman Filter shown in Equation 2.21. In fact, assuming a constant position model, the *a priori* estimate $\tilde{t}_k$ would be $\vec{t}_{k-1}$, complying with the form of the Extended Kalman Filter.

Equation 3.15 includes in the measurement vector all the tracked feature points contained in the set $\mathtt{Y}_k$. However, since the translational model used here is just an approximation of the real dynamics of the system, it can be simplified for efficiency using only a single feature point. In order to get robustness against outliers and other errors, instead of using a real feature point, the projection of the centroid of the visible 3D points has been used. The measurement corresponding to this point is taken as the average pixel displacement of the visible feature points, so the final measuring function is:

$$\vec{y}_c^k \approx \tilde{y}_c^k + \mathbf{J}_{k-1}\left(\hat{t}_k - \vec{t}_{k-1}\right) + \mathrm{HOT} \tag{3.16}$$

where $\tilde{y}_c^k$ is the projection of the centroid $\vec{z}_c$, and $\vec{y}_c^k$ is the measured pixel displacement of the visible feature points.

The estimated $\hat{t}_k$ can be obtained as a Gaussian random variable using the update equations of the Kalman Filter. First, the *a priori* estimate of the translation is obtained using the time update function, that as mentioned, it has been chosen to be a constant position model:

$$\tilde{t}_k = f(\vec{t}_{k-1}, \vec{0}) = \vec{t}_{k-1}. \tag{3.17}$$

The covariance matrix $\mathbf{P}_k^{(-)}$ of this prediction is approximated as the translational velocity of the camera in the last frame. From this matrix, the gain of the measurement update equation can be calculated from Equation 2.19 replacing the measurement matrix by the Jacobian of the observation function:

$$\mathbf{G}_k = \mathbf{P}_k^{(-)} \mathbf{J}_{k-1}^{\top} \left(\mathbf{J}_{k-1} \mathbf{P}_k^{(-)} \mathbf{J}_{k-1}^{\top} + \mathbf{R}_{k-1} \right)^{-1} \tag{3.18}$$

where $\mathbf{R}_{k-1}$ is the covariance matrix of the measurement process. Since this covariance models the uncertainty of the feature tracker, as stated in Section 3.2, it is set as $\mathbf{R}_{k-1} = \mathrm{diag}(2)$.

Having the measurement $\vec{y}_c^k$, the *a posteriori* state is obtained from Equation 2.18, but using Equation 3.14 as the nonlinear measurement function $h()$:

$$\hat{t}_k = \tilde{t}_k + \mathbf{G}_k \left(\vec{y}_c^k - h(\tilde{t}_k, \vec{0}) \right), \tag{3.19}$$

and its covariance can be updated as:

$$\mathbf{P}_k = (\mathbf{I}_3 - \mathbf{G}_k \mathbf{J}_{k-1}) \mathbf{P}_k^{(-)}. \tag{3.20}$$

A natural solution would be to approximate $P(\vec{x}_k | \vec{x}_{k-1})$ sampling the translation directly from this Gaussian instead of using Equation 3.13, however, the solution would be biased in the direction of the estimation of the Kalman filter.

Having the estimation of the translational model, the covariance of the random variable T can be set as:

$$\Sigma_{trans} = \mathrm{diag}\left(\mathrm{abs}(\hat{t}_k - \vec{t}_{k-1}) \right) + \mathbf{P}_k. \tag{3.21}$$

3.4.2 Weighting the Pose Samples

Each sample is weighted using Equation 3.12, which only depends on the marginal probability of the observation given the state hypothesis, i.e., $P\left(\mathtt{Y}_k | \vec{x}_k^{(i)}, \mathtt{Z} \right)$. As stated, this probability is set proportional to the observation model of the system given by Equation 2.11.

In this work, a bottom-up approach is used, and the observation density is approximated as the Euclidean distance between point projections given the current hypothesis and corresponding 2D features:

$$P\left(\mathtt{Y}_k|\vec{x}_k^{(i)},\mathtt{Z}\right) \propto \exp\left(-\sum_{j=1}^{n}\left\|\Pi\left(\mathbf{R}_k^{(i)}\vec{z}_j+\vec{t}_k^{(i)}\right)-\vec{y}_j^k\right\|\right) \tag{3.22}$$

where $\mathbf{R}_k^{(i)}$ is the rotation matrix corresponding to $\vec{q}_k^{(i)}$. Other authors like (Pupilli and Calway, 2006) use the inlier-outlier likelihood, approximating the distribution as:

$$P\left(\mathtt{Y}_k|\vec{x}_k^{(i)},\mathtt{Z}\right) \propto \exp\left(-\sum_{j=1}^{n}\mathrm{d}\left(\vec{y}_j^k,\vec{z}_j,\vec{x}_k^{(i)}\right)\right) \tag{3.23}$$

where $\mathrm{d}()$ is a distance function defined by:

$$\mathrm{d}\left(\vec{y}_j^k,\vec{z}_j,\vec{x}_k^{(i)}\right) = \begin{cases} 1 & \text{if} \quad \left\|\Pi\left(\mathbf{R}_k^{(i)}\vec{z}_j+\vec{t}_k^{(i)}\right)-\vec{y}_j^k\right\| > \epsilon_d \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

The value ϵ_d corresponds to a threshold value that indicates if the point is an inlier or not. This observation density produces good results as well, however the estimated motion is less smooth and an annealing step is required. These two weighting functions are shown in Figure 3.12.

The pose that is used for rendering virtual objects is chosen to be the one with the greatest weight. Moreover, resampling is done each frame since it has no sense in bottom-up schemes. The reason is that having 2D features already matched, there is no need for multimodality.

The likelihood of the frame is calculated as a threshold function that depends on the number of inliers that the sample gets. If there are enough inliers, the frame is considered a keyframe, and it will be used by the structure adjustment algorithm. In addition, only points that have been inliers in the previous frames are used for the pose estimation. There are more complex methods for keyframe selection in the literature, but this solution is easy to implement and gets good results with this random sampling method.

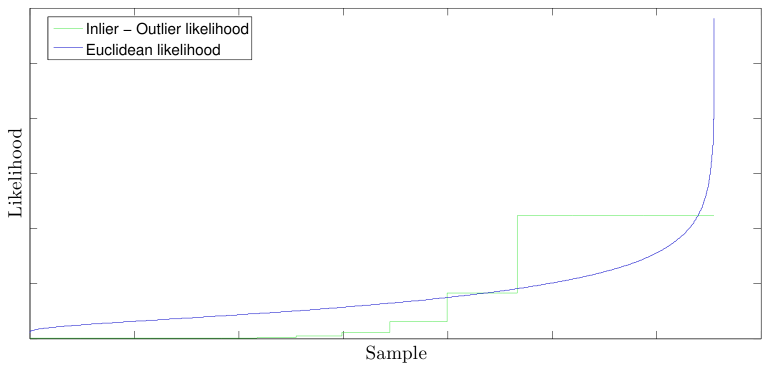


Figure 3.12: Weighting likelihoods compared.

Finally, in order to get a smooth motion and avoid jitter, the pose that is used to render virtual objects is filtered using a *desp filter* (LaViola, 2003) that maintains a record of the previous poses in order to get the smoothed estimate.

3.5 Scene Reconstruction

Following the same argumentation as in the pose estimation, the 3D structure map is reconstructed using SMC as well. The structure estimation has two major parts, i.e. point initialization and point adjustment. New points are added constantly to the map as described in Section 3.3, and mapped points are adjusted using the information from new frames.

Structure points are parameterized as 3-vectors, so that $\vec{z}_j = [x_j, y_j, z_j]^T$. Every frame, a list of points having their reprojection error greater than a threshold is built. For each of those points, the adjustment process is executed iteratively.

In this case, it makes no sense to talk about a state-space model, since the scene is considered to be rigid and hence invariant along the sequence. However, the same approach that has been employed in the motion estimation can be used if a single point is considered a dynamic system whose transition function is a random walk model.

Under this assumption, the distribution $P(\vec{z}_j|\mathtt{Y}_{1:k}, \mathtt{X}_{1:k})$ can also be approximated as a set of weighted samples. The proposal density is chosen to be $P(\vec{z}_j|\vec{z}_j^{(-)})$ where $\vec{z}_j^{(-)}$ denotes the point location in the previous frame, so the weights of the samples are calculated as:

$$w_j^{(i)} \propto P(\mathtt{Y}_{1:k} | \vec{z}_j^{(i)}, \mathtt{X}_{1:k}). \tag{3.25}$$

The observation density is approximated using the measurement model given in Equation 2.11. This can be seen as a nonlinear minimization that is computed for each structure point. A summary of these steps is illustrated in Figure 3.13. Following sections describe in detail how these distributions are approximated.

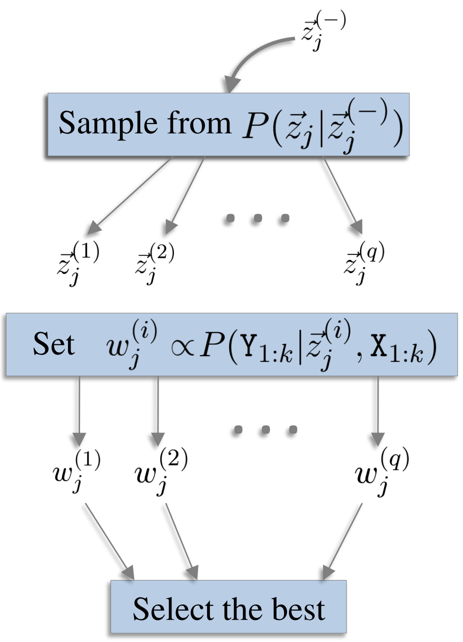

$\vec{z}_j^{(-)}$
Sample from $P(\vec{z}_j|\vec{z}_j^{(-)})$
$\vec{z}_j^{(1)}$
$\vec{z}_j^{(2)}$
$\vec{z}_j^{(q)}$
Set $w_j^{(i)} \propto P(\mathtt{Y}_{1:k}|\vec{z}_j^{(i)}, \mathtt{X}_{1:k})$
$w_j^{(1)}$
$w_j^{(2)}$
$w_j^{(q)}$
Select the best

Figure 3.13: Description of the point adjustment algorithm.

3.5.1 Point Sampling

For each point $\vec{z}_j$, a set of random samples $B = \left\{\vec{z}_j^{(1)}, \ldots, \vec{z}_j^{(q)}\right\}$ is generated around its neighborhood using a random walk transition function:

$$\vec{z}_j^{(i)} = f\left(\vec{z}_j^{(-)}, \vec{\psi}_i\right) = \vec{z}_j + \vec{\psi}_i \tag{3.26}$$

where $\vec{z}_j^{(-)}$ is the prior location of the point being adjusted and ψ is a sample taken from a random variable following any distribution. Again, Gaussianity is not mandatory, however a normally distributed variable $\Psi \sim N\left(\vec{0}, \Sigma_j\right)$ has been used due to its simplicity, so the proposal $P(\vec{z}_j|\vec{z}_j^{(-)})$ is also normal.

The covariance matrix of the noise distribution can be bounded using the method described for the covariance of the translation vector in the pose sampling function. Following the same argumentation, the projection of the point $\vec{z}_j$ can be linearized around its prior location $\vec{z}_j^{(-)}$ using the first order Taylor expansion:

$$\vec{y}_j^k = \hat{y}_j^k + \mathbf{J}_j^k\left(\vec{z}_j - \vec{z}_j^{(-)}\right) + \text{HOT} \tag{3.27}$$

where $\mathbf{J}_j^k$ is the Jacobian of Equation 3.1 evaluated in the last frame with respect to $\vec{z}_j^{(-)}$, and $\hat{y}_j^k$ is its projection.

The solution can be estimated using the same update functions detailed in Section 3.4. In this case, the covariance matrix of the time update function is set as the sample covariance of the point in the previous adjustment step. In Section 3.5.2 will be shown in detail how to compute this matrix.

The output of this filter is an estimation $\hat{z}_j$ about the new position of the point. However, it should be noted that only the information of the last view is used in Equation 3.27. This implies that the prediction of the filter will be optimal (assuming linearity and Gaussianity) only with respect to the last observation and not globally optimal for the whole sequence, and thus incorrect. However, this prediction can be used as the bounds of the sampling function, setting the covariance matrix of Ψ as:

$$\Sigma_j = \text{diag}\left(\text{abs}(\hat{z}_k - \vec{z}_j^{(-)})\right) + \mathbf{P}_k, \tag{3.28}$$

where $\mathbf{P}_k$ is the covariance matrix of the estimation of the filter.

3.5.2 Weighting the Point Samples

For each sample, the point is projected along the past keyframes and its weight is calculated using Equation 3.25. There are a lot of possible choices in order to approximate the distribution $P(\mathtt{Y}_{1:k}|\vec{z}_j^{(i)}, \mathtt{X}_{1:k})$, like inlier-outlier likelihoods or more complex distance functions. Here, it is set proportional to the Euclidean distance between the projections of the sample in the previous keyframes and the 2D features measured in those keyframes:

$$P(\mathtt{Y}_{1:k}|\vec{z}_j^{(i)}, \mathtt{X}_{1:k}) \propto \exp\left(-\sum_{l=1}^{k}\left\|\Pi\left(\mathbf{R}_l\vec{z}_j^{(i)} + \vec{t}_l\right) - \vec{y}_j^{\,l}\right\|\right), \tag{3.29}$$

which corresponds to a bottom-up approach. The solution is taken as the point with the greatest weight. However, as mentioned in Section 3.5.1, the covariance of the sample set is needed in order to estimate the search space for the sampling function in the next adjustment step. In order to obtain this covariance, first the resulting weights need to be normalized so that $\sum_{i=1}^{q}\hat{w}_j^{(i)} = 1$. Having these weights, the mean $\bar{z}$ and the covariance $\mathbf{S} = [s_{u,v}]$ of the sample can be obtained as:

$$\bar{z} = \sum_{i=1}^{q}\hat{w}^{(i)}\vec{z}^{(i)} \tag{3.30}$$

$$s_{u,v} = \frac{\sum_{i=1}^{k}\hat{w}^{(i)}\left(z_u - \bar{z}_u\right)\left(z_v - \bar{z}_v\right)}{1 - \sum_{k=1}^{N}\left(\hat{w}^{(i)}\right)^2}, \quad 1 \leq u, v \leq 3 \tag{3.31}$$

where z_u denotes the u^{th} component of the vector $\vec{z}$. Note that the subscript relating the point index has been omitted for readability.

The covariance information results very useful in order to detect bad estimations. This covariance is describing an ellipsoid in the space where the true position of the point is with a high probability. As new measures become available, the size of this ellipsoid decreases, which means that the estimate is becoming stable. The axes of the ellipsoid can be obtained as the eigenvectors of the matrix $\mathbf{S}$ and the magnitude as its eigenvalues. The eigenvalues can be used, for example, to determine which points are the best candidates for the pose estimation.

If after the adjustment the posterior residual is less than the prior and minor than a threshold, the point in the map is modified. If not, the point is considered as an outlier and removed from the structure map.

3.6 Remapping

The tracking of an image feature can be lost due to noise or simply because it disappears from the image due to camera motion. When this happens, the corresponding 3D point in the map is marked as lost. In each frame these lost points are projected using the pose information estimated by the 3D tracker. If the projections are located within the image, the system tries to remap them.

Since each feature has a SIFT descriptor associated, a fixed size window is used around the projection of the point being remapped looking for a feature having a similar descriptor. Figure 3.14 illustrates an example of a point being remapped. In this example, the stored descriptor is compared with the descriptor of the five features located inside the window.

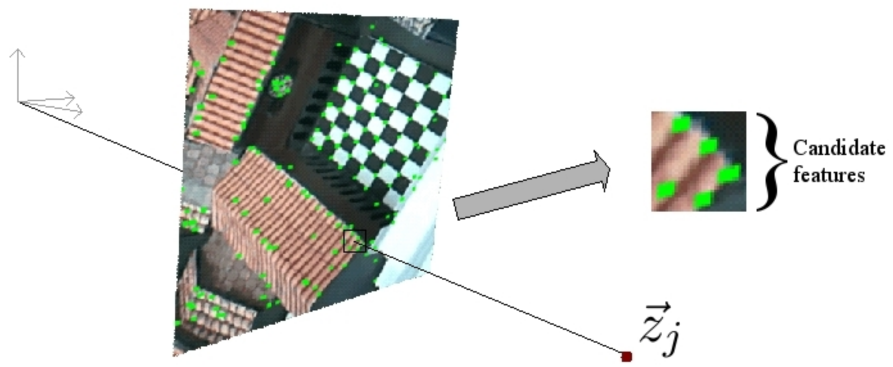


Figure 3.14: A lost point being remapped.

If the difference between the best match and the second best is greater than a threshold, the rematching is considered stable and the point is remapped. Algorithm 3.1 shows the summary of the remapping strategy:

Algorithm 3.1 Remapping strategy.

```
Require: $W \rightarrow$ Remapping window.
Require: $\vec{d_i} \rightarrow$ descriptor of the point remapped.
  F $\leftarrow$DetectFeaturePoints($W$)
  bestDifference $\leftarrow \infty$
  secondBest $\leftarrow \infty$

  for all $\vec{p}$ in F do
    diff=descriptor($\vec{p}$)-$\vec{d_i}$
    if diff<bestDifference then
      secondBest $\leftarrow$ bestDifference
      bestDifference $\leftarrow$ diff
    else if diff<secondBest then
      secondBest $\leftarrow$ diff
    end if
  end for

  if bestDifference>threshold and bestDifference> $k*$secondBest then
    remmap($\vec{z_i}$)
  end if
```

3.7 Experimental Results

The accuracy of the method described in this chapter is analyzed in this section. In this way, various tests have been done using both synthetic and real data.

Synthetic data consist on a virtual video sequence rendered using a camera moving around a virtual object. The object is 1 m wide and initially it is located at 3 m from the camera center. The projections of its vertices simulate the 2D features that act as the input of the algorithm, allowing to analyze its behavior without introducing in the experiments the bias produced by the feature tracker. Anyway, this uncertainty is simulated adding noise to the projections. Figure 3.15 shows this virtual scene while Table 3.1 summarizes some of the parameters used to render it.

Real sequences have been recorded using a Logitech webcam working at 320×240. For these tests, two video sequences have been used. The first video, shown in Figure 3.22, consists in a desktop size indoor scenario composed by

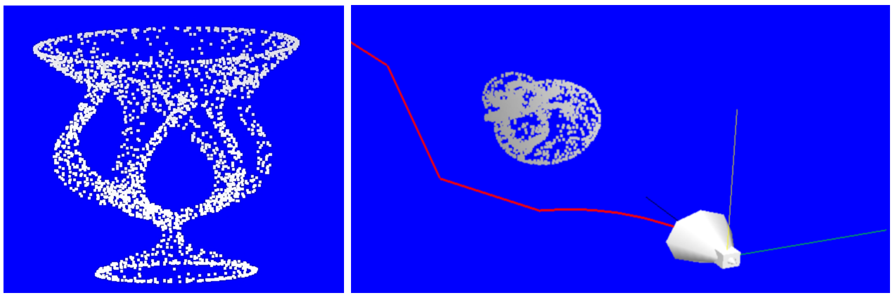

(a) Virtual object. (b) Camera path.

Figure 3.15: The synthetic data used for the experiments.

a mock-up of a city. This scenario contains a lot of textured shapes and a very structured geometry. In contrast, the second video, shown in Figure 3.23, is an outdoor sequence consisting in a more homogeneous environment. The ground truth values of these videos has not been measured, therefore the errors are expressed as the reprojection residuals of the estimated structure using the estimated pose, compared to the image measurements given by the feature tracker.

Number of frames	100
Frame rate	25 fps.
Focal length	500 pix.
View port	320x240
Tracked features	450

Table 3.1: Parameters of the synthetic video.

Unless otherwise stated, 512^2 samples are used for the pose estimation and 16^3 for the structure estimation.

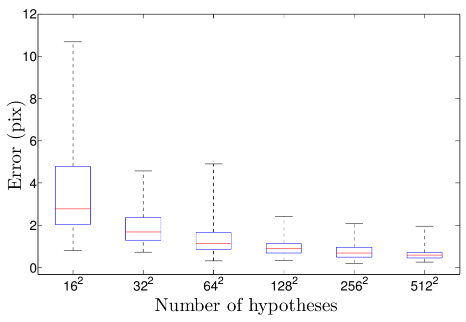
3.7.1 Pose Estimation Accuracy

In order to establish the number of hypotheses needed to converge successfully, the pose estimation algorithm has been tested independently from the structure reconstruction part. Using the synthetic sequence, the pose of the camera has been computed for each frame using different amount of hypotheses in each experiment. Moreover, two versions of the test sequence have been used, i.e., a clean sequence without any perturbation in the input data, and a perturbed sequence with Gaussian noise added to the positions of the 2D feature points. The mean is set as 0 and the covariance as 2 for each dimension, following the assumptions made in Section 3.2 about the uncertainty associated to the 2D tracker. In addition, a 20% of the matches are intentionally modified in order to test the robustness against outliers.

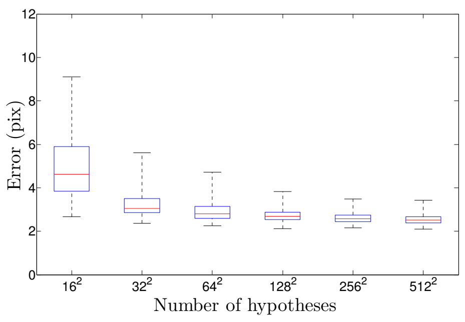
The motion estimator is fed using 100 structure points leaving the rest for measuring the errors. The resulting camera path is compared with the ground truth data and the error is measured as the Euclidean distance between the projections of the structure points using the estimated path and the positions of the 2D features given by the synthetic sequence:

$$\epsilon(i) = \frac{\sum_{j=1}^{m} \left\| \Pi\left(\mathbf{R}_i \vec{z}_j + \vec{t}_i\right) - \vec{y}_j^{\,i} \right\|}{m}. \tag{3.32}$$

Figure 3.16 shows the results of these experiments as a box plot. On each box, the central mark is the median of the reprojection error, the edges of the box are the 25th and 75th percentiles and the whiskers extend to the maximum and minimum values. Figure 3.16a corresponds to experiments done using the clean data, while Figure 3.16b shows the results of the perturbed sequence.



(a) Without noise in the input data.



(b) With additive noise and outliers.

Figure 3.16: Pose estimation accuracy in terms of the reprojection error.

As seen, the dispersion from the median decreases with the number of samples taken. This is the normal behavior of a Monte Carlo approximation, since the target distribution is better described as the number of samples increases.

From the perturbed data, it could be inferred that the pose estimation error increases with the noise in the feature points. However, it is not necessarily true since the projections are compared against the noisy feature points. In contrast, Figure 3.17 shows the difference between the real and the estimated camera paths using the perturbed sequence and a configuration of 256^2 hypotheses. One plot for each translation and rotation axis is shown.

As can be seen, the estimated motion is very near to the ground truth data even with a relatively high reprojection error. Moreover, due to the random walk sampling function, the algorithm can handle very well erratic motion patterns like the sudden change of direction that appears in the frame 50.

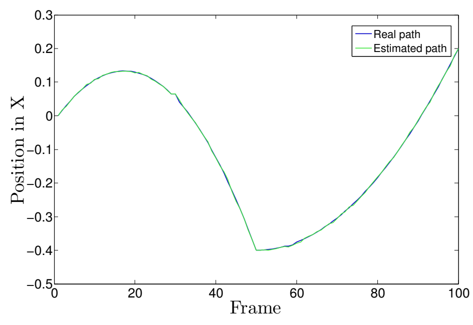
Table 3.2 shows a resume of the tests done with the synthetic data. Each column contains the Mean Squared Error (MSE) and the standard deviation of the pose estimation resulting from perturbing the sequence with additive Gaussian noise. The covariance of the noise is diagonal and its magnitude is specified by the header of the column.

Table 3.2: Accuracy measured in a synthetic sequence.

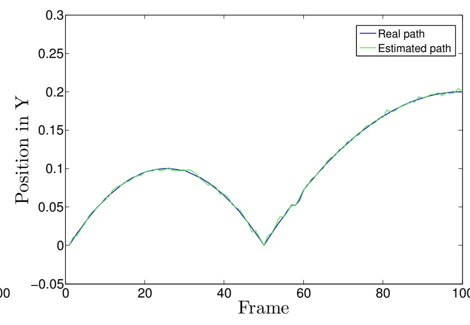
		Noise 2	**Noise 4**	**Noise 6**	**Noise 8**
Tx	**MSE**	4.62×10^{-6}	1.05×10^{-5}	1.98×10^{-5}	2.66×10^{-5}
	Std. dev.	7.61×10^{-6}	1.55×10^{-5}	2.91×10^{-5}	3.62×10^{-5}
Ty	**MSE**	3.93×10^{-6}	9.91×10^{-6}	1.92×10^{-5}	2.94×10^{-5}
	Std. dev.	6.41×10^{-6}	1.40×10^{-5}	2.72×10^{-5}	4.57×10^{-5}
Tz	**MSE**	1.30×10^{-4}	3.79×10^{-4}	4.73×10^{-4}	8.86×10^{-4}
	Std. dev.	1.88×10^{-4}	7.03×10^{-4}	6.85×10^{-4}	1.30×10^{-3}
Rx	**MSE**	2.19×10^{-5}	7.18×10^{-5}	1.05×10^{-4}	1.65×10^{-4}
	Std. dev.	2.42×10^{-5}	1.15×10^{-4}	1.44×10^{-4}	2.57×10^{-4}
Ry	**MSE**	3.57×10^{-5}	6.33×10^{-5}	1.25×10^{-4}	2.05×10^{-4}
	Std. dev.	5.13×10^{-5}	9.17×10^{-5}	2.27×10^{-4}	3.07×10^{-4}
Rz	**MSE**	2.51×10^{-5}	5.01×10^{-5}	1.16×10^{-4}	1.55×10^{-4}
	Std. dev.	4.07×10^{-5}	7.01×10^{-5}	1.65×10^{-4}	2.19×10^{-4}

The MSE represents the drift in the estimation and the standard deviation the jitter. As expected, both quantities increase with the noise. However, the tracking converges even with a noise of 8 pixels of standard deviation in the input.

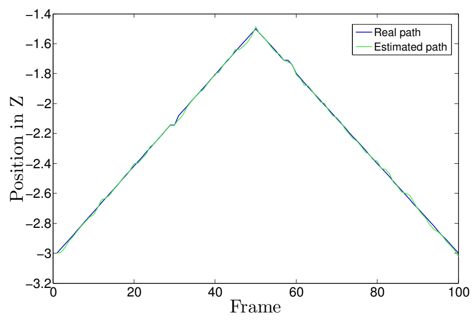
Resuming, it has been shown that the proposed estimator is robust enough to deal with erratic motions even with high levels of noise in the input data. However, these benefits are obtained in exchange for huge computational resources. Note that in these tests a total of 512^2 samples have been used.



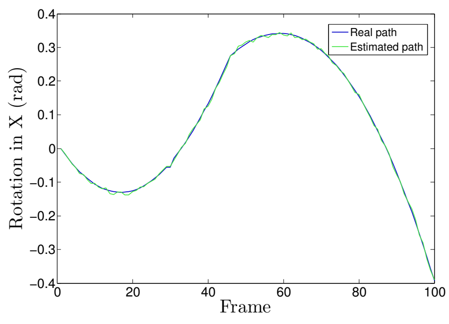
(a) Position comparison in X.



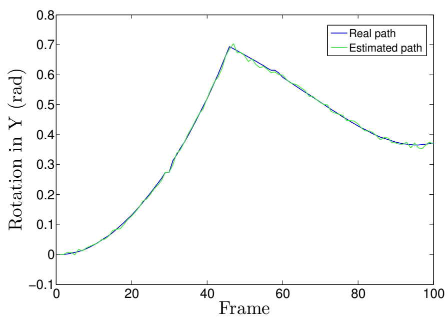
(b) Position comparison in Y.



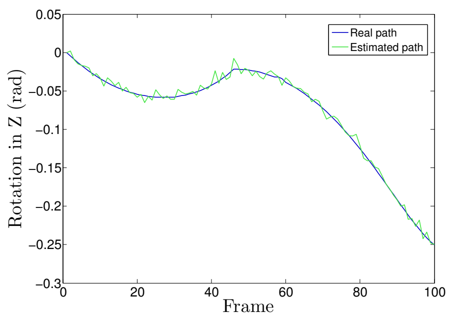
(c) Position comparison in Z.



(d) Rotation comparison in X.



(e) Rotation comparison in Y.

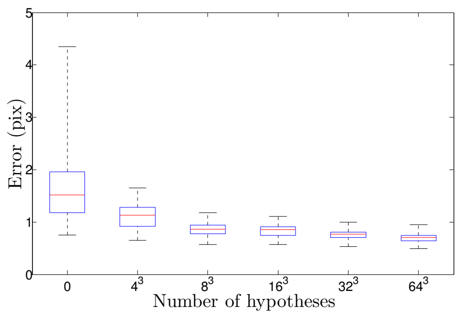


(f) Rotation comparison in Z.

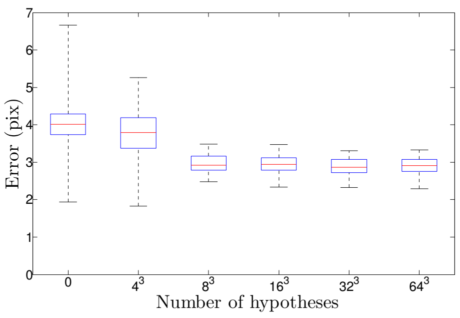
Figure 3.17: Estimated camera path compared to the real.

3.7.2 Scene Reconstruction Accuracy

Similarly to the experiments done with the pose estimation, the scene reconstruction part has also been evaluated using the same synthetic sequence. In order to measure the behavior of the algorithm with respect to the number of samples and not to bias the result with the noise introduced by the pose estimation, it has been tested separately using the ground truth pose information of the simulated camera. Figure 3.18 shows the results of the experiments using the clean sequence, and the sequence with the feature points perturbed with noise. The noise is set as a Gaussian with mean 0 and covariance 2. The errors are expressed as the reprojection error that the final structure has in each frame of the sequence using Equation 3.32.



(a) Without noise in the input data.



(b) With additive noise and outliers.

Figure 3.18: Reconstruction accuracy in terms of the reprojection error.

The first box of each plot corresponds to the structure obtained by linear triangulation without applying any adjustment to it. From both figures 3.18a and 3.18b follow that the filter performs better as the number of hypotheses used increased. However, as can be seen in Figure 3.18b, depending on the input data, the reconstruction can converge to the optimum with less hypotheses. It should be noticed that the minimum reprojection error increases after the optimization. However, it is normal, since reducing the overall error usually means that the error in some frames gets incremented.

These results demonstrate that the method is effective reaching the minimization objective, that depends on the reprojection error. However, it not gives information about the accuracy of the estimation. In order to measure this accuracy, the estimated structure has been compared with the real, available from the simulated data. This test has been done with the noisy dataset, using a total of 16^3 hypotheses. The resulting reconstruction has 637 points. In order to establish a comparison, both structures are scaled and displaced so their centroid is at the origin and the RMS distance of the points to the origin is $\sqrt{3}$. This implies that the mean distance of the points to the origin is 1, so the Euclidean distance between the estimated points and the ground truth points can be interpreted as a percentage. Figure 3.19 shows the histogram of errors between the two point clouds.

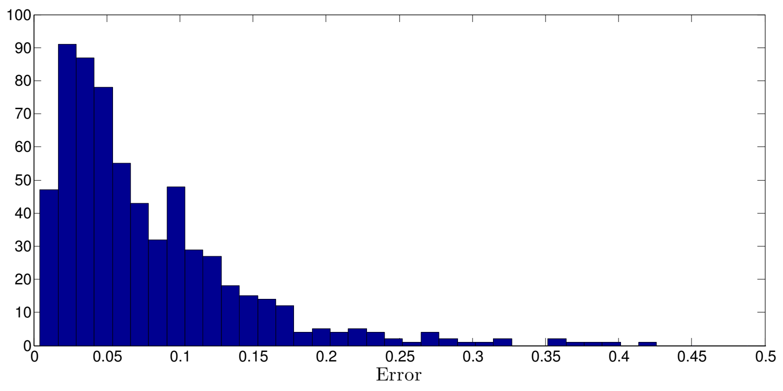


Figure 3.19: Histogram of the reconstruction error.

As seen in the results, the mean reconstruction error is around 2.5% and the number of outliers is very low. These values are acceptable for augmented reality applications, however they may not be sufficient for other applications, like robot navigation.

3.7.3 SLAM Accuracy

In order to evaluate the full SLAM application, some results using the real video sequences are presented in this section. Moreover, the accuracy of the proposed method has been compared with the popular implementation of BA given in (Lourakis and Argyros, 2009).

Starting with the indoor sequence, the reprojection error of both algorithms is shown in Figure 3.20. The tests are executed using the same feature tracker and the same structure initialization. BA is parameterized with the default configuration given by the author. The projection errors are measured using the state of the SLAM algorithm in the last frame of the video using Equation 3.32. From the plot follows that the proposed method performs better than BA. However, it should be noted that since outlier detection and keyframe selection depend on the output of the minimization algorithm, the SLAM sequence evolves differently in each case. This means that the final reconstruction has different amount of points, different scale, etc. Moreover, the difference between them is about 1-2 pixels, so it could be said that they obtain the same precision level.

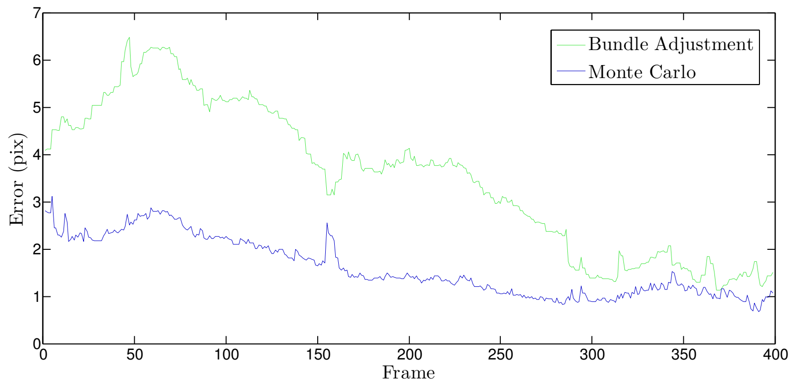


Figure 3.20: Projection error comparison.

Finally, Figure 3.21 shows the reprojection error in the outdoor sequence. It can be seen that even with a bad initialization, the error drops to a few pixels quickly.

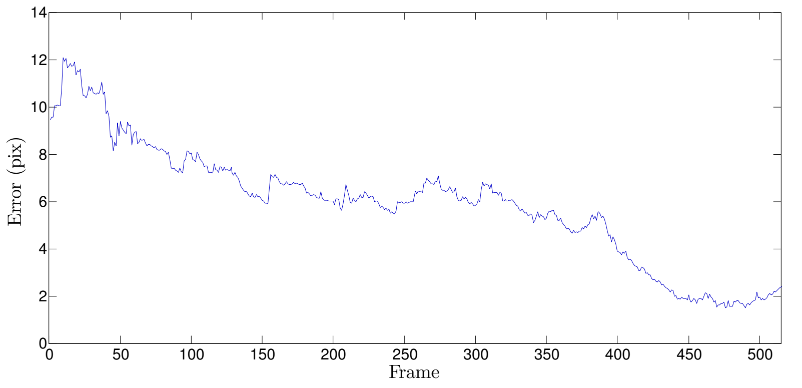


Figure 3.21: Projection error in pixels in the outdoor scene.

3.8 Discussion

In this chapter a full SLAM method has been described that uses Monte Carlo simulations to estimate separately both the motion of the camera and the structure of the scene. Some experiments are also provided, demonstrating the suitability of the method. The design of these filters has involved two main challenges: the design of a good sampling function and a representative weighting function.

Normally, the dynamics of the systems modeled are very predictable, like in the case of a robot for example. However, a handheld camera is more complex, since its motion cannot be predicted and its uncertainty cannot be modeled as easily as other sensors. In contrast, structure points are easier to model, because strictly speaking they remain fixed in the entire sequence. However, it has been shown that it is very practical to model them as dynamic particles in order to apply the same tools used for the motion estimation.

In the case of point estimation, the sample covariance has been successfully used to guide the sampling procedure of the next estimation. On the other hand, it is not calculated for the pose, since the erratic nature of a handheld camera makes it impractical and it can be reasonably approximated by its velocity. However, it could be a good clue about the quality of the tracking and could help detecting degenerate estimations, like those derived from blurry images. Computationally speaking, calculating the sample covariance also implies calculating the sample mean, which involves iterating twice over the sample set. Taking into account that the number of samples needed for the pose estimation is very high compared with the structure estimation, it becomes quite expensive to do.

Regarding to the weighting functions, the pixel projection error has been selected as the representative measurement of the likelihood. Since there is no more information available in the images, the decision to take only refers to the metric to be used. This work uses the Euclidean distance, however, there exist more robust metrics that can also be used. Taking into account that the probability distributions of all the parts can be approximated from the weighted samples and propagated along the transforms, the Mahalanobis distance could be a good solution for future improvements of the method.

A final consideration can also be done about the number of samples used to approximate the distributions. In the case of the pose estimation, the motion is very smooth in most frames, so the result can be obtained using a few samples. In the same way, in the structure adjustment, when the point is near to the optimum the number of samples could be reduced as well. These facts can be exploited modifying online the number of hypotheses used in each frame, reducing the time needed to converge in many cases. In fact, there are already some works in this line that could be adapted for this purpose (Fox, 2003).

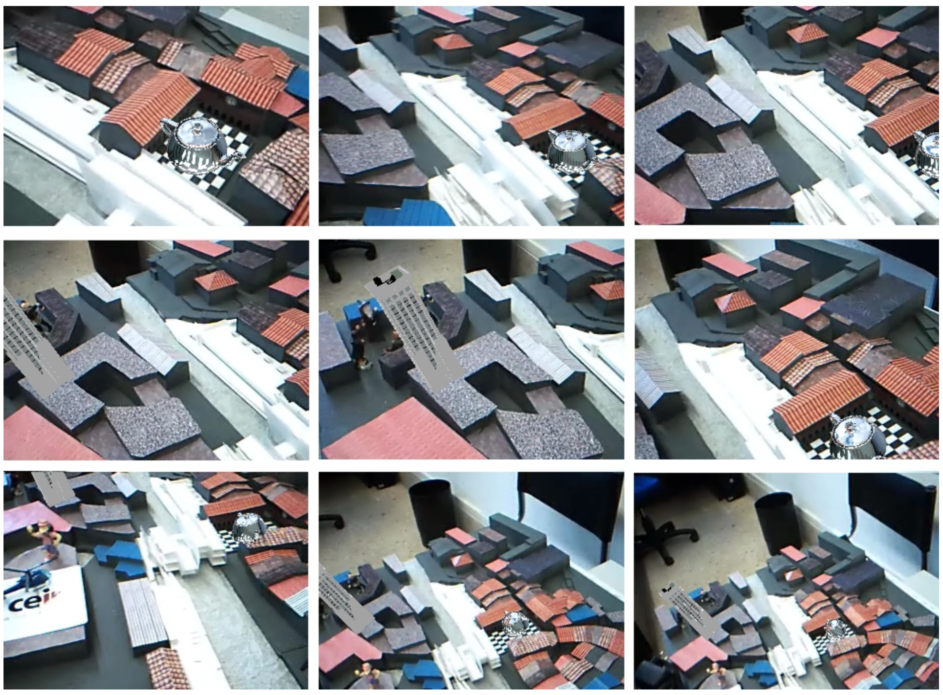
Summarizing, the framework given can be used to port any minimization problem to a GPU. It should be done iteratively following the next steps:

1. Determine the Jacobian of the objective function and evaluate it for the current candidate solution.

2. From the Jacobian, determine the covariance for the sampling function using the residual error and the measurement update equations.

3. Get the sampling set using the desired sampling function.

4. Weight each sample using the objective function.


(a) Augmented video.

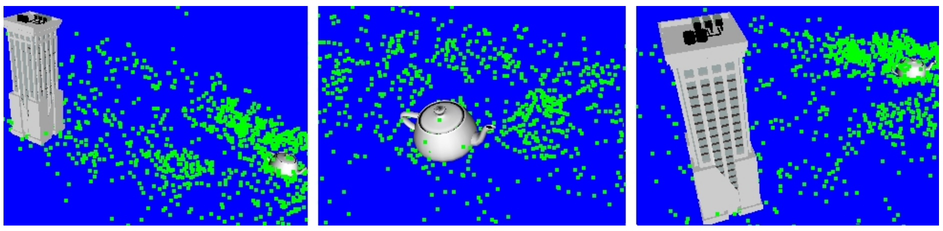
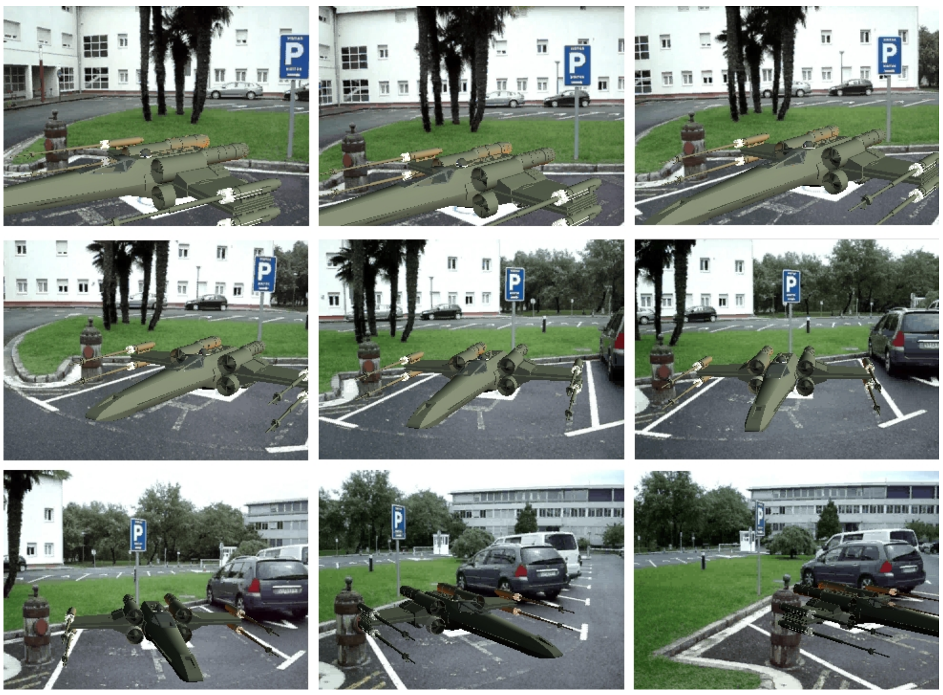

(b) Scene reconstruction.

Figure 3.22: Indoor sequence.


(a) Augmented video.

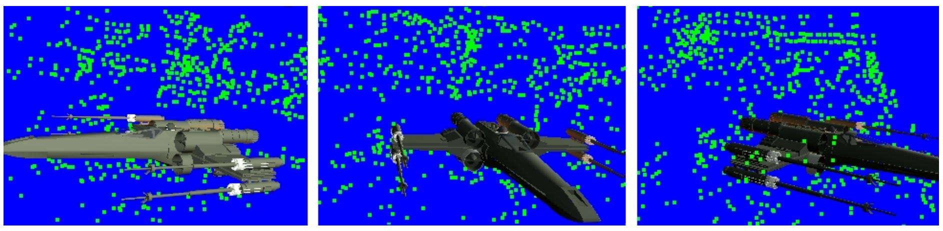

(b) Scene reconstruction.

Figure 3.23: Outdoor sequence.

Chapter 4

GPU Implementation

This section explains the details of the implementation of the algorithm in the parallel computing device. The reference implementation has been developed using the CUDA computing language (Kirk and Hwu, 2010), however it can be programmed using any other alternative. The main characteristics desired in a CUDA program (or any other GPU based computing language) are few communications with the host, no dependencies between parallel threads, non divergent functions (avoiding if-then-else constructors) and high arithmetic versus memory access ratio. Taking this into account, the next sections explain the implementation details of the pose estimation and point adjustment algorithms. For interested readers, an introduction to the GPU architecture is also given in Appendix C.

A synthesis of this chapter and Chapter 3 has been presented in:

Sánchez, J. R., Álvarez, H., and Borro, D. "Towards Real Time 3D Tracking and Reconstruction on a GPU Using Monte Carlo Simulations". In International Symposium on Mixed and Augmented Reality (ISMAR'10), pp. 185–192. Seoul, Korea. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. "GFT: GPU Fast Triangulation of 3D Points". In Computer Vision and Graphics (ICCVG'10), volume 6375 of Lecture Notes in Computer Science, pp. 235–242. Warsaw, Poland. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. "GPU Optimizer : A 3D reconstruction on the GPU using Monte Carlo simulations". In Poster proceedings of the 5th International Conference on Computer Vision Theory and Applications (VISAPP'10), pp. 443–446. Angers, France. 2010.

4.1 Program Overview

The implementation of the method has been divided in two main computing kernels, one for the pose estimation and other for the structure adjustment. The remaining parts of the SLAM pipeline, such as the feature tracker or the point triangulation, are implemented directly in the CPU. These two kernels run recursively every frame following the flow shown in Algorithm 4.1. The functions that are executed in the GPU are marked with the prefix “CU”.

Algorithm 4.1 Overview of the SLAM implementation.

```
$\mathtt{Y}_k \leftarrow$ trackFeatures($\mathtt{Y}_{k-1}$);
for all $\vec{z}_j$ in $\mathtt{Z}$ do
  if isVisible($z_j$) in current frame and isGoodForTrack($\vec{z}_j$) then
    markToTrack($\vec{z}_j$);
  end if
end for

$\vec{x}_k \leftarrow$ CUestimatePose($\mathtt{Y}_k$, $\mathtt{Z}$);
$\mathtt{Z} \leftarrow$ $\mathtt{Z} \cup$ triangulatePoints($\mathtt{Y_k}$, $\mathtt{X}_{1:k}$);

for all $\vec{z}_j$ in $\mathtt{Z}$ do
  $\vec{p}_j \leftarrow$ Projection($\vec{z}_j$, $\vec{x}_k$)
  if $norm(\vec{p}_j - \vec{y}_j^k) > threshold$ then
    $\vec{z}_j \leftarrow$ CUadjustPoint($\vec{z}_j$, $\mathtt{X}_{1:k}$, $\mathtt{Y}_{1:k}$)
  end if
end for
```

As shown, the pose estimation kernel runs once in each frame. Before its execution, the points that are going to be used for the estimation are marked depending on its visibility, and only if they have been inliers in the previous frames. Then, using the estimated pose, the structure estimation kernel is executed for each point whose reprojection error is above a threshold.

4.2 Memory Management

It is very important to choose correctly the memory structures used in CUDA kernels, since the main bottleneck of the graphics hardware is the bandwidth and the latency of its memory. The data that are frequently used should be stored in the constant memory, and the use of the global memory should be reduced as much as possible. Moreover, data transfers between the host and the device should be minimized, or at least try to make them asynchronously whenever possible due to the latency they have.

The set $\mathtt{Y_k}$ containing the feature points of the last tracked frame and the set $\mathtt{Z}$ with the 3D structure are stored in the constant memory of the device. There is no need to use special data structures, since the CUDA device can handle vector types. In this way, 2D features are stored in an array of 2-vectors, and 3D points in another array of 4-vectors (homogeneous representation). The information about the camera path is also stored in the constant memory as a rotation matrix and a translation vector for each frame. However, it must be taken into account that the amount of constant memory of the current hardware is very limited, making it necessary to keep only a subset of the data in the memory of the GPU.

The array of feature points must be updated every frame after making the image measurements. It is updated so that the indices of corresponding 3D-2D points in their arrays match. The time needed for each update is hidden making the data transfer asynchronous.

In contrast, the array containing the 3D structure remains constant along the sequence, except when a point is adjusted or when it is lost. In the first case, the point in the array must be updated, needing only one device-to-device copy, since the new point is computed in the CUDA device. In the second case, the lost point is swapped with the last point in the array, doing the same with its corresponding 2D feature in order to preserve the matching. These device-to-device operations are fast and they can be done asynchronously as well. New points are just added to the end of the array when they are triangulated.

The pose history is managed as a circular queue and new poses are added using a device-to-device transfer as well, after the execution of the pose estimation kernel.

Since the GPU lacks from random number generating functions, an array with random numbers is precomputed in the CPU and loaded into the memory of the CUDA device. This array is used in every frame without updating it. There are some algorithms to compute random numbers directly in the GPU, like (Howes and Thomas, 2007). However, as experiments demonstrate, this pseudo-random sampling gives good results avoiding the calculation of thousands of random numbers for each frame.

In the case of the pose sampling, two arrays are used. Each element of these arrays is a 4-vector representing the perturbations to the rotation quaternion and translation vector respectively (homogeneous representation). The size of these arrays is set as the number of hypotheses sampled in each frame. A standard normal distribution has been chosen since it can be easily modified using the covariance matrix computed in Equation 3.21, thus avoiding the updating of the arrays every frame.

Although this array is an input datum that acts as read only memory, due to size constraints it is stored in the global memory instead of in the constant memory. Generated hypotheses are also stored in the global memory, but in this case it is mandatory since the constant memory can only be used for read operations.

For the structure optimization, an alternative sampling approach has been used. Although in Section 3.5.1 it has been stated that the space is randomly sampled, in order to achieve better performance and drawing on the fact that mapped points have 3 degrees of freedom, the built-in trilinear texture interpolator of the CUDA device has been used to sample the position of each hypothesis. In this way, a 3 dimensional $3 \times 3 \times 3$ cubic array is defined whose center texel has the value $(0, 0, 0)$. As seen in Figure 4.1, the value of the each texel is set as the coordinates it has in the texture.

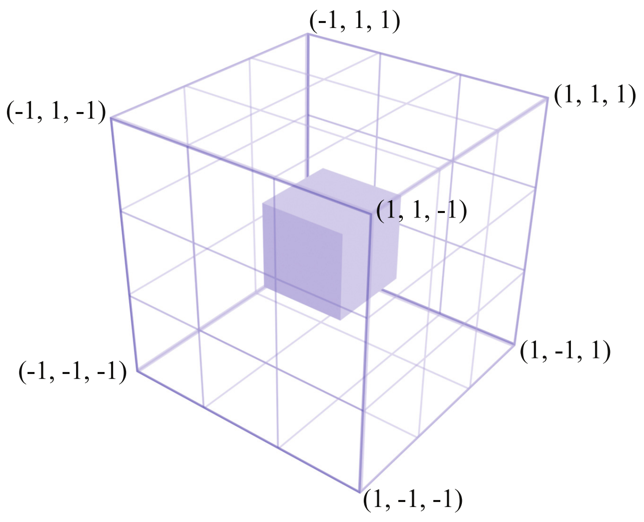


Figure 4.1: Interpolation cube. The center texel has the $(0, 0, 0)$ value.

Using the hardware interpolator, a uniform sampling can be done very fast using this cubic texture. Additionally, an approximation of a normal distribution can also be obtained using the inverse of its Cumulative Distribution Function (CDF). Setting in each texel the value of the CDF evaluated at its location, an approximation of the curve defined by that function can be interpolated. Since it is approximated linearly, its accuracy depends on the number of texels used in each dimension, who are acting as the control points of the interpolated curve. Figure 4.2 shows the approximated CDF for a single dimension.

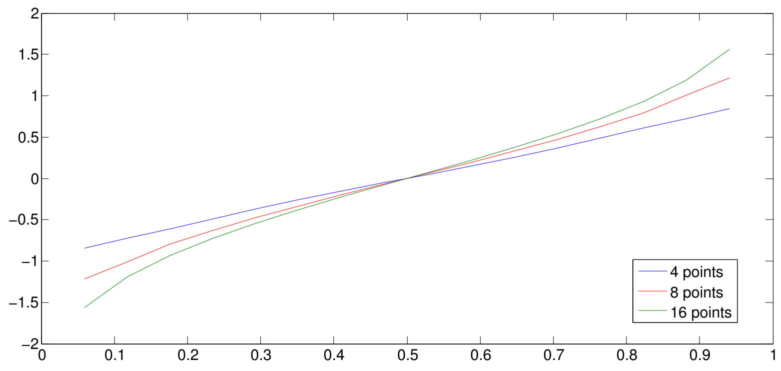


Figure 4.2: Interpolated Gaussian CDF using different number of control points.

The interpolated value must be properly scaled using the covariance matrix computed in Equation 3.28. The resulting hypotheses are stored in the global memory, as in the case of the pose samples.

For every point $\vec{z}_j$ optimized it is also necessary to transfer its corresponding 2D features in the past keyframes $\left\{\vec{y}_j^k, \vec{y}_j^l, \ldots, \vec{y}_j^m\right\}$. This set is built in the host and transferred to the constant memory of the device each time a point is adjusted. The pose information for the corresponding frames is also stored in the constant memory as a history of previous poses. This set is updated each frame adding the estimated pose using a device-to-device transfer.

Summarizing, Figure 4.3 shows the memory layout including the data transfers done by each kernel during the execution of the algorithm. The arrows indicate the direction of the transfers.

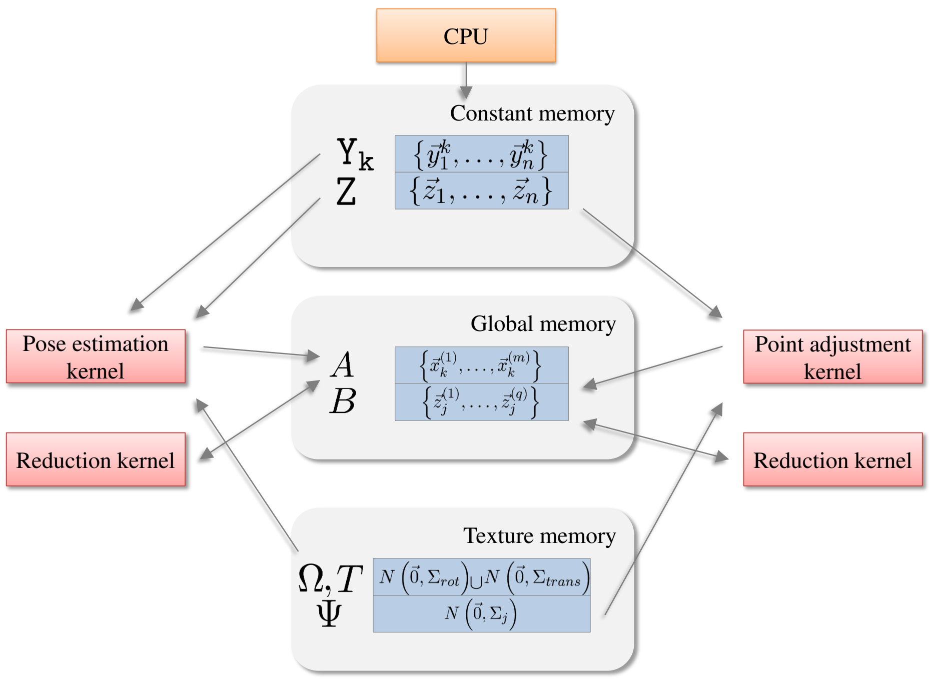

CPU
Constant memory
Υ_k
Z
$\{\vec{y}_1^k, \ldots, \vec{y}_n^k\}$
$\{\vec{z}_1, \ldots, \vec{z}_n\}$
Global memory
A
B
$\{\vec{x}_k^{(1)}, \ldots, \vec{x}_k^{(m)}\}$
$\{\vec{z}_j^{(1)}, \ldots, \vec{z}_j^{(q)}\}$
Texture memory
Ω, T
Ψ
$N(\vec{0}, \Sigma_{rot}) \cup N(\vec{0}, \Sigma_{trans})$
$N(\vec{0}, \Sigma_j)$
Pose estimation kernel
Reduction kernel
Point adjustment kernel
Reduction kernel

Figure 4.3: Memory layout and involved data transfers.

4.3 Pose Estimation

The pose is estimated each frame using a single computing kernel. It has two parts: sampling and weighting. Both parts are performed by the parallel device. In this way, the data needed are the 3D structure and its projections in the current frame.

The problem has been partitioned so that each computing thread samples and weights a single hypothesis. Therefore, there is no need of communications between threads, since each hypothesis is processed regardless of the others. As said in Section 4.2, in each frame k, the CPU only transfers the coordinates of the tracked feature points Υ_k. The 3D reconstruction is not transferred since it is permanently stored in the constant memory of the CUDA device. Algorithm 4.2 shows a resume of the computing kernel.

Although there is a conditional statement in the kernel, it does not suppose a performance drop, since the condition is evaluated equal in all the threads and thus without producing divergences in the program flow.

Algorithm 4.2 Overview of the CUestimatePose() kernel.

```
Require: $randomR$ → Array with random rotations
Require: $randomT$ → Array with random translations
  $i \leftarrow getThreadIndex()$
  $\vec{x}_k^{(i)}.R \leftarrow norm(randomR[i] + \vec{x}_{k-1}.R)$
  $\vec{x}_k^{(i)}.T \leftarrow \Sigma_{trans} * randomT[i] + \vec{x}_{k-1}.T$
  $w_k^{(i)} \leftarrow 0$
  for all $\vec{z}_j$ in Z do
    if isMarkedToTrack($z_j$) then
      $\vec{p}_j \leftarrow \Pi(\vec{x}_k^{(i)}.R * \vec{z}_j + \vec{x}_k^{(i)}.T)$
      $w_k^{(i)} \leftarrow w_k^{(i)} + norm(\vec{p}_j - \vec{y}_j^k)$
    end if
  end for
  $w_k^{(i)} \leftarrow \exp\left(-w_k^{(i)}\right)$
```

After the execution of the kernel, a list of weighted samples is obtained. The best sample is chosen as the current camera pose. Looking for the best sample implies looping through the list, comparing the weights of all samples. This is not the best situation for a GPU because this search requires comparisons that cannot be done directly in parallel. Since the system typically works with a large amount of samples, transferring the whole list to the main memory of the CPU becomes prohibitive. Because of this, the search is also done in the GPU using a parallel reduction.

A reduction is characterized by an input vector and an output element that is computed recursively as function of the input elements. In each iteration, the size of the input vector is reduced so that in the last iteration the resulting element is the solution sought. In this case, the function is a comparison between two elements that can be done in parallel. In this thesis the implementation proposed in (Harris, 2008) has been used, that uses a more sophisticated approach in order to get the peak performance of the CUDA device taking into account some architectural issues. Finally, the resulting pose is transferred to the CPU and moved to the pose history that is kept in the memory of the GPU.

Figure 4.4 shows the data flow in the execution of the pose estimation kernel. As seen, two floating point numbers are transferred for each feature before starting the execution and one pose vector after finishing. All these memory transfers can be done asynchronously, allowing the CPU to work on other tasks.

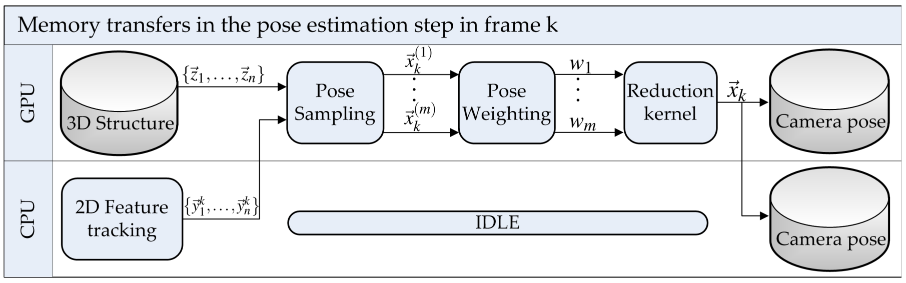


Figure 4.4: Execution of the pose estimation kernel.

4.4 Scene Reconstruction

The reconstruction is adjusted if the last tracked frame is marked as a keyframe. Only points having their projection error greater than a threshold are subject to be adjusted (typically 2-3 pixels). Since the resolution strategy is similar to the one used in the tracking kernel, the problem is partitioned in a similar manner. There is only one computing kernel, and each thread computes a single hypothesis about the position of a point.

Not all keyframes are used in the minimization. In order to cover most parts of the sequence and to avoid falling into local minima because of using only the last frames, the chosen keyframes are scattered along the time line. A Fibonnaci sequence has been used to generate the indices of the keyframes starting from $currentFrame - Fib\,(2)$. In this way, a lot of recent frames are used in the adjustment but some old frames are used too, leading to a better global behavior of the minimization.

For each point, the CPU must transfer to the GPU the indices of the frames to be used in the adjustment and the 2D features in those frames. As noted above, the poses corresponding to the frames are already stored in the constant memory of the GPU. Algorithm 4.3 shows the summary of the adjustment kernel.

Like in the pose estimation kernel, the best sample is chosen using a reduction algorithm and then transferred to the host. If the posterior projection error is less than the prior, the point is updated in the map. Figure 4.5 shows the overview of the kernel execution for each 3D point being adjusted.

Algorithm 4.3 Overview of the CUadjustPoint() kernel.

```
Require: $interpTex$ → Interpolation texture
Require: $indicesToAdjust$ → Indices of the frames to be used
  $i \leftarrow getThreadIndex()$
  $\vec{z}_j^{(i)} \leftarrow \vec{z}_j^{(-)} + \Sigma_{\vec{z}_j} * interpTex[i]$
  $w_j^{(i)} \leftarrow 0$
  for all $k$ in $indicesToAdjust$ do
    if isVisible($\vec{z}_j^{(-)}$) in $k$ then
      $\vec{p}_j \leftarrow \Pi(\vec{x}_k.R * \vec{z}_j^{(i)} +, \vec{x}_k.T)$
      $w_j^{(i)} \leftarrow w_j^{(i)} + norm(\vec{p}_j - \vec{y}_j^k)$
    end if
  end for
  $w_j^{(i)} \leftarrow \exp\left(-w_j^{(i)}\right)$
```

The sample mean and covariance are also calculated in the device using a parallel reduction. Two reduction passes are needed, since the mean is necessary in order to compute the covariance.

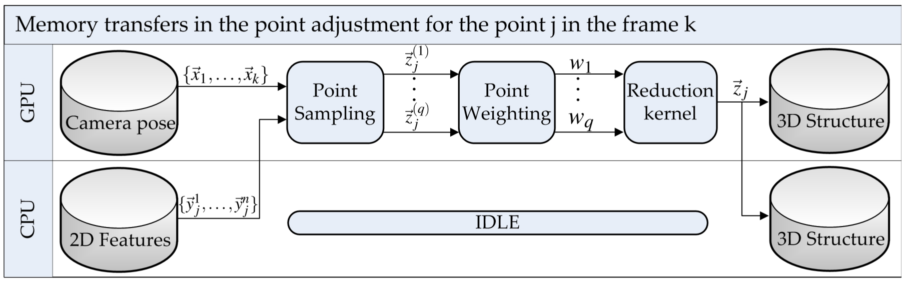


Figure 4.5: Execution of the structure adjustment kernel.

4.5 Experimental Results

This section details the performance tests done using the proposed implementation. The hardware used for the experiments is an Intel Core 2 duo at 3.0GHz with 4GB of RAM memory and a nVidia GeForce GTX260 whose capabilities are shown in Table 4.1.

Table 4.1: Capabilities of the used CUDA device.

Processor	**Number of cores**	192
	Clock rate	1.15GHz
Memory	**Global memory**	896MB
	Shared memory per block	64KB
	Constant memory	64KB
CUDA	**CUDA capabilities**	1.3
	Maximum number of threads per block	512
	Maximum dimension of a block	$512\times512\times64$
	Maximum dimension of a grid	$65535\times65535\times1$

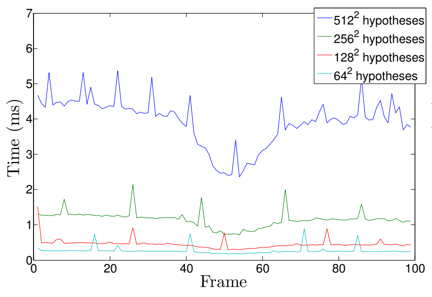
The experiments detailed in next sections have been divided in three groups corresponding to the motion estimation performance, the structure adjustment performance and finally the full SLAM system performance.

4.5.1 Pose Estimation Performance

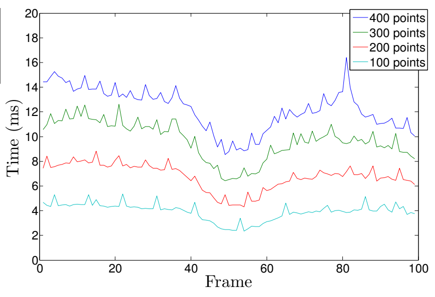
For the pose estimation experiments the synthetic sequence described in Table 3.1 has been used. The time needed by the kernel is only conditioned by the number of points used and by the number of hypotheses evaluated in each frame. Therefore, two sets of experiments have been done in order to measure the influence of these parameters in the estimation time.

Starting with the number of hypotheses, the total time needed is shown in Figure 4.6a. The number of points used has been fixed to 100 for each test. This time includes the computation itself and the data transfers needed to send the input data to the device and to read the results back. As seen, the estimation can be easily done in real time, needing an average of 4 ms for each frame using 512^2 hypotheses and, as expected, the time increases linearly with the number of samples taken. In addition, it should be noted that this operation is done asynchronously, so the CPU remains idle during this time.

In the same way, Figure 4.6b shows the time spent varying the number of points used, using 512^2 hypotheses. Like in the previous experiment, the time increases linearly with the number of points. This fact could be used to adapt dynamically the number of points used in function of the required computation time.



(a) Varying the number of hypotheses.



(b) Varying the number of points.

Figure 4.6: Pose estimation time.

4.5.2 Scene Reconstruction Performance

The performance of the reconstruction depends on the number of points adjusted in each frame, the total number of hypotheses and the amount of keyframes used in the optimization. The experiments described in this section measure the time needed by the structure adjustment kernel, taking into account the time spent transferring the data. Unless otherwise stated, for each adjustment a total of 13 keyframes are used and a total of 16^3 samples are evaluated.

Figure 4.7 shows the execution time needed to adjust the structure depending on the number of points optimized. The experiment has been repeated for different amount of samples.

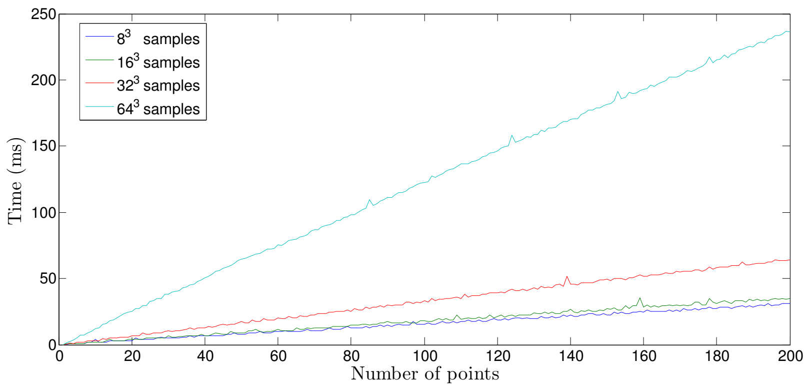


Figure 4.7: Reconstruction time varying the number of samples used.

The kernel is executed once for each point, thus the total time is increased linearly with the number of points. As shown, in each frame up to 200 points can be adjusted in real time using the default 16^3 number of samples and 13 frames.

Looking to Figure 4.8, it can be seen that the number of frames used has very little influence in the total time. From Algorithm 4.3 follows that the number of matrix operations is directly determined by the amount of keyframes used, as it is the control variable of the loop that computes the likelihood of the sample. This fact means that the limiting factor of the algorithm is the bandwidth of the system, rather than its arithmetical power. However, due to memory limitations, the maximum number of frames that can be stored in the device used in the experiments is 13.

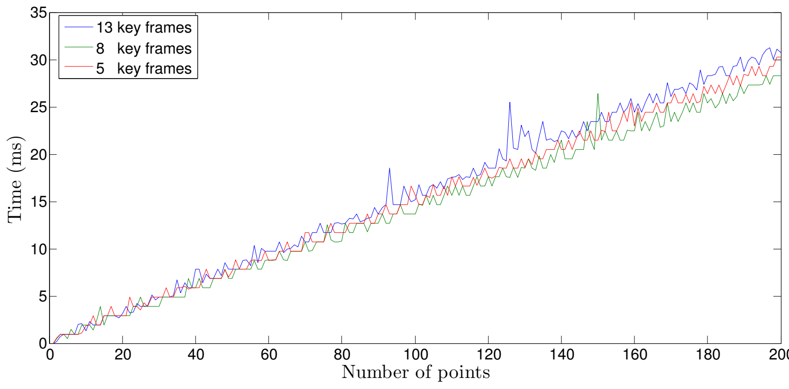


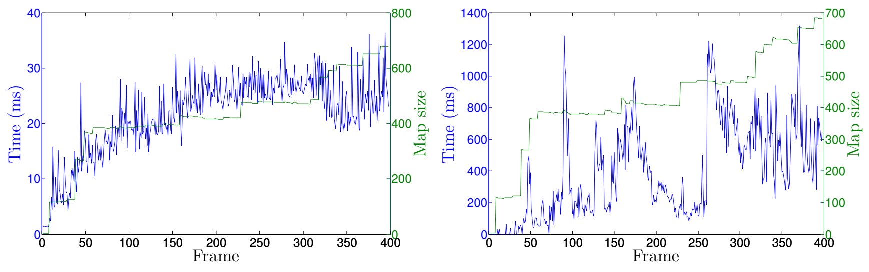
Figure 4.8: Reconstruction time varying the number of keyframes used.

4.5.3 SLAM Performance

The performance of complete SLAM system has been measured using the indoor sequence shown in Figure 3.22. Figure 4.9a shows the total time used in each frame including the pose estimation, the structure adjustment and the point remapping. The size of the 3D structure map is shown in the plot as well. In each frame, 512^2 hypotheses about the camera location are sampled and weighted. For each structure point a total of 13 keyframes are used in order to adjust its position.

As shown, the algorithm needs 25 ms per frame in average, meaning a frame rate of about 40 fps. Given that standard cameras work at 15-30 fps, the achieved frame rate is enough for real time monocular SLAM applications. Comparisons with a CPU implementation of this algorithm are not provided, since it is very slow and unusable for real time applications.

Figure 4.9b shows the time needed by BA to process the same video using the same parameters. As seen, it is not usable in real time tracking with the number of frames and features provided. Although a better parameterization could improve these times, the difference is very large. Abrupt changes are because the LM minimization can detect when the solution is near to the local minima and it finishes the process. It is a desirable feature, however the adjustment method proposed in this thesis cannot detect this situation because of its parallel nature.



(a) Total time on a real sequence using GPU. (b) Total time using Bundle Adjustment.

Figure 4.9: Execution times.

Finally, Figure 4.10 compares the total time needed to execute the algorithm in the outdoor sequence shown Figure 3.23, using different GPUs. For readability, the frames have been reordered in the plot depending on the execution time.

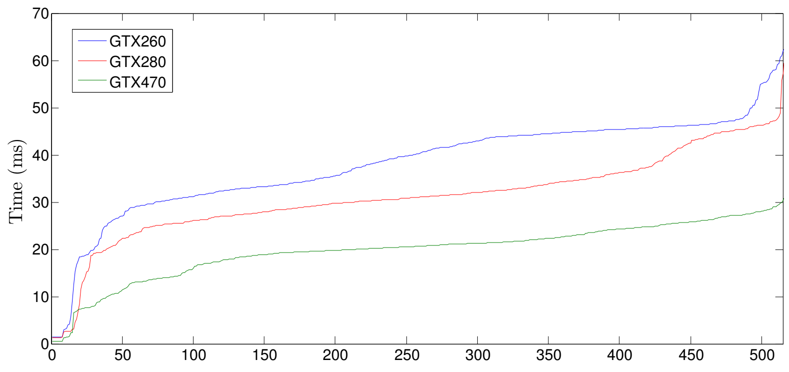


Figure 4.10: Comparison between different GPUs.

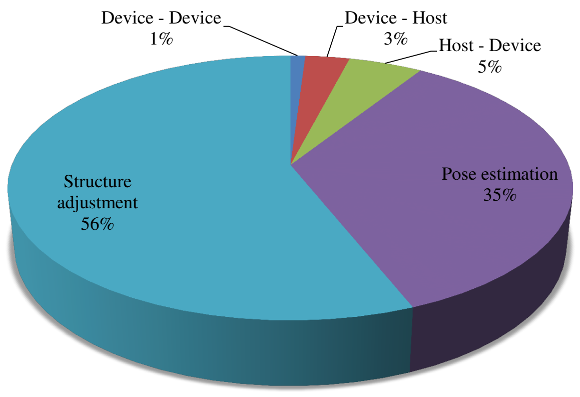


Figure 4.11: Percentage of GPU usage.

In this scenario, the average time is greater than in the previous sequence. The reason is that more points per frame are adjusted (a mean of 120 per frame), probably caused by the homogeneity of the road and garden areas. These zones introduce noise in the feature tracking process, and thus in the 3D reconstruction.

Summarizing, Figure 4.11 shows the percentage of GPU time used by each part. The plot includes the memory transfers done in all the three possible directions, i.e., host - device, device - host and device - device. As shown, the most time consuming part corresponds to the point adjustment. Some bottlenecks like memory read/write operations inside the kernels are not detailed, but they are included inside the kernel execution parts.

4.6 Discussion

In this chapter, the details of the CUDA implementation of the method are given. It can easily work in real time despite the limitations of the current hardware. In the case of the pose estimation, the speed is clearly limited by the computing power of the device, since the number of samples that need to be evaluated is very high. On the other hand, the structure estimation is limited by the bandwidth of the device. It is mainly because a lot of data transfers are needed for each point because the GPU cannot handle the data structures that a good implementation needs.

The computational power limitation could be currently solved using a system with multiple CUDA devices, allowing to handle a bigger reconstruction or more samples per estimation. These kind of systems are not very common at user level, however, other applications like large scale reconstructions are already using computer clusters that could take advantage of this implementation.

There are also some functionality aspects, as the lack of random number generating functions or the size of the shared and the constant memory spaces, that are limiting the method. However, it is a matter of time to overcome these limitations. In fact, hardware manufactures have already announced the random number functionality in their future products.

The hardware evolves very rapidly, and this is still a very new technology. In this way, the algorithm presented here has been designed to comply all the requirements needed to grow together with this technology.

Part III

Conclusions

Chapter 5

Conclusions and Future Work

Better that I devote myself to study the other great mystery of the universe...
DR. EMMET BROWN

5.1 Conclusions

This thesis presents a new approach to solve the SLAM problem in real time. Rather than adapt a traditional method, a fully parallelizable method based on Monte Carlo simulations has been proposed for both camera tracking and scene reconstruction. These types of methods have a prohibitive computational cost. However, thanks to the processing capabilities of GPU devices, a real time implementation has also been proposed.

The SLAM problem has been addressed using a bottom-up approach. In this context, all the parts of the SLAM pipeline have been covered through the document, including the 2D feature tracking, camera motion estimation, scene reconstruction and point remapping. The main contributions and conclusions obtained for each of these parts are:

1. **Feature point tracker:** The proposed 2D feature tracker can detect and track salient points in the images combining some state of the art techniques. More specifically, the Shi and Tomasi detector has been used combined with the optical flow of Lucas and Kanade. It has been demonstrated that the robustness against outliers can be greatly improved using a linear Kalman Filter in order to detect unpredictable motion

patterns. However, there are still some issues induced by false corners that cannot be detected at this level.

In addition, for each feature point a simplified version of the SIFT descriptor is computed. Its simplicity allows an efficient implementation with a negligible computational cost making it usable for real time SLAM applications, but loosing some features that standard SIFT descriptors have. Moreover, a lot of resources can be dedicated to calculate them, since the CPU time is not wasted in the pose and structure estimation tasks.

2. **Pose estimation:** The proposed camera tracker can estimate the 6 degrees of freedom of the camera using a Monte Carlo sampling method. The main contribution in this part has been a fully parallelizable sampling and weighting method that, as demonstrated, can run very fast in the graphics hardware available in almost any desktop computer. In this part, the strategy used to predict the uncertainty region in the sampling step has been crucial in order to reduce the number of samples needed by the filter, and thus increasing its performance. Moreover, its stochastic nature is very robust against outliers and other measuring errors, thus being a good alternative to classic trackers. It also has been demonstrated that the sampling function is flexible enough to deal with erratic motions, like those that appear in handheld cameras.

3. **Structure estimation:** The structure reconstruction is approached in a similar way. In contrast to Bundle Adjustment method, it is separately estimated from the pose, but it has been demonstrated in the experiments that the solution converges to the correct result as the video sequence advances.

   Starting from the initial solution obtained by means of linear triangulation, structure points are adjusted using a Monte Carlo sampling as well. The sampling and weighting methods have been specifically designed to obtain the peak performance of the graphics hardware being able to run in real time together with the camera tracker. In this part, the uncertainty region of the sampling function has also been obtained using the linearization of a hypothetical dynamic model.

4. **Point remapping:** Finally, a simple point remapping method is also proposed, so that lost points can be relocated in the images using the SIFT descriptors computed by the feature tracker. This remapping allows to recognize already visited places reducing the drift of the tracking each time

an old point is remapped. It has also been seen that the simplifications done in the descriptors can be assumed taking into account the spacial coherency that can be obtained from the SLAM system.

Synthetic and real data have been used in order to validate the proposed SLAM solution in accuracy and performance terms. As shown in the experiments, the results are very similar to the classical structure from motion approach using Bundle Adjustment in conjunction with nonlinear minimization methods. However, in performance terms, a mean of 20x speed up is achieved, with the advantage of leaving the CPU free to do other tasks.

In contrast, since the observation densities of both estimators follow a bottom-up approach, they suffer the classic drawback of this type of methods: they are very sensitive to quick camera motions that generate motion blur and make the measures given by the feature tracker incorrect. Taking into account that bottom-up approaches are driven by those measurements and future predictions depends on past predictions, these noisy images can lead to a catastrophic result in the output of the predictors. Normally, this situation can be detected and the tracking can be recovered using relocalization techniques (Williams et al., 2007). However, they can lead to incorrect mappings that degrade the 3D structure and to an unrecoverable error state.

Like other computer vision methods, it also depends on the quality of the camera used and on the lighting factors, but these problems cannot be easily avoided. However, all the tests with real videos have been done using a low cost webcam, and its minimum resolution, showing that the method can tolerate quite well cheap cameras and low resolutions.

Augmented reality needs a lot of computational resources to work. The trend today is to improve the hardware by means of parallelism rather than in arithmetic power. Just like happened years ago with 3D graphics, a good solution is to use a specialized massively parallel hardware. As this thesis demonstrates, in the field of augmented reality, random sampling algorithms are the best candidates for future methods because of their parallelism level, high accuracy and robustness against outliers.

In summary, and very important to note, this work presents a GPU computational framework that can be used to solve many computer vision problems. In this thesis it has been used to solve the camera tracking and 3D reconstruction problems, but it could be applied to almost any measurement driven minimization algorithm, getting a fast, robust and easy to implement method.

5.2 Future Work

The work presented in this document leaves several research lines opened for future investigations. They are enumerated according to the part to which they belong.

1. **Feature tracking:** As said, an important drawback is the use of a bottom-up approach. It could be solved using a top-down approach eliminating the feature tracking and matching the point correspondences using other criterion. The more natural solution would be using a complete version of the SIFT descriptor. However, nowadays its computational cost is unaffordable even for modern processors. Currently, there are some efforts to get fast implementations computing the descriptors using the graphics hardware, like (Changchang, 2007). Its speed is still bellow the requirements of a SLAM application, nevertheless in the future it could be a good improvement to this SLAM approach.

   Meanwhile, a hybrid approach could benefit from the speed of the bottom-up approach and the relocalization capabilities of a top-down implementation. This could be achieved using the bottom-up observation density only if the uncertainty of the pose estimation is bellow some threshold, switching to the top-down density when the uncertainty is high. In this way a good compromise between speed and flexibility could be obtained.

   The tracking itself could also be improved using a single Kalman filter containing all the points instead of having one estimation per point. This would give an overall overview of the motion of the features rather than several local predictions, and the information about the correlation that the covariance of the state would give could be used to improve the robustness of the tracker against false corners.

2. **Pose estimation:** The pose estimation could be improved both in the sampling and weighting aspects. Regarding to the sampling phase, its capacity to predict high probability areas is fundamental in order to reduce the number of hypotheses needed. Apart to the improvements that could be done in the estimation of the sampling covariance, it could also benefit from an annealed approximation, sequentially reducing the sampling space in a hierarchical way. This would probably reduce the number of samples needed in each anneal level, compared with the direct sampling approach.

However, it only would be useful in the case that the estimated uncertainty is bigger than the real.

The weighting function could also be improved using more sophisticated metrics, taking into account the covariances of the 3D points. In this way, the points whose position is better approximated would have more weight in the final likelihood, getting a more accurate result and adding robustness to the observation function.

Finally, as discussed in Section 3.8, it is very reasonable to reduce the number of samples in the states where the uncertainty is low.

3. **Structure estimation:** Regarding to the performance aspects of the filter, the structure estimation could also benefit from the improvements proposed to the pose estimation algorithm, including the annealing step, the top-down approach, the study of more robust metrics and the adaptative size of the sampling space.

   Moreover, because of the rigidity of solid objects, structure points are usually highly correlated. This means that the more probable position of a 3D point is influenced by their neighbors, because the distance between them tends to be conserved. This relationship could be used to drive the minimization and remapping steps, leading to faster convergences and more accurate results. Moreover, it would be possible to adjust a point even without having its corresponding measurements, as it would be pulled to the optimal position by the other points. The main drawback is that this "global" approach requires a very high dimensional space, increasing the total time, and probably loosing the real time performance, at least with the current hardware.

Finally, the augmented reality field still has some open research aspects in the line of realistic rendering. A good tracking is not sufficient to create a realistic feeling of coexistence between real and virtual objects. Topics like illumination and occlusions have to be also solved.

The scene illumination can be estimated using external reflective markers using the Image Based Lighting (IBL) technique (Debevec, 2002), however there is still a lot to do in markerless light extraction. Some interesting techniques are already being applied in order to extract an approximate illumination from single images (López-Moreno et al., 2009) in the image processing field. They can extract the position and the intensity of multiple light sources requiring only the presence of a convex object that the user has to select. Although the exact positions are not extracted, the authors claim that it is enough for rendering purposes. However, since in an augmented reality application the pose of the camera is known, the method could be extended taking this information into account, getting an even better approximation. Moreover, the manual object selection could be avoided taking into account that a 3D representation of the scene is also available.

The occlusions are normally solved using dense structure reconstruction techniques. However, usually those techniques are not suitable for real time augmented reality applications. Besides, a dense reconstruction combined with the information about the illumination could be used to project the shadows between real and virtual objects.

Part IV

Appendices

Appendix A

Projective Geometry

We are used to describe our world using the traditional Euclidean geometry. This geometry describes in a natural way some properties that can be measured in the space like parellelism, lengths and angles. These properties remain invariant after applying any Euclidean transformation, however, as can be seen in Figure A.1 the transformation that a camera applies to a 3D scene does not preserve these properties. Parallel lines are no longer parallel, the distances between objects cannot be measured and the angles are not preserved. This is because the perspective transformation that happens inside a camera belongs to the group of projective transformations. The projective geometry is a more general case of the Euclidean geometry that allows describing a larger class of transformations, such as perspective projection. Moreover, it allows describing special points, like points lying in the infinity, in the same way as ordinary points.

The projective space of dimension n, $\mathtt{P}^n$ is defined as the set of points $\vec{x} \in \mathtt{R}^{n+1} - \{\vec{0}_{n+1}\}$ where two points are said to be equivalent or similar if and only if $\vec{x}_1 \sim \vec{x}_2 \Leftrightarrow \vec{x}_1 = \lambda\vec{x}_2$. The following sections describe some aspects of the projective plane $\mathtt{P}^2$ and the projective space $\mathtt{P}^3$. It is a very brief description, but enough to understand this thesis. For a deeper discussion oriented to computer vision applications, the interested reader can refer to (Hartley and Zisserman, 2004).

A synthesis of this appendix has been presented in:

Sánchez, J. R. and Borro, D. "Automatic Affine Structure Recovery Using RANSAC". In Congreso Español de Informática Gráfica (CEIG'10), pp. 155–164. Valencia, Spain. 2010.

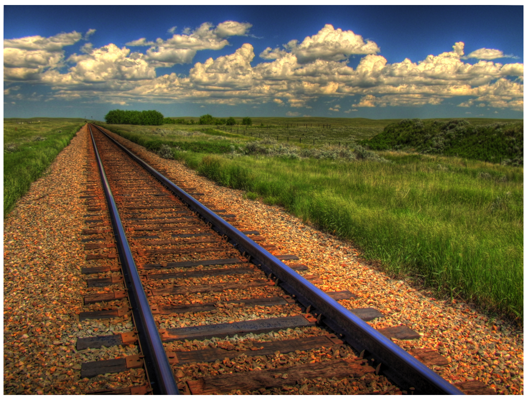


Figure A.1: Perspective distortion caused by a camera. Parallel lines meet at a common point.[1]

A.1 The Projective Plane

In projective geometry, a point on a plane is represented by an homogeneous 3-vector

$$\vec{m} = (x, y, w)^\top \tag{A.1}$$

such that at least one entry is not zero. Usually, if $w = 0$ the point is said to be at infinity.

A point transformation in the projective plane is represented by a non singular 3×3 matrix with eight degrees of freedom: $\vec{m}' = \mathbf{H}\vec{m}$ called homography. These transformations are also known as collineations, since they preserve point collinearity.

Lines in the projective plane are also defined by 3-vectors and have the form $\vec{l} = (a, b, c)^\top$. A point $\vec{m}$ lies on the line $\vec{l}$ if and only if:

$$\vec{m}^\top \vec{l} = 0. \tag{A.2}$$

A line can be defined with two points as:

$$\vec{l} = \vec{m}_1 \times \vec{m}_2. \tag{A.3}$$

[1]Photo by Evan Leeson `http://www.flickr.com/photos/ecstaticist/`

In the same manner, the intersection point defined by two lines is represented by $\vec{m} = \vec{l}_1 \times \vec{l}_2$. This symmetry is applicable to any statement concerning to points and lines and it is called the duality principle of the projective plane.

A.2 The Projective Space

Similarly, points in projective space are represented by homogeneous 4-vectors:

$$\vec{M} = (X, Y, Z, W)^\top . \tag{A.4}$$

A point transformation in projective space is represented by a non singular 4×4 matrix: $\vec{X}' = \mathbf{H}\vec{X}$.

A plane in the projective space is also represented by a 4-vector $\vec{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)^\top$. A point $\vec{X}$ lies on the plane $\vec{\pi}$ if:

$$\vec{\pi}^\top \vec{X} = 0. \tag{A.5}$$

Equation A.5 is unaffected by multiplication, so only the ratios $\{\pi_1 : \pi_2 : \pi_3 : \pi_4\}$ are significant. This means that a plane in projective space has 3 degrees of freedom. From this fact it follows that a plane can be defined by three non-collinear points $\vec{X}_1$, $\vec{X}_2$ and $\vec{X}_3$:

$$\begin{bmatrix} \vec{X}_1^\top \\ \vec{X}_2^\top \\ \vec{X}_3^\top \end{bmatrix} \vec{\pi} = \vec{0}. \tag{A.6}$$

The duality principle is also applicable in projective space. In this case, the dual of the point is the plane. In the same way that a plane is defined by three points, a point can be defined by three planes:

$$\begin{bmatrix} \vec{\pi}_1^\top \\ \vec{\pi}_2^\top \\ \vec{\pi}_3^\top \end{bmatrix} \vec{X} = \vec{0}. \tag{A.7}$$

Lines in projective space can be defined by the two points or by the intersection of two planes. They have various possible representations. The most common is the null-space and span representation where a line is represented by a 2×4 matrix with the form

$$\mathbf{W} = \begin{bmatrix} \vec{A}^\top \\ \vec{B}^\top \end{bmatrix} \tag{A.8}$$

where the span $\lambda\vec{A} + \mu\vec{B}$ is the pencil of points lying on the line. In the same way, a line can be represented as the intersection of two planes using a matrix composed of two planes:

$$\mathbf{W}^* = \begin{bmatrix} \vec{P}^\top \\ \vec{Q}^\top \end{bmatrix}. \tag{A.9}$$

The span $\lambda'\vec{P} + \mu'\vec{Q}$ is the pencil of planes with the line as axis.

A.3 The Stratification of 3D Geometry

We are used to perceive the world in a Euclidean way. Having a Euclidean representation of an object, it can be measured from it almost everything (lengths, angles, volumes, etc.). However, it is sometimes impossible to obtain a Euclidean representation of an object, so we must handle a simplified version of it. This is the case in most of computer vision applications, since the camera is a projective machine. That is why the concept of the stratification of the 3D geometry (Faugeras, 1995) is needed.

In 3D geometry, there are four different layers: projective, affine, metric and Euclidean. Each layer is a subset of the previous one and it has associated a group of transformations that leave some properties of the geometrical entities invariant.

The Euclidean layer is the most restrictive one. A Euclidean transformation is represented by a 6 degrees of freedom 4×4 matrix, 3 for rotation and 3 for translation:

$$\mathbf{T}_e = \begin{bmatrix} \mathbf{R} & \vec{t} \\ \vec{0}_3^\top & 1 \end{bmatrix} \tag{A.10}$$

where $\vec{t} = (t_x, t_y, t_z)^\top$ is a 3D translation vector and $\mathbf{R}$ is a 3D orthogonal rotation matrix. This group of transformations leave the volume of objects invariant.

The next layer is the metric or similarity one. Similarity transformations are 7 degrees of freedom 4×4 matrices and are represented by:

$$\mathbf{T}_m = \begin{bmatrix} s\mathbf{R} & \vec{t} \\ \vec{0}_3^\top & 1 \end{bmatrix} \tag{A.11}$$

where s is a scalar representing an arbitrary scale factor. This transformation leaves invariant relative distances, angles and a special entity called the absolute conic.

The next layer is the affine one. Affine transformations are 12 degrees of freedom 4×4 matrices represented by:

$$\mathbf{T}_a = \begin{bmatrix} \mathbf{A} & \vec{t} \\ \vec{0}_3^\top & 1 \end{bmatrix} \tag{A.12}$$

where $\mathbf{A}$ is an invertible 3×3 matrix. Transformations belonging to this layer leave invariant the parallelism of planes, volume ratios, centroids and a special entity called the plane at infinity.

The less restrictive layer is the projective one. Projective transformations are 15 degrees of freedom 4×4 matrices represented by:

$$\mathbf{T}_p = \begin{bmatrix} \mathbf{A} & \vec{t} \\ \vec{v}^\top & v \end{bmatrix} \tag{A.13}$$

where $\vec{v}$ is a 3-vector and v is a scalar. As in the case of plane homographies, this transformation is a collinearity that leaves invariant intersections and surfaces tangencies in contact.

Figure A.2 shows the effect of applying these transformations in a cube represented in Euclidean space.

A.4 Stratified 3D Reconstruction

Having only two views of a scene captured by an uncalibrated camera, it is only possible to retrieve a 3D reconstruction up to an arbitrary projective transformation. This means that if we want the real Euclidean structure, we have to apply a projective transformation to the retrieved reconstruction. In the example

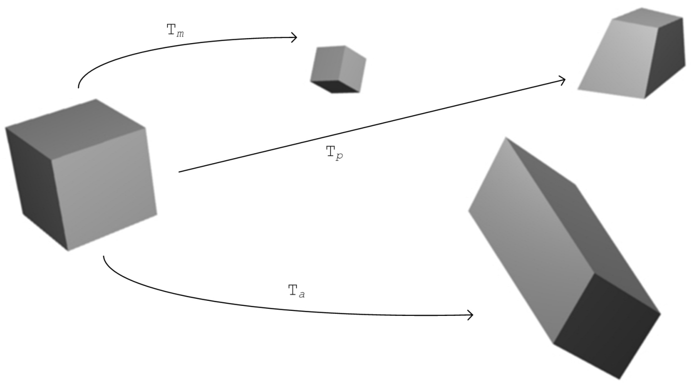


Figure A.2: Metric, affine and projective transforms applied on an object in Euclidean space.

shown in Figure A.2 the transformation required to upgrade the projective cube to Euclidean geometry is $\mathbf{T}_p^{-1}$.

This may not be sufficient for some tasks, like augmented reality, but it can be the starting point of a stratified reconstruction, i.e. add richer information to the reconstruction allowing it to be upgraded to the next geometric layer. First, the projective reconstruction is upgraded to the affine layer, then to the metric and finally to the Euclidean.

A.4.1 Projective Reconstruction

The projective reconstruction of the scene can be recovered using the Fundamental matrix $\mathbf{F}$. It is an 8 degrees of freedom 3×3 matrix that relates the projection of a 3D point in two different views.

From the Fundamental matrix a projective 3D transformation relating two views can be obtained. With this information the 3D structure can be recovered via linear triangulation methods. Appendix B explains in detail these topics.

A.4.2 From Projective to Affine

Upgrading from projective to affine supposes to find the plane at infinity. This is the plane where parallel lines meet. In affine space, this plane is truly located at infinity and has coordinates $\vec{\pi}_\infty = (0, 0, 0, 1)^\top$, so points lying on it have $W = 0$

(they are at infinity). However, this does not happen in projective space because parallelism is not an invariant property of it. Lines that are parallel in affine space meet in a normal plane in projective space (not at infinity). It is said that the plane at infinity is not on its canonical position. Finding this plane allows us to obtain a transformation that moves it to its canonical position, making parallel lines meet again at infinity. The projective transformation that allows the upgrade from projective to affine space is defined by:

$$\mathbf{T}_{pa} = \begin{bmatrix} \mathbf{I}_3 & \vec{0} \\ \vec{\pi}_\infty^\top & \end{bmatrix} \Leftrightarrow \mathbf{T}_{pa}^{-\top} \vec{\pi}_\infty = (0,0,0,1)^\top . \tag{A.14}$$

As seen in Figure A.3, three sets of parallel lines are enough to locate the plane at infinity using Equation A.6.

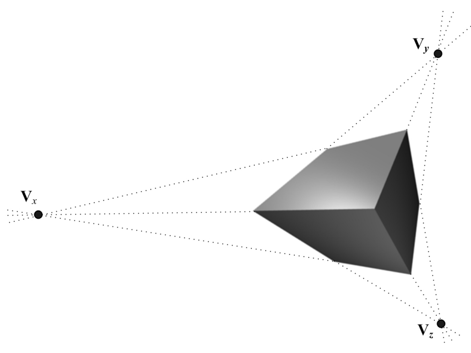


Figure A.3: The plane at infinity is the plane defined by $\vec{V}_x$, $\vec{V}_y$ and $\vec{V}_z$. These points are the *vanishing points* of the cube.

A.4.3 From Affine to Metric

The metric level is the richest one that can be obtained from images. Upgrading from affine to metric supposes finding the absolute conic $\mathbf{\Omega}_\infty$. This entity is a planar conic that lies on the plane at infinity. The key in this step is to find an affine transformation that maps the identified conic to its canonical position in Euclidean space ($X^2 + Y^2 + Z^2 = 0$ and $W = 0$).

A.4.4 From Metric to Euclidean

This step can be performed only if real lengths of the reconstructed object are known. This allows to obtain a scale factor s that upgrades metric reconstruction to Euclidean space.

A.5 A Proposal for Affine Space Upgrading

This section proposes a new method to upgrade a projective reconstruction to affine space. It is based on locating the plane at infinity assuming that there exist parallel lines in the scene. This is an acceptable assumption in almost any man-made scene.

The first step is to obtain a projective reconstruction of the scene and then locate the maximum number of lines on it. The line search can be done either manually or automatically. If the projective reconstruction has been retrieved from an image pair (using the Fundamental matrix and feature point matching) the lines can be detected in images locating the feature points that define them and then matching these points with the reconstructed vertices. Figure A.4 shows this process. In this way, the result would be a 3D structure containing edges and not only points. These 3D edges are the only information needed to obtain the plane at infinity. There are several edge detectors that can be used to carry out this detection, like Canny operator (Canny, 1986) or the Hough transform (Duda and Hart, 1972).

Once lines are detected, the next step is to compute the intersection of all line pairs. Due to numerical stability reasons, it may be a good idea to normalize the 3D points so that the centroid is at the origin and the RMS distance of 3D points to the origin is $\sqrt{3}$.

Some of these intersection points will be vanishing points and others will not. The idea is to find vanishing points and use them to update the projective structure to affine space. Since vanishing points are located in the plane at infinity, there will be a plane defined by three of the computed intersection points that will contain all the vanishing points. This plane can be identified as the plane to which more intersection points belong, assuming that intersection points not corresponding to parallel lines (non vanishing points) are randomly scattered across the space.

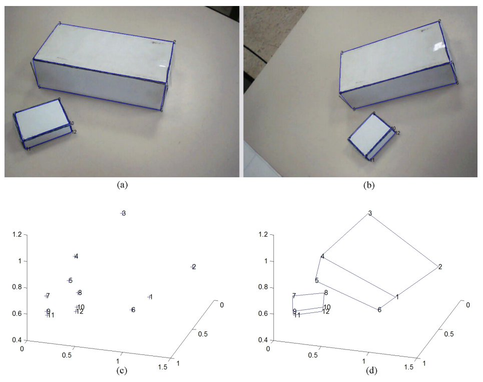


Figure A.4: Edge matching process. (a) and (b) feature matching between images. (c) Projective reconstruction. (d) Projective reconstruction after edge detection and matching.

A.5.1 Data Normalization

The transformation matrix related to this normalization is:

$$\mathbf{H}_N = \begin{bmatrix} \frac{\sqrt{3}}{rms} & 0 & 0 & \frac{-\sqrt{3}*c_x}{rms} \\ 0 & \frac{\sqrt{3}}{rms} & 0 & \frac{-\sqrt{3}*c_y}{rms} \\ 0 & 0 & \frac{\sqrt{3}}{rms} & \frac{-\sqrt{3}*c_z}{rms} \end{bmatrix} \tag{A.15}$$

where RMS is the root mean squared distance of 3D points to the origin and $\vec{c}$ is the centroid.

This normalization can be inverted when the plane at infinity is found and before doing the affine rectification. If $\vec{\pi}_\infty$ is the plane at infinity of the unnormalized structure, then from Equation A.5 follows:

$$\vec{\pi}_\infty^\top \vec{X} = \vec{\pi}_\infty^\top \mathbf{H}_N^{-1} \mathbf{H}_N \vec{X} = \mathbf{H}_N^{-\top} \vec{\pi}_\infty \mathbf{H}_N \vec{X} \tag{A.16}$$

so the plane at infinity recovered from the normalized structure is $\pi_{\infty N} = \mathbf{H}_N^{-\top} \vec{\pi}_\infty$. The original plane can be retrieved from the normalized one using $\pi_\infty = \mathbf{H}_N^\top \pi_{\infty N}$.

A.5.2 Computing Intersection Points

The intersection of two lines defined respectively by the pairs of points $\left(\vec{A}, \vec{B}\right)$ and $\left(\vec{C}, \vec{D}\right)$ can be computed equaling the span of these points:

$$\lambda_1 \vec{A} + \mu_1 \vec{B} = \lambda_2 \vec{C} + \mu_2 \vec{D}. \tag{A.17}$$

This leads to the next linear system of equations:

$$\begin{bmatrix} A_1 & B_1 & -C_1 & -D_1 \\ A_2 & B_2 & -C_2 & -D_2 \\ A_3 & B_3 & -C_3 & -D_3 \\ A_4 & B_4 & -C_4 & -D_4 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \mu_1 \\ \lambda_2 \\ \mu_2 \end{bmatrix} = \vec{0}. \tag{A.18}$$

The solution of this system can be computed as the singular vector corresponding to the smallest singular value of the coefficients matrix. This leads to two intersection points. Theoretically they must be equal, but can be different because lines might not intersect as shown in Figure A.5 due to noise or numerical stability reasons.

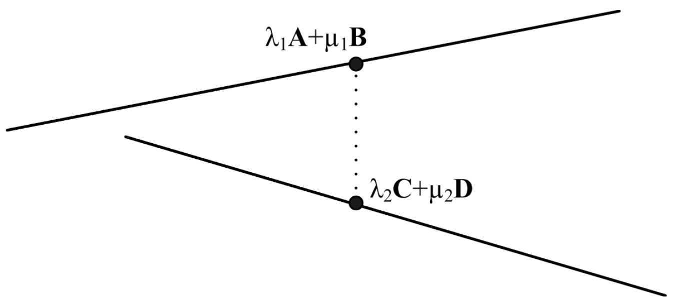


Figure A.5: The lines might not intersect.

A good choice is the middle point of the line joining these two points.

A.5.3 Plane Localization

The plane at infinity is located using RANSAC (Fischler and Bolles, 1981) over all the computed intersection points. RANSAC is an iterative algorithm that randomly takes groups of three points and computes the plane defined by them using Equation A.6. Then, the restriction described in Equation A.5 is tested on all intersection points and if the residual error is near to zero, the point is considered to lie on the plane. Points lying on the plane are called *inliers* and points not lying on it are called *outliers*. RANSAC takes as a solution the group that generates less outliers. The parameterization needed is the threshold value of Equation A.6 to consider that a point lies on the plane and the maximum number of iterations allowed to RANSAC.

Theoretically, all lines along the same direction intersect in the same vanishing point. However, due to noise in the points defining lines and numerical stability reasons, this may not be true. As seen in Figure A.6, small perturbations in the position of a vertex modifies considerably the intersection point. Furthermore, the error in the intersection is proportional to the distance from the computed intersection point to the corrupted vertex. If $\vec{V}$ is a point belonging to the line $\lambda\vec{A} + \mu\vec{B}$, then if $\vec{B}$ is corrupted, error in $\vec{V}$ will grow linearly with μ:

$$\lambda\vec{A} + \mu\left(\vec{B} + \vec{E_B}\right) = \vec{V} + \mu\vec{E_B}. \tag{A.19}$$

To solve this problem, intersection points close to each other are considered equal and their centroid is taken as the true intersection point, since the error in the centroid will be less than or equal to the maximum error. Due to Equation A.19, the distance used to compare points depends linearly on the norm of one of them.

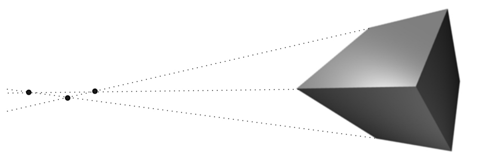


Figure A.6: Computing intersections with noise.

Since the intersection points that appear many times are suspected of being vanishing points, the original RANSAC has been modified to take this into account. Each intersection point has an associated weight w_i that conditions the probability to be chosen by RANSAC. This coefficient is selected according to the number of times the point appears when computing intersections. Thus, if there are n intersection points and $\vec{P}$ is the centroid of k nearby points, the weight associated to $\vec{P}$ would be $w_P = k/n$ being $\sum w_i = 1$. These modifications make RANSAC faster and more robust against noise in vertex position.

A.5.4 Convergence

In absence of noise, the minimum number of pairs of parallel lines required is four, where at least three of them have different directions. If only three pairs are supplied, the maximum number of inliers that would be detected when trying this group by RANSAC would be three, just like when trying another group of non parallel lines.

Appendix B

Epipolar Geometry

Epipolar geometry, also known as multiview geometry, describes the relation that exists between different views of a scene. In other words, the projection of a point in one image can be restricted if the projection of that point in other views is known.

Some early works in stereo vision, like (Marr and Poggio, 1979), established the relations between the projections of a point in a stereo system and its depth, where the views are related by a known translation. This work was extended by Longuet-Higgins (Longuet-Higgins, 1981) giving the relation that exists between two arbitrary views and an algorithm that computes the 3D reconstruction of the captured scene. This work is considered the basis of all the multiview geometry field.

The following sections describe the aspects of epipolar geometry needed to understand this thesis. Only the case of two views is analyzed but the interested reader can refer to (Hartley and Zisserman, 2004) for a broader review of these concepts extended to *n* views.

B.1 Two View Geometry

In a projective camera (like a pin-hole) 3D points that are collinear with respect to the optical center have the same projection. However, if these points are viewed from a different viewpoint, their projections change giving the impression that their relative position has changed. This effect is known as motion parallax.

The projections of these points in the second image are constrained in some way. If a plane that contains $\vec{z}_1$, $\vec{z}_2$ and the optical centers of the two cameras

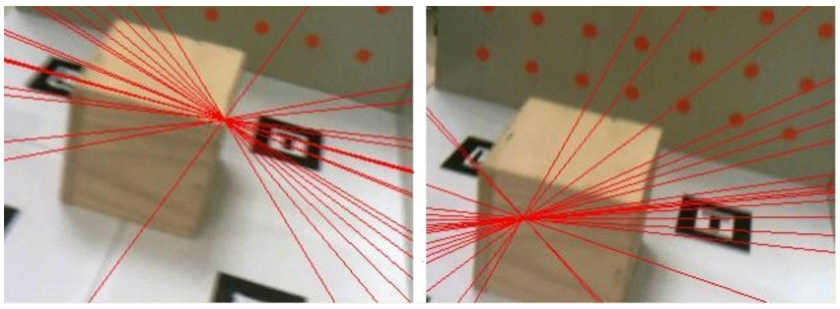


Figure B.1: Set of epipolar lines intersecting in the epipoles.

is defined, the projections of these points in the second image are constrained to lie in the line defined by the intersection of the plane with the image. In fact, every plane that contains both optical centers intersects with the images in two lines known as epipolar lines. These lines have a common point called epipole. Epipoles are the projections of the camera centers in the opposite image. Figure B.1 shows the epipolar lines and the epipoles of a pair of images.

This relation is encapsulated by a 3×3 matrix $\mathbf{F}$ called Fundamental matrix. For every pair of projections $\vec{y}^j$ and $\vec{y}^k$ of a point $\vec{z}$ the epipolar constraint is defined as:

$$\vec{y}^{k\top}\mathbf{F}\vec{y}^j = 0. \tag{B.1}$$

Since point $\vec{y}^k$ is restricted to lie in the epipolar line $\vec{l}^k$, from Equation A.2 and substituting in Equation B.1 the epipolar lines can be obtained as:

$$\begin{aligned} \vec{l}^k &= \mathbf{F}\vec{y}^j \\ \vec{l}^j &= \mathbf{F}^\top\vec{y}^k. \end{aligned} \tag{B.2}$$

B.2 The Fundamental Matrix

The Fundamental matrix is the algebraic representation of the epipolar geometry. It depends only on the position and orientation of the two cameras that define the two views and it can be derived using their projection matrices.

Suppose an arbitrary 3D point $\vec{z}$ whose projection in the first image is defined as $\vec{y^j} = \mathbf{P}_j\vec{z}$ where $\mathbf{P}_j$ is the projection matrix of the first camera. The line that goes through $\vec{z}$ and the camera center can be obtained backprojecting the point $\vec{y^j}$ as follows:

$$\vec{z}(\lambda) = \mathbf{P}_j^+\vec{y^j} + \lambda\vec{C_j} \tag{B.3}$$

where $\mathbf{P}_j^+$ is the Moore-Penrose pseudo inverse of $\mathbf{P}_j$ and $\vec{C_j}$ is the camera center of the first camera. Taking the point defined by $\lambda = 0$, its projection in the second image can be written as $\mathbf{P}_k\mathbf{P}_j^+\vec{y^j}$. In the same way, the epipole in the second image can be obtained projecting the camera center of the first image as

$$\vec{e_k} = \mathbf{P}_k\vec{C_j}. \tag{B.4}$$

Having these projections, the epipolar line can be recovered using Equation A.3:

$$\vec{l^k} = [\vec{e_k}]_\times\mathbf{P}_k\mathbf{P}_j^+\vec{y^j} \tag{B.5}$$

where $[\vec{e_k}]_\times$ is the 3×3 antisymmetric matrix representing the vectorial product with $\vec{e_k}$. Finally, using Equation B.2 the Fundamental matrix can be obtained as:

$$\mathbf{F} = [\vec{e_k}]_\times\mathbf{P}_k\mathbf{P}_j^+. \tag{B.6}$$

If the first camera is centered at the origin of the coordinate system, the projection matrices can be expanded as:

$$\begin{aligned} \mathbf{P}_j &= \mathbf{K}_j[\mathbf{I}_3|\vec{0}_3] \\ \mathbf{P}_k &= \mathbf{K}_k[\mathbf{R}|\vec{t}]. \end{aligned} \tag{B.7}$$

Since the epipole $\vec{e_k}$ is the projection of the first camera center in the second image, Equation B.4 can be rewritten as:

$$\vec{e_k} = \mathbf{K}_k[\mathbf{R}|\vec{t}]\vec{C_j} = \mathbf{K}_k[\mathbf{R}|\vec{t}]\begin{bmatrix}\vec{0}_3\\ 1\end{bmatrix} = \mathbf{K}_k\vec{t}. \tag{B.8}$$

Substituting in Equation B.6, the Fundamental matrix can be expressed in terms of the camera matrices:

$$\mathbf{F} = [\mathbf{K}_k\vec{t}]_\times \mathbf{K}_j[\mathbf{R}|\vec{t}] \begin{bmatrix} \mathbf{K}_j^{-1} \\ \vec{0}_3^\top \end{bmatrix} = [\mathbf{K}_k\vec{t}]_\times \mathbf{R}\mathbf{K}_j^{-1} = \mathbf{K}_k^{-\top}[\vec{t}]_\times \mathbf{R}\mathbf{K}_j^{-1}. \tag{B.9}$$

The Fundamental matrix has rank 2 and its overall scale is not significant, so it has eight degrees of freedom, i.e. $\lambda\mathbf{F}$ is projectively equivalent to $\mathbf{F}$ for any non zero scalar λ. From Equation B.4 it follows that if there is no displacement, i.e. the two camera centers are the same, the Fundamental matrix would be the zero matrix.

B.3 The Essential Matrix

The Essential matrix is a particular case of the Fundamental matrix when the calibration matrices of the cameras are known. Without loss of generality, consider two views taken by the same camera whose calibration matrix is $\mathbf{K}$. In this case, the projections $\vec{y}^j$ and $\vec{y}^k$ can be normalized as:

$$\begin{aligned} \hat{y}^j &= \mathbf{K}^{-1}\vec{y}^j \\ \hat{y}^k &= \mathbf{K}^{-1}\vec{y}^k. \end{aligned} \tag{B.10}$$

Under this assumption, Equation B.1 can be expressed in terms of normalized points:

$$\hat{y}^{k\top}\mathbf{E}\hat{y}^j = 0 \tag{B.11}$$

that can be rewritten using denormalized coordinates as $\vec{y}_k^\top \mathbf{K}^{-\top}\mathbf{E}\mathbf{K}\vec{y}_j = 0$. From this follows that the Essential matrix can be expressed in terms of the Fundamental matrix as:

$$\mathbf{F} = \mathbf{K}^{-\top}\mathbf{E}\mathbf{K}. \tag{B.12}$$

Taking Equation B.9 into account, the Essential matrix can be expressed in terms of the rotation matrix and the translation vector that relate the two cameras:

$$\mathbf{E} = [\vec{t}]_\times \mathbf{R}. \tag{B.13}$$

However, since there is an overall scale ambiguity because of the homogeneous quantities, the Essential matrix has only five degrees of freedom.

The translation vector can be recovered solving the problem $\mathbf{E}^\top \vec{t} = \vec{0}$. Because of the scale factor ambiguity, only the direction can be recovered, so the solution is chosen as the vector complaining $\left\|\vec{t}\right\| = 1$. The opposite direction $-\vec{t}$ is also a valid solution.

In order to obtain the rotation, the matrix $\mathbf{E}$ needs to be decomposed in the form of Equation B.13 where $\left[\vec{t}\right]_\times$ is an antisymmetric matrix and $\mathbf{R}$ is an orthogonal matrix. From the Singular Value Decomposition (SVD) follows that $\mathbf{E} = \mathbf{S}\,\mathrm{diag}\left(\sigma, \sigma, 0\right)\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. The singular value diagonal matrix can be rewritten as:

$$\mathrm{diag}\left(\sigma, \sigma, 0\right) = \sigma \underbrace{\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\mathbf{Z}} \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{W}} \tag{B.14}$$

where $\mathbf{Z}$ is antisymmetric and $\mathbf{W}$ is orthogonal. Taking into account that $\mathbf{E}$ is defined up to an arbitrary scale factor, it can be assumed without loss of generality that $\sigma = 1$. Since $\mathbf{U}$ and$\mathbf{V}$ are orthogonal, the SVD decomposition can be rewritten by means of a product of an antisymmetric matrix and an orthogonal matrix:

$$\mathbf{E} = [\vec{t}]_\times \mathbf{R} = \left(\mathbf{U}\mathbf{Z}\mathbf{U}^\top\right)\left(\mathbf{U}\mathbf{W}\mathbf{V}^\top\right), \tag{B.15}$$

so $\mathbf{R} = \mathbf{U}\mathbf{W}\mathbf{V}^\top$. There are also two possible rotations depending on the sign of the axis. The second solution is obtained as $\mathbf{R} = \mathbf{U}\mathbf{W}^\top\mathbf{V}^\top$, so taking into account the two possible translation vectors, there are a total of four possible camera configurations. The true solution can be obtained as the one that reconstructs points in the front of the camera.

Appendix C

Programming on a GPU

With the arrival of programmable GPUs, many people have been using them to perform more than special effects. GPGPU computing is a recent field of work in computer science. The goal of this area is to take advantage of the high computational power of modern graphic processors (which currently is an order of magnitude higher than CPUs) and their specific operation sets for computer graphics. Among the most common techniques to perform GPGPU computing can be found the use of shaders and, more recently, CUDA. Although other CUDA-like technologies exist, such as Direct Computing (Microsoft) and OpenCL, only CUDA 2.3 will be treated in this section, since it was the one employed in the work, and the three are very similar and most of the concepts can be shared.

GPGPU computing is mainly based in the SIMD (Single Instruction, Multiple Data) programming model. In this model, multiple processing units execute the same instruction over a data set under the supervision of a common control unit, i.e., data is processed in parallel but synchronously.

Part of this appendix has been presented in:

Eskudero, I., Sánchez, J. R., Buchart, C., García-Alonso, A., and Borro, D. "Tracking 3D en GPU Basado en el Filtro de Partículas". In Congreso Español de Informática Gráfica (CEIG'09), pp. 47–55. San Sebastián, Spain. 2009.

C.1 Shaders

Shaders are special programs used to modify the fixed rendering pipeline, employed in conjunction with graphic APIs such as OpenGL or Direct3D. Depending on the graphic API used, the shader programming language to be employed may vary. The most important shading languages are Cg© (NVIDIA®), HLSL (Microsoft®) and GLSL (Khronos Group). Cg and HLSL are very similar, and the three are C-like languages. Different shader types exist to manipulate data in different stages of the pipeline:

- **Vertex shaders**: transform 3D vertices (once per run) to the 2D coordinate system of the viewport. Vertex shaders can modify the position, color and texture coordinates of the vertices, but cannot create new vertices.
- **Geometric shaders**: assemble the geometric primitive that will be sent to the rasterizer. These shaders can create or destroy vertices and are usually employed for tessellation of parametric or implicit surfaces.
- **Fragment shaders**: also known as pixel shaders, compute the color of the individual pixels that come from the rasterizer. Fragment shaders are very flexible and are commonly used in objects lighting and texturing, special effects and even in non-polygonal based visualization, such as volume rendering. Their main disadvantage is that they cannot write data to a different pixel coordinate than the one assigned by the rasterizer.

Given natural analogies with the SIMD model, GPGPU computing usually makes use of fragment shaders to work. Although less formal, it is easier to see how it works in a scheme:

- First, data is stored in textures, as if they were arrays. The only limitations here are those self imposed by the texture structure: all the elements must have the same structure and each of them can stored up to four values of the same basic data type (floats, integers), corresponding to each of the four color components: red, green, blue or alpha channel.
- The viewport is configured appropriately to draw data. For example, if the computation is one output per each input, the viewport must be setup to have the same size as the texture. The rendering output is set to be another texture, so results can be written back to memory.

- The fragment shader is enabled. Textures and any other individual parameters are loaded.

- A textured rectangle is drawn to fulfill the viewport. In this way, each rasterized pixel will correspond to a texel.

In this way, the fragment shader is executed for each element of the data and the results are written to the specified texture that can be later read or used as the input of another shader (thus avoiding the costly transfer to and from the main memory).

Some common GPGPU applications are iterative processes of the form of $x_{i+1} = f(x_i)$. In this case, a technique called ping–pong rendering is commonly used. It consists in the use of two interchangeable textures of the same size and structure, one for reading the data and one for writing; after each iteration, their roles are simply swapped.

Fragment shaders impose some restrictions in the programming model that must be taken into account:

- No random position scattering (writing). A fragment shader can only write in the position specified by the rasterizer. For example, it is not possible for a shader to store its results in different cells of a grid; in this case, the value of each cell must be computed by an individual shader and each shader must have the corresponding rasterizer position. This follows that the viewport determines the structure of the output.

- Only modern GPUs allow branching (execution bifurcations produced by conditional statements and loops), but its use must be reduced as much as possible in order to avoid high speed penalties. If two threads of the same shader enter different regions of a branch, each set is executed by the two threads but only the corresponding memory states are kept for each one.

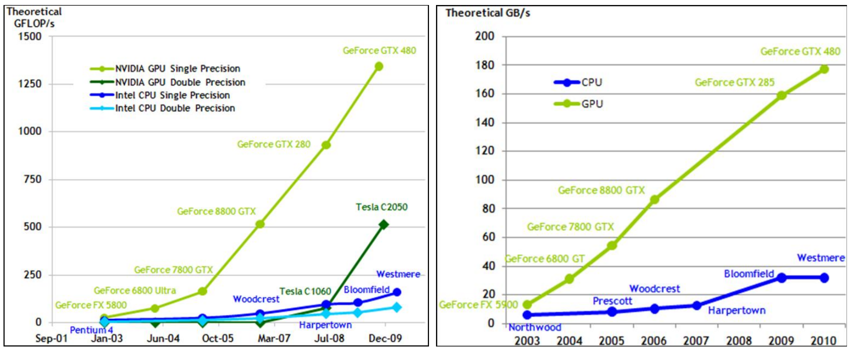
- Transfers between main memory and graphic memory must be carefully scheduled to reduce bandwidth overhead.

- Current graphics hardware imposes different restrictions with respect to the texture size. Initially these constrains included not only the maximum size, but also that the size must be a power-of-two. Nowadays this limitation has disappeared and the maximum size is often 4096×4096.

C.2 CUDA

CUDA™ (*Computing Unified Device Architecture*) is a C extension developed by NVIDIA®, which allows a higher level of abstraction than that obtained with shaders. CUDA capable devices can accelerate the execution of computationally intensive programs exploiting the data level parallelism of the executed algorithms. These devices can work together in order to solve large problems and they always work within a host (a PC). This technology was introduced in desktop computers with the G80 GPU in late 2006 that was included in the GeForce® 8800 graphics card family. At the same time, NVIDIA launched its Tesla® dedicated GPGPU device. Basically, the only difference between a Tesla device and a normal GPU is that the first lacks a display output. Recent Tesla devices based on the Fermi™ architecture also have four times more arithmetic precision than its graphics device equivalent.

As seen in Figure C.1 the peak performance of GPUs has been greatly improved compared with general purpose microprocessors. However, this performance can only be achieved exploiting the data level parallelism that CUDA devices require.



(a) Floating point operations per second. (b) Memory bandwidth comparison.

Figure C.1: Performance evolution comparison between CPUs and GPUs. (NVIDIA, 2010)

CUDA has solved some of the main disadvantages of GPGPU programming through shaders, e.g. the fixed-position scattering limitation. This has allowed numerous algorithms to become easier to implement in the GPU. Regardless of this ease, it is still necessary to design the algorithms to use efficiently the resources of the CUDA device.

Additionally to CUDA, other similar technologies exist, such as Direct Computing (Microsoft) and OpenCL (Khronos Group). The following sections will introduce some basic concepts on CUDA, however, most of them are applicable to other computing technologies.

C.2.1 CUDA Program Structure

A CUDA program is built using both regular functions, that are executed in the host, and CUDA functions, called kernels, that are executed in the CUDA device. These functions are separately compiled using the standard C++ compiler for the CPU code and the NVIDIA *nvcc* compiler for the CUDA kernels.

CUDA kernels, when called, are executed many times in parallel using the threading capabilities of the device. As shown in Figure C.2, these threads are grouped in blocks, that at the same time are grouped in a grid. Grids and blocks are one, two or three dimensional arrays and their size is only limited by the CUDA device.

The size of the grid and the blocks can be set by the user in every kernel call. Each thread executes the same kernel code and has a unique ID that can be accessed from the kernel giving a natural way to do computations across the elements of a matrix.

C.2.2 Occupancy

When a group of threads is received to be executed, the multiprocessor device splits them into warps that are individually scheduled. A *warp* is a group of threads (32 by the time this memory was written) that starts together at the same program address but that are free to branch independently. A warp executes a command at a time, so full efficiency is realized when all the threads of the warp follow the same instruction path.

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. It is an important metric in determining how effectively the hardware is used: a higher occupancy eases the device to hide memory latency and therefore helps to improve performance. For more information about occupancy and CUDA programs optimization, refer to (NVIDIA, 2010).

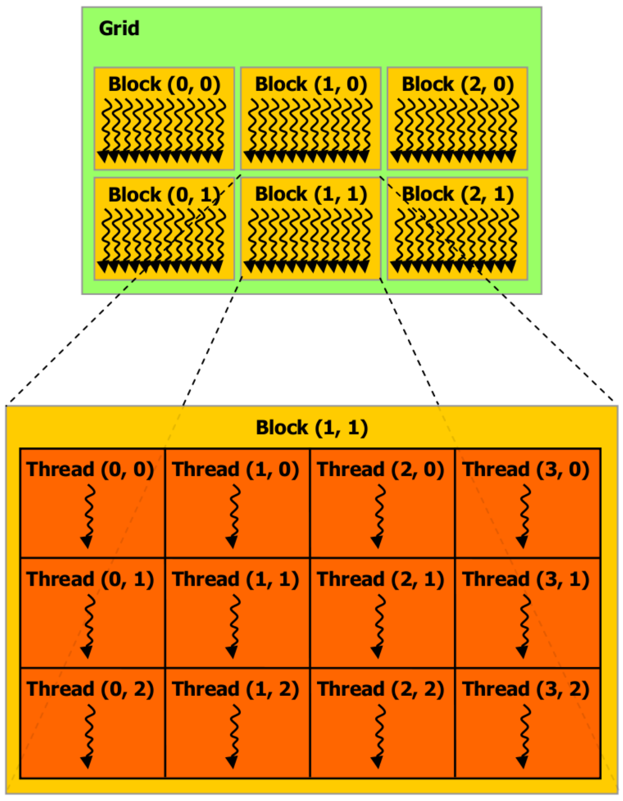


Figure C.2: Thread hierarchy. (NVIDIA, 2010)

C.2.3 CUDA Memory Model

CUDA devices have their own memory space and threads can not access directly to the host memory. The data must to be transferred from host to device in order to make the computations and from device to host in order to get the results.

Figure C.3 shows an overview of the device memory. Each thread has a private local memory where local variables are stored. At block level, there is a shared memory visible to all threads of the block whose size can be set dynamically before the kernel invocation. Finally, there is a global memory that can be randomly accessed by all threads and it is persistent across kernel executions. There are also two read only memories, i.e., constant memory and texture memory. Constant memory is a small space that can be accessed randomly very fast. In contrast texture memory, which inherits from graphics applications, is a large memory that can be organized in up to three dimensions. It is locally cached and can be accessed through a hardware interpolator.

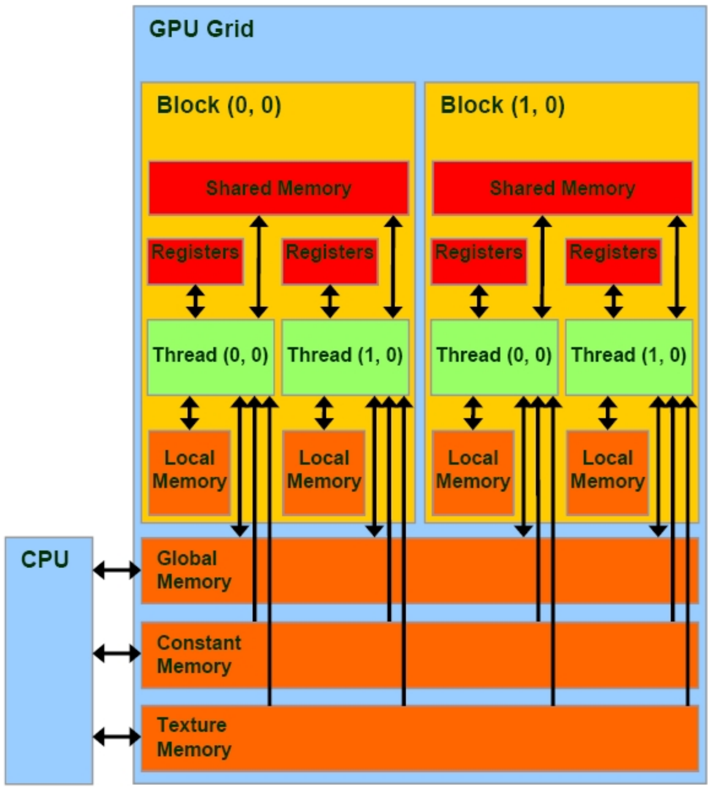


Figure C.3: Memory hierarchy. [1]

Compared to traditional shaders, CUDA threads can randomly read and write in any position of the global memory. However, this memory has a lot of access latency and should be used with care. Although having many threads can hide the access latency, the global memory has a limited amount of bandwidth and can be easily collapsed making computing units go idle. This problem can be solved moving the data from global memory to shared memory, which has much less latency, and using it across the threads of the same block.

CUDA devices can read up to 16 bytes from global memory in a single instruction. If threads of a half-warp access simultaneously to different words lying in the same memory segment, accesses are coalesced in a single memory

[1]Courtesy: NVIDIA

transaction of 4, 8 or 16 bytes. In general, a memory transaction will be executed for each memory segment requested by threads in a block. Figure C.4 shows an example of coalesced memory accesses that are desirable in any CUDA program.

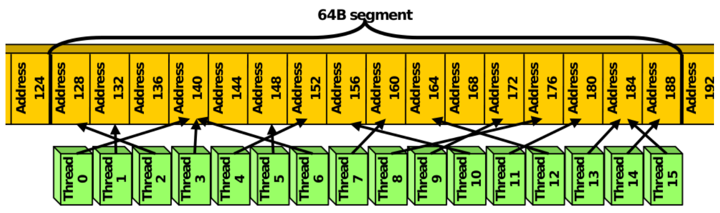


Figure C.4: Example of a coalesced memory access. (NVIDIA, 2010)

In conclusion, in order to use efficiently the CUDA device, special care should be taken in the data level parallelism of the algorithm and in the memory access patterns, trying to minimize the memory transfers between the host and the device. For these reasons, many existing algorithms can not be directly implemented in CUDA, needing new approaches that exploit the benefits of this architecture.

Appendix D

Generated Publications

D.1 Conference Proceedings

Sánchez, J. R., Álvarez, H., and Borro, D. "Towards Real Time 3D Tracking and Reconstruction on a GPU Using Monte Carlo Simulations". In International Symposium on Mixed and Augmented Reality (ISMAR'10), pp. 185–192. Seoul, Korea. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. "GFT: GPU Fast Triangulation of 3D Points". In Computer Vision and Graphics (ICCVG'10), volume 6375 of Lecture Notes in Computer Science, pp. 235–242. Warsaw, Poland. 2010.

Sánchez, J. R. and Borro, D. "Automatic Affine Structure Recovery Using RANSAC". In Congreso Español de Informática Gráfica (CEIG'10), pp. 155–164. Valencia, Spain. 2010.

Eskudero, I., Sánchez, J. R., Buchart, C., García-Alonso, A., and Borro, D. "Tracking 3D en GPU Basado en el Filtro de Partículas". In Congreso Español de Informática Gráfica (CEIG'09), pp. 47–55. San Sebastián, Spain. 2009.

Sánchez, J. R. and Borro, D. "Non Invasive 3D Tracking for Augmented Video Applications". In IEEE Virtual Reality 2007 Conference, Workshop "Trends and Issues in Tracking for Virtual Environments", pp. 22–27. Charlotte, NC, USA. 2007.

D.2 Poster Proceedings

Sánchez, J. R., Álvarez, H., and Borro, D. "GPU Optimizer : A 3D reconstruction on the GPU using Monte Carlo simulations". In Poster proceedings of the 5th International Conference on Computer Vision Theory and Applications (VISAPP'10), pp. 443–446. Angers, France. 2010.

Sánchez, J. R. and Borro, D. "Automatic Augmented Video Creation for Markerless Environments". In Poster Proceedings of the 2nd International Conference on Computer Vision Theory and Applications (VISAPP'07), pp. 519–522. Barcelona, Spain. 2007.

Barandiaran, J., Moreno, I., Ridruejo, F. J., Sánchez, J. R., Borro, D., and Matey, L. "Estudios y Aplicaciones de Realidad Aumentada en Dispositivos Móviles". In Conferencia Española de Informática Gráfica (CEIG'05), pp. 241–244. Granada, Spain. 2005.

References

Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. "Building rome in a day". In *IEEE International Conference on Computer Vision (ICCV'09)*, pp. 72–79. Kyoto, Japan. September, 2009.

Azarbayejani, A. and Pentland, A. "Recursive estimation of motion, structure, and focal length". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, N. 6, pp. 562–575. June, 1995.

Barandiaran, J., Moreno, I., Ridruejo, F. J., Sánchez, J. R., Borro, D., and Matey, L. "Estudios y Aplicaciones de Realidad Aumentada en Dispositivos Móviles". In *Conferencia Española de Informática Gráfica (CEIG'05)*, pp. 241–244. Granada, Spain. 2005.

Bay, H., Ess, A., Tuytelaars, T., and Vangool, L. "Speeded-Up Robust Features (SURF)". *Computer Vision and Image Understanding*, Vol. 110, N. 3, pp. 346–359. June, 2008.

Bouguet, J. "Pyramidal implementation of the lucas kanade feature tracker description of the algorithm". 2000.

Broida, T. J., Chandrashekhar, S., and Chellappa, R. "Recursive 3-D motion estimation from a monocular image sequence". *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 26, N. 4, pp. 639–656. 1990.

Brown, D. C. "Decentering Distortion of Lenses". *Photogrammetric Engineering*, Vol. 32, N. 3, pp. 444–462. 1966.

Brown, D. C. "The bundle adjustment - progress and prospects". In *XIIIth Congress of the International Society for Photogrammetry (ISPRS'76)*, pp. 1–33. Helsinki, Finland. 1976.

Canny, J. “A Computational Approach to Edge Detection”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, N. 6, pp. 679–698. November, 1986.

Caudell, T. P. and Mizell, D. “Augmented reality: an application of heads-up display technology to manual manufacturing processes”. In *Hawaii International Conference on System Sciences (HICSS’92)*, pp. 659–669. Kauai, USA. 1992.

Changchang, W. “SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)”. 2007.

Chekhlov, D., Pupilli, M., Mayol-Cuevas, W., and Calway, A. “Real-time and robust monocular SLAM using predictive multi-resolution descriptors”. In *2nd International Symposium on Visual Computing*, volume 4292, pp. 276–285. Lake Tahoe, NV, USA. 2006.

Chiuso, A., Favaro, P., and Soatto, S. “Structure from motion causally integrated over time”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24, N. 4, pp. 523–535. April, 2002.

Cornelis, K. *From uncalibrated video to augmented reality*. PhD thesis, Katholike Universiteit Leuven. 2004.

Davison, A. J. “Real-Time Simultaneous Localisation and Mapping with a Single Camera”. In *IEEE International Conference on Computer Vision (ICCV’03)*, pp. 1403–1410 vol.2. Nice, France. 2003.

Debevec, P. “Image-Based Lighting”. *IEEE Computer Graphics Applications*, Vol. 22, N. 2, pp. 26–34. 2002.

Duda, R. O. and Hart, P. E. “Use of the Hough transformation to detect lines and curves in pictures”. *Communications of the ACM*, Vol. 15, N. 1, pp. 11–15. January, 1972.

Eskudero, I., Sánchez, J. R., Buchart, C., García-Alonso, A., and Borro, D. “Tracking 3D en GPU Basado en el Filtro de Partículas”. In *Congreso Español de Informática Gráfica (CEIG’09)*, pp. 47–55. San Sebastián, Spain. 2009.

Faugeras, O. “Stratification of three-dimensional vision: projective, affine, and metric representations”. *Journal of the Optical Society of America*, Vol. 12, N. 3, pp. 465–484. 1995.

Faugeras, O., Luong, Q., and Maybank, S. “Camera self-calibration: Theory and experiments”. In *European Conference on Computer Vision (ECCV’92)*, volume 588, pp. 321–334. Santa Margherita Ligure, Italy. 1992.

Fischler, M. A. and Bolles, R. C. “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography”. *Communications of the ACM*, Vol. 24, N. 6, pp. 381–395. 1981.

Fox, D. “Adapting the Sample Size in Particle Filters Through KLD-Sampling”. *The International Journal of Robotics Research*, Vol. 22, N. 12, pp. 985–1003. December, 2003.

Gordon, N., Salmond, D., and Smith, A. “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In *IEE Proceedings of Radar and Signal Processing*, volume 140, pp. 107–113. 1993.

Harris, C. and Stephens, M. “A combined corner and edge detector”. *Alvey vision conference*, Vol. 15, pp. 147–151. 1988.

Harris, M. “Optimizing Parallel Reduction in CUDA”. 2008.

Hartley, R. I. “Euclidean reconstruction from uncalibrated views”. *Applications of invariance in computer vision*, Vol. 825, pp. 237–256. 1994.

Hartley, R. I. “Self-calibration from multiple views with a rotating camera”. In *European Conference on Computer Vision (ECCV’94)*, volume 800, pp. 471–478. Stockholm, Sweden. 1994.

Hartley, R. I. “Kruppa’s Equations derived from the Fundamental Matrix”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, N. 2, pp. 133–135. March, 1997.

Hartley, R. I. and Zisserman, A. *Multiple View Geometry in Computer Vision* (ISBN: 0521-54051-8). Cambridge University Press. 2004.

Hemayed, E. “A survey of camera self-calibration”. In *IEEE Conference on Advanced Video and Signal-Based Surveillance*, pp. 351–357. Miami, FL, USA. 2003.

Henderson, S. J. and Feiner, S. “Evaluating the benefits of augmented reality for task localization in maintenance of an armored personnel carrier turret”. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR’09)*, pp. 135–144. Orlando, FL,USA. October, 2009.

Howes, L. and Thomas, D. *Efficient random number generation and application using CUDA* (ISBN: 0-321-51526-9), chapter 37. NVIDIA Corporation. 2007.

Huber, P. J. *Robust Statistics* (ISBN: 9780471418054). Wiley. 1981.

Julier, S. J. and Uhlmann, J. K. "A New extension of the Kalman filter to nonlinear systems". In *International Symposium of Aerospace/Defense Sensing, Simulation and Controls (ISASSC'97)*, pp. 182–193. Orlando, FL,USA. 1997.

Kahl, F. "Multiple View Geometry and the L∞ norm". In *International Conference on Computer Vision (ICCV'05)*, pp. 1002–1009. Beijing, China. 2005.

Kalman, R. E. "A new approach to linear filtering and prediction problems". *Journal of basic Engineering*, Vol. 82, N. Series D, pp. 35–45. 1960.

Kanazawa, Y. and Kanatani, K. "Do we really have to consider covariance matrices for image features?" In *IEEE International Conference on Computer Vision (ICCV'01)*, number 5, pp. 301–306. Vancouver, Canada. 2001.

Kapasi, U. J., Rixner, S., Dally, W. J., Khailany, B., Mattson, P., and Owens, J. D. "Programmable stream processors". *Computer*, Vol. 36, N. 8, pp. 54–62. August, 2003.

Kato, H. and Billinghurst, M. "Marker tracking and HMD calibration for a video-based augmented reality conferencing system". In *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*, pp. 85–94. San Francisco, CA, USA. 1999.

Kirk, D. B. and Hwu, W. W. *Programming Massively Parallel Processors* (ISBN: 978-0-12-381472-2). 2010.

Klein, G. and Murray, D. "Parallel tracking and mapping for small AR workspaces". In *International Symposium on Mixed Augmented Reality (ISMAR'07)*, pp. 1–10. Nara, Japan. November, 2007.

LaViola, J. J. "Double exponential smoothing: an alternative to Kalman filter-based predictive tracking". In *Proceedings of the workshop on Virtual environments*, volume 39, pp. 199–206. 2003.

Levenberg, K. “A Method for the Solution of Certain non-linear problems in Least Squares.” *The Quarterly of Applied Mathematics*, Vol. 2, pp. 164–168. 1944.

Liu, J. S. and Chen, R. “Sequential Monte Carlo Methods for Dynamic Systems”. *Journal of the American Statistical Association*, Vol. 93, N. 443, p. 1032. September, 1998.

Longuet-Higgins, H. C. “A computer algorithm for reconstructing a scene from two projections”. *Nature*, Vol. 293, N. 5828, pp. 133–135. September, 1981.

López-Moreno, J., Hadap, S., Reinhard, E., and Gutierrez, D. “Light source detection in photographs”. In *Congreso Español de Informática Gráfica (CEIG’09)*, volume 0, pp. 161–167. San Sebastián, Spain. 2009.

Lourakis, M. I. A. and Argyros, A. A. “Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment?” In *IEEE International Conference on Computer Vision (ICCV’05)*, pp. 1526–1531. Beijing, China. 2005.

Lourakis, M. I. A. and Argyros, A. A. “SBA: A Software Package for Generic Sparse Bundle Adjustment”. *ACM Transactions on Mathematical Software*, Vol. 36, N. 1, pp. 1–30. March, 2009.

Lowe, D. G. “Object recognition from local scale-invariant features”. In *International Conference on Computer Vision (ICCV’99)*, volume 2, pp. 1150–1157 vol.2. Kerkyra, Corfu, Greece. 1999.

Lowe, D. G. “Distinctive image features from scale-invariant keypoints”. *International journal of computer vision*, Vol. 60, N. 2, pp. 91–110. November, 2004.

Marquardt, D. W. “An Algorithm for Least Squares Estimation of Nonlinear Parameters”. *Journal of the Society for Industrial and Applied Mathematics*, Vol. 11, N. 2, pp. 431–441. August, 1963.

Marr, D. and Poggio, T. “A Computational Theory of Human Stereo Vision”. *Proceedings of the Royal Society B: Biological Sciences*, Vol. 204, N. 1156, pp. 301–328. May, 1979.

Meteopolis, N. and Ulam, S. “The monte carlo method”. *Journal of the American Statistical Association*, Vol. 44, N. 247, pp. 335–341. 1949.

Milgram, P. and Kishino, F. “A taxonomy of mixed reality visual displays”. *IEICE Transactions on Information Systems*, Vol. E77-D, N. 12, pp. 1321–1329. 1994.

Moravec, H. P. “Towards Automatic Visual Obstacle Avoidance”. In *International Joint Conference on Artificial Intelligence (IJCAI’77)*, pp. 584–590. Cambridge, MA, USA. 1977.

NVIDIA. *NVIDIA CUDA C Programming Guide*. 2010.

Ozuysal, M., Calonder, M., Lepetit, V., and Fua, P. “Fast keypoint recognition using random ferns.” *IEEE transactions on pattern analysis and machine intelligence*, Vol. 32, N. 3, pp. 448–61. March, 2010.

Pupilli, M. and Calway, A. “Real-time visual slam with resilience to erratic motion”. In *International Conference on Computer Vision and Pattern Recognition (CVPR’06)*, pp. 1244–1249. New York, NY, USA. 2006.

Qian, G. and Chellappa, R. “Structure from motion using sequential monte carlo methods”. *International Journal of Computer Vision*, Vol. 59, N. 1, pp. 5–31. August, 2004.

Sánchez, J. R., Álvarez, H., and Borro, D. “GFT: GPU Fast Triangulation of 3D Points”. In *Computer Vision and Graphics (ICCVG’10)*, volume 6375 of *Lecture Notes in Computer Science*, pp. 235–242. Warsaw, Poland. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. “GPU Optimizer : A 3D reconstruction on the GPU using Monte Carlo simulations”. In *Poster proceedings of the 5th International Conference on Computer Vision Theory and Applications (VISAPP’10)*, pp. 443–446. Angers, France. 2010.

Sánchez, J. R., Álvarez, H., and Borro, D. “Towards Real Time 3D Tracking and Reconstruction on a GPU Using Monte Carlo Simulations”. In *International Symposium on Mixed and Augmented Reality (ISMAR’10)*, pp. 185–192. Seoul, Korea. 2010.

Sánchez, J. R. and Borro, D. “Automatic Augmented Video Creation for Markerless Environments”. In *Poster Proceedings of the 2nd International Conference on Computer Vision Theory and Applications (VISAPP’07)*, pp. 519–522. Barcelona, Spain. 2007.

Sánchez, J. R. and Borro, D. “Non Invasive 3D Tracking for Augmented Video Applications”. In *IEEE Virtual Reality 2007 Conference, Workshop “Trends and Issues in Tracking for Virtual Environments”*, pp. 22–27. Charlotte, NC, USA. 2007.

Sánchez, J. R. and Borro, D. “Automatic Affine Structure Recovery Using RANSAC”. In *Congreso Español de Informática Gráfica (CEIG’10)*, pp. 155–164. Valencia, Spain. 2010.

Shi, J. and Tomasi, C. “Good features to track”. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR’94)*, pp. 593–600. Seattle, WA, USA. 1994.

Stewénius, H., Engels, C., and Nistér, D. “Recent developments on direct relative orientation”. *ISPRS Journal of Photogrammetry and*, Vol. 60, N. 4, pp. 284–294. June, 2006.

Sutherland, I. “A head-mounted three dimensional display”. In *Proceedings of the Fall Joint Computer Conference*, volume 33, pp. 757–764. San Francisco, California, USA. 1968.

Triggs, B., McLauchlan, P., Hartley, R. I., and Fitzgibbon, A. “Bundle adjustment - A modern synthesis”. *Vision algorithms: Theory and Practice*, Vol. 1883, pp. 298–372. 2000.

Tsai, R. “A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses”. *IEEE Journal of robotics and Automation*, Vol. 3, N. 4, pp. 323–344. August, 1987.

Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. “Real-Time Detection and Tracking for Augmented Reality on Mobile Phones”. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 16, N. 3, pp. 355–368. 2010.

Wang, G., Tsui, H., Hu, Z., and Wu, F. “Camera calibration and 3D reconstruction from a single view based on scene constraints”. *Image and Vision Computing*, Vol. 23, N. 3, pp. 311–323. March, 2005.

Welch, G. and Bishop, G. “An introduction to the Kalman filter”. In *Siggraph’01*. Los Angeles, CA, USA. 2001.

Williams, B., Klein, G., and Reid, I. “Real-time SLAM relocalisation”. In *2007 IEEE 11th International Conference on Computer Vision (ICCV’07)*, pp. 1–8. Rio de Janeiro, Brazil. October, 2007.

Zhang, Z. “A flexible new technique for camera calibration”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, N. 11, pp. 1330–1334. 2000.

Zhong, H. and Hung, Y. “Self-calibration from one circular motion sequence and two images”. *Pattern Recognition*, Vol. 39, N. 9, pp. 1672–1678. September, 2006.

Index