Monocular-SLAM dense mapping algorithm and hardware architecture for FPGA acceleration

Abiel Aguilar-Gonzalez

Monocular-SLAM dense mapping algorithm and hardware architecture for FPGA acceleration

by

M. Sc. Abiel Aguilar-González

A thesis proposal submitted in partial fulfillment of
the requirements for the degree of

DOCTOR EN CIENCIAS EN LA ESPECIALIDAD DE CIENCIAS COMPUTACIONALES

at the

Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE)

INAOE, Computer science department
Luis Enrique Erro #1 Sta. Ma. Tonantzintla, Puebla, Mexico.
Thèse soutenue le 13-06-2019

Advisors:

Dr. Miguel Octavio Arias-Estrada, INAOE, Mexico
Prof. François Berry, Université Clermont Auvergne (UCA), France

Updated version at: 22/05/2019

Abstract

Simultaneous Localization and Mapping (SLAM) is the problem of constructing a 3D map while simultaneously keeping track of an agent location within the map. In recent years, work has focused on systems that use a single moving camera as the only sensing mechanism (monocular-SLAM). This choice was motivated because nowadays, it is possible to find inexpensive commercial cameras, smaller and lighter than other sensors previously used and, they provide visual environmental information that can be exploited to create complex 3D maps while camera poses can be simultaneously estimated. Unfortunately, previous monocular-SLAM systems are based on optimization techniques that limits the performance for real-time embedded applications. To solve this problem, in this work, we propose a new monocular SLAM formulation based on the hypothesis that it is possible to reach high efficiency for embedded applications, increasing the density of the point cloud map (and therefore, the 3D map density and the overall positioning and mapping) by reformulating the feature-tracking/feature-matching process to achieve high performance for embedded hardware architectures, such as FPGA or CUDA. In order to increase the point cloud map density, we propose new feature-tracking/feature-matching and depth-from-motion algorithms that consists of extensions of the stereo matching problem. Then, two different hardware architectures (based on FPGA and CUDA, respectively) fully compliant for real-time embedded applications are presented. Experimental results show that it is possible to obtain accurate camera pose estimations. Compared to previous monocular systems, we are ranked as the 5th place in the KITTI benchmark suite, with a higher processing speed (we are the fastest algorithm in the benchmark) and more than $\times 10$ the density of the point cloud from previous approaches.

Resumen

La localización y mapeo simultáneo (SLAM, por sus siglas en inglés) es el problema de construir un mapa en 3D mientras al mismo tiempo, se realiza un seguimiento de la ubicación de un agente dentro del mapa. En los últimos años, el trabajo previo se ha centrado en sistemas que utilizan una única cámara en movimiento como sensor (SLAM monocular). Esta elección fue motivada porque hoy en día, es posible encontrar cámaras comerciales económicas, mas pequeñas y livianas que otros sensores utilizados anteriormente, y proporcionan información visual que puede ser explotada para crear mapas 3D complejos, mientras que la posición de la cámara puede estimarse simultáneamente. Desafortunadamente, los sistemas SLAM monocular anteriores se basan en técnicas de optimización que limitan el rendimiento en aplicaciones embebidas en tiempo real. Para resolver este problema, en este trabajo proponemos una nueva formulación de SLAM monocular basada en la hipótesis de que es posible alcanzar una alta eficiencia para aplicaciones embebidas, aumentando la densidad de los puntos utilizados para estimar el posicionamiento y mapeo simultaneo mediante la reformulación del proceso de feature-tracking/feature-matching para alcanzar un alto rendimiento para arquitecturas de hardware embebidas, como FPGA o CUDA y al mismo tiempo, incrementar la densidad con respecto a los algoritmos anteriores. Para esto, proponemos nuevos algoritmos de feature-tracking/feature-matching y depth-from-motion, ambos definidos como una extención del problema estéreo. Luego, se presentan dos diferentes arquitecturas de hardware (basadas en FPGA y CUDA, respectivamente) para aplicaciones embebidas en tiempo real. Los resultados experimentales demuestran que es posible obtener estimaciones precisas de la posición de la cámara (considerando los sistemas monoculares anteriores, estamos clasificados como el quinto lugar en el conjunto de pruebas de KITTI) con una mayor velocidad de procesamiento (somos el algoritmo más rápido en la prueba) y más de $\times 10$ la densidad de la nube de puntos de los enfoques anteriores (estatus en el benchmark de KITTI en enero de 2019).

Résumé

La localisation associée à une cartographie simultanées (SLAM) revient à un problème de construction d'une carte 3D tout en conservant la trajectoire passée de "l'agent" dans la carte. Ces dernières années, les travaux se sont concentrés sur des systèmes utilisant une seule caméra mobile comme unique système de détection (monoculaire-SLAM). Ce choix est motivé par le fait qu'il est aujourd'hui possible de trouver des caméras commerciales peu coûteuses, plus petites et plus légères que les capteurs utilisés auparavant tout en fournissant des informations visuelles qui peuvent être exploitées pour créer des cartes 3D complexes. Malheureusement, les systèmes de SLAM monoculaires sont basés sur des techniques d'optimisation qui limitent les performances dans des contextes embarqués et temps réel. Pour cela, nous proposons une nouvelle formulation du SLAM monoculaire permettant d'atteindre une efficacité élevée pour un système embarqué. Le principe est d'augmenter la densité de la carte des nuages de points (et donc la densité de la carte 3D et le positionnement et la cartographie d'ensemble) en reformulant le processus de suivi et de correspondance des caractéristiques afin d'être complient avec des architectures matérielles basée sur des processeur tel que des FPGA ou des GPU. Afin d'augmenter la densité de la carte des nuages de points, nous proposons de nouveaux algorithmes de suivi et de correspondance des caractéristiques et de profondeur à partir du mouvement, qui revient à une extension du problème d'appariement stéréo. Ensuite, deux architectures matérielles différentes (basées sur FPGA et GPU) entièrement compatibles pour les applications embarquées temps réel sont présentées. Les résultats expérimentaux démontrent qu'il est possible d'obtenir des estimations précises de la pose de la caméra. Par rapport aux systèmes monoculaires de l'état de l'art, nous nous plaçons en 5ème position dans les benchmarks KITTI, avec une vitesse de traitement et une densité du nuage de points 10 fois plus élevée.

Acknowledgments

I would like to thank to Dr. Miguel Octavio Arias Estrada and Dr. François Berry for their support and guidance for the developing of this research work.

I would like to thank to the members of my academic committee for their comments. Thanks to Dr. Luis Enrique Sucar Succar, Dr. René Armando Cumplido Parra, Dra. Claudia Feregrino Uribe, Dr. Alfonso Martínez Cruz. Dr. Víctor Manuel Brea Sánchez, Dr. Madaín Pérez Patricio and Dr. Jean-Philippe Diguet.

I would like to thank to the CONACyT (scholarship No. 567804) and Campus France ("bourses d'excellence EIFFEL", dossier No. MX17-00063) for the financial support for the developing of this research work.

Contents

List of Figures

List of Tables

Acronyms

SIFT (Scale-Invariant Feature Transform)

SURF Speeded Up Robust Features

ORB Oriented FAST and Rotated BRIEF

GPU Graphics Processing Unit

GPGPU General-Purpose Computing on Graphics Processing Units

BoW Bag of Words

CUDA Compute Unified Device Architecture

RGB Red, Green and Blue

CPU Central Processing Unit

GHz GigaHertz

RAM Random-Access Memory

GB GigaByte

PC Personal Computer

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) is the problem of constructing a map with respect to an unknown environment while simultaneously keeping track of an agent location within the map. In general, the SLAM problem investigates two main areas: localization and mapping [123]. Localization consists of determining the spatial position from a moving agent within an unknown environment at a given time. On the other hand, the mapping integrates partial observations from an unknown environment into maps (usually 3D reconstructions). In order to develop a SLAM solution, it is necessary to have sensors that observe the environment and, if possible, sensors that provide information about the trajectory of the agent. There are several sensors available for this purpose (odometers, gyrometers, laser rangefinders, cameras, sonars, etc.), and several SLAM solutions based on: EKF (Extended Kalman filter), graph optimization and visual features are available in the literature. In recent years, the most popular trend is for visual features-based solutions, using a single camera in motion as the only detection mechanism (monocular-SLAM). In this case, visual information such as color, texture and light intensity offers an important advantage compared to other sensors, since the images can be exploited to create appearance-based descriptions of the map components [119]. In addition, this approach requires low power consumption and cost compared to other visual-SLAM formulations (stereo-based, multi camera-based or RGBD-based solutions), and avoids addressing the cameras synchronization, the interpretation of the different responses of each image sensor to color/luminance and the mechanical alignment between cameras.

1.1 Monocular-SLAM: general formulation, performance and limitations

SLAM systems that use a single camera as sensor (monocular-SLAM) have the problem of estimating the pose of a moving camera simultaneously a 3D representation of the observed scene is constructed. In all cases, real-time processing is desirable because most real-world SLAM applications, such as autonomous vehicle navigation [131], mobile robotics [99, 130] and augmented reality [28] require a real-time processing. In the monocular-SLAM formulation, a single camera captures visual information of a scene. In this case, 3D information is unknown [119], so to estimate the 3D position, it is necessary to consider at less two different viewpoints of the same scene, as shown in **Fig.** 1.1.

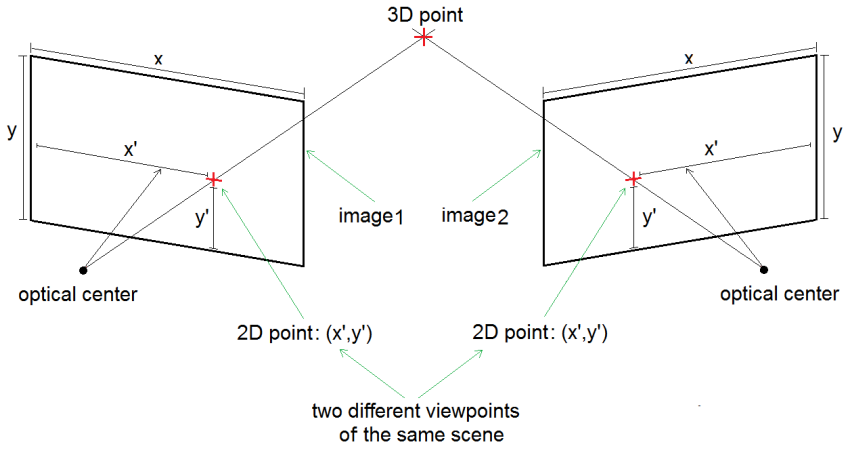


Figure 1.1. Camera geometry of two viewpoints. The 3D position of an observed scene point is unknown in a single image; however, this 3D position can be estimated (thought optimization techniques) if the same point is observed from different viewpoints.

In order to solve the monocular-SLAM problem, first, a 2D point-based set has to be created by detecting feature points in an image (the first viewpoint). Common feature points are corners, edges or intersections. **Fig.** 1.2 shows the feature points detected by applying the FAST corner detection algorithm [112] over the first viewpoint (the first frame) from a video sequence. After the feature points (from the first viewpoint) are detected, they have to be tracked across different viewpoints (frames) captured by the moving camera. There are several feature tracking algorithms in the literature. Algorithms based on two viewpoints such as SIFT [82], ORB [113] or SURF [15], allow the tracking of

long trajectories. On the other hand, algorithms based on frame-by-frame tracking, such as KL [83], KLT [132] and Mean Shift [29]; allow accurate results with sub pixel accuracy. **Fig.** 1.3 shows the feature tracking across two consecutive frames from a video sequence applying the KLT algorithm. The circles represent the feature points while lines show the point correspondences between images. Finally, the point matches generated by any type of feature tracking algorithm can be used to compute the 3D position of the tracked points and, at the same time, to estimate the camera pose for each frame where the feature points were tracked. For this purpose, there are two main approaches: stochastic-based methods and optimization-based methods. Stochastic methods use the features observation in a recursive way to estimate the camera pose and map [57]. Methods based on optimization can be defined as an optimization problem where the variables to be optimized are the camera pose and 3D positions [57]. **Fig.** 1.4 illustrates the SLAM solution obtained by applying the Least Squares optimization technique over the feature tracking shown in **Fig.** 1.3.

In the last few years, monocular-SLAM algorithms have provided a useful tool for several computer vision applications: augmented reality, autonomous vehicle navigation, etc. Unfortunately, in previous work, monst monocular-SLAM solutions solutions have an iterative behavior and requires relatively high computational resources. As a result, several monocular-SLAM algorithms have limitations for embedded applications, since processing speed is often limited between 5-10 estimations per second. On the other hand, traditional monocular-SLAM systems extract feature points by applying any type of feature extraction algorithm. For this purpose, several algorithms are available: FAST [112], SUSAN [120], Harris [56], etc. Nevertheless, in most cases the maximum number of features that can be extracted varies between 0.5% and 4.0% of all pixels in the image, depending on the extraction algorithm of the selected function and its configuration. There are some works that after the camera pose estimation, estimate the depth for all pixels in the image (dense monocular-SLAM) [35, 55, 129]. However, these works have high hardware requirements (at least two high-end GPUs in SLI) and this is a limitation for embedded applications. In practice, traditional monocular-SLAM systems (suitable for embedded applications) work with configurations that extract feature points that correspond to around 1% of the pixels of the processed image [119]. This limits the performance of real-world applications, since only 1% of the image points is used to obtain the 3D information, therefore, the understanding of the visual environment, the application of high-level descriptors and the recognition of objects/structures within the map have a low density (sparse point clouds) and low stability in real-world scenarios.

(a) Input image **(b)** Feature points

Figure 1.2. Visual features extraction applying the FAST algorithm.

Figure 1.3. Feature matching applying the KLT algorithm.

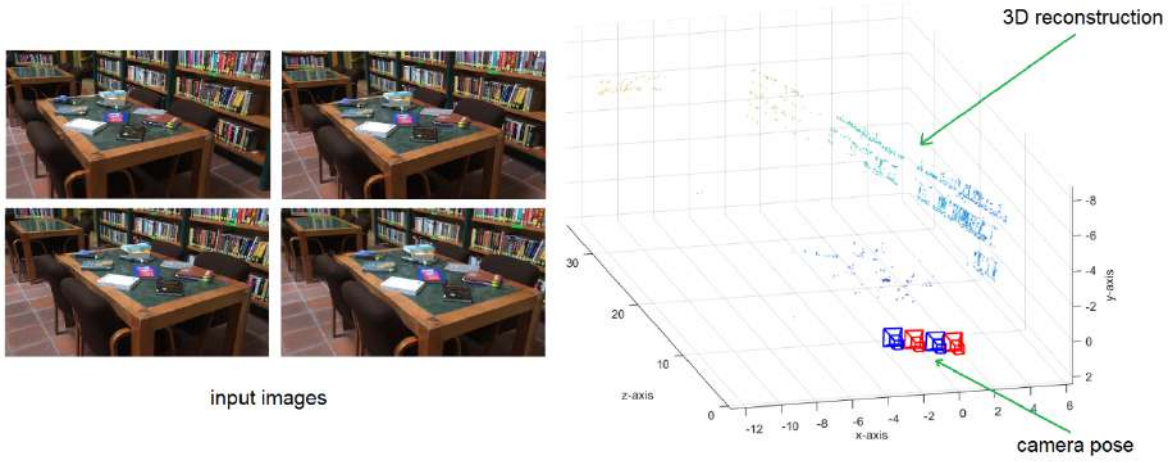


Figure 1.4. Example of a traditional monocular-SLAM solution. Using the FAST algorithm as visual feature extractor, the KLT algorithm as feature matching core and applying the Least Squares algorithm as optimization technique.

1.2 Problem definition

The scientific problem that investigates this work consists in the "architectural limitations for real-time embedded applications" in monocular-SLAM systems based on RGB image sensors.

- **Definition:** In previous works, the most accurate and used solution is based on optimization techniques implemented in sequential processors. This makes it possible to achieve high accuracy for the camera pose estimation but limits the processing speed, the embedded capabilities and deliver sparse point clouds. In this research, we reformulate the problem of monocular-SLAM to facilitate an FPGA/CUDA implementation, suitable for embedded applications, real-time processing and dense point cloud estimations.

1.3 Research question

Considering a single RGB camera, what approach could deliver a high point density and high performance for embedded applications in real time? Previous works reach high accuracy for the camera pose estimation but they limit the processing speed, embedded capabilities and in addition, they deliver clouds of scattered points. In order to achieve high performance in embedded applications, a new SLAM formulation that complies with the embedded architectures such as FPGA or CUDA is required. On the other hand, to increase the density of the point cloud, a new high density feature matching algorithm is required.

1.4 Hypothesis

It is possible to achieve high efficiency for embedded applications by increasing the density of the point cloud map (and therefore the 3D map density and overall positioning and mapping) by reformulating the tracking/matching features process by matching a parallel algorithm with an embedded architecture, such as FPGA or CUDA.

1.5 Objectives

Main objective: to develop a new monocular-parallel-SLAM algorithm that delivers high performance and high density of 3D points for real-time embedded applications.

Particular objectives:

- Propose a highly parallelizable algorithm formulation that achieves efficient hardware utilization for an FPGA/CUDA implementation.
- Reach dense mapping, superior to previous work.
- Validate the algorithm in an FPGA-based smart camera for monocular-SLAM.

1.6 Methodology

In order to estimate the camera pose (localization), we propose a new algorithm for estimating the pose of the camera in which the pixel displacements between frames are used as linear/dependent parameters for the the estimation of the camera pose, for what can be estimated the camera pose without iterative behavior and without geometric constraints and this makes possible the parallelism under FPGA/CUDA architectures. For the point cloud (mapping): we propose a new depth-from-motion algorithm based on a pixel-parallel/window-parallel configuration (similar to the stereo matching problem). This makes it possible to deliver dense depth maps and, therefore, improves the point cloud density.

To validate our hypothesis, we carried out accuracy measurement (localization), performed in MatLab using benchmark datasets and real world scenarios. Further, we carried out measurement of density (mapping), in MatLab, using small workspaces. In both cases, promising results are obtained. Experimental results demonstrate that it is possible to obtain accurate camera pose estimations. Compared to previous monocular systems, we are ranked as the 5th place in the KITTI benchmark suite, with higher processing speed (we are the faster algorithm in the benchmark) and more than $\times 10$ the point cloud density from most previous algorithms.

1.7 Contributions

Along this research work several contributions were published. We summarize them as follows. A novel monocular-SLAM formulation suitable for embedded applications [2]. A new feature tracking (pixel tracking) algorithm that provides a high density of 2D point correspondences [1]. A new feature extraction algorithm that achieves a simple FPGA implementation and that offers a high density of 2D feature points per frame [6]. A new FPGA architecture for pixel tracking (feature matching) with straightforward FPGA implementation and a high density of 2D point correspondences [5]. A novel approach to camera pose estimation suitable for embedded systems [4]. And finally, a new hardware architecture for embedded visual odometry, and a new hardware architecture for embedded monocular depth estimation.

1.8 Organization of the thesis

The thesis is structured as follows. In **Chapter** 2 the background about monocular-SLAM is presented while **Chapter** 3 a review of the current state of the art, performance and limitations are discussed. **Chapter** 4 presents the proposed algorithms, describes their properties, and we discuss about how these algorithms fulfil with our general and particular objectives. In **Chapters** 5 and 6 the implementation results using CUDA and FPGA respectively are presented and the performance of the proposed algorithms with previous works are compared. Finally, in **Chapter** 7 the conclusions and possible improvements are discussed.

Chapter 2

Monocular-SLAM: traditional formulation

The aim of this chapter is to provide a introduction of monocular SLAM. All parts of the traditional formulation are discussed in detail and are complemented with graphic explanations. First, the visual features (blobs, corners, edges, etc.) have to be extracted and tracked for at least two different frames form a video sequence. Then, the techniques based on geometry that use the projective geometry between 3D points of the scene and their projections in the plane of the image (visual features), are used to compute the camera pose (localization) and the structure of the 3D scene (mapping) simultaneously. In **Fig.** 2.1, the most popular monocular-SLAM formulation is shown [11, 39]. It consists of six different steps: initialization, data association, pose estimation, map construction, refinement and loop closure. At the beginning, there is not *a priori* information about the camera pose or the 3D scene structure so, the initialization step establishes an initial 3D map and the first camera poses in the system. Then, when a new frame is available, the data association deliver geometrical correspondences between the previous frames (at least one) and the current frame. The pose estimation step uses the previous camera poses and the geometrical correspondences previously computed in order to estimate the pose for the current frame. This pose is used to establish associations with the 3D map (map construction step). Then, in order to ensure the coherency of the map, reduce errors and remove outliers, a refinement step and a loop closure step continuously optimizes the map and the camera poses simultaneously.

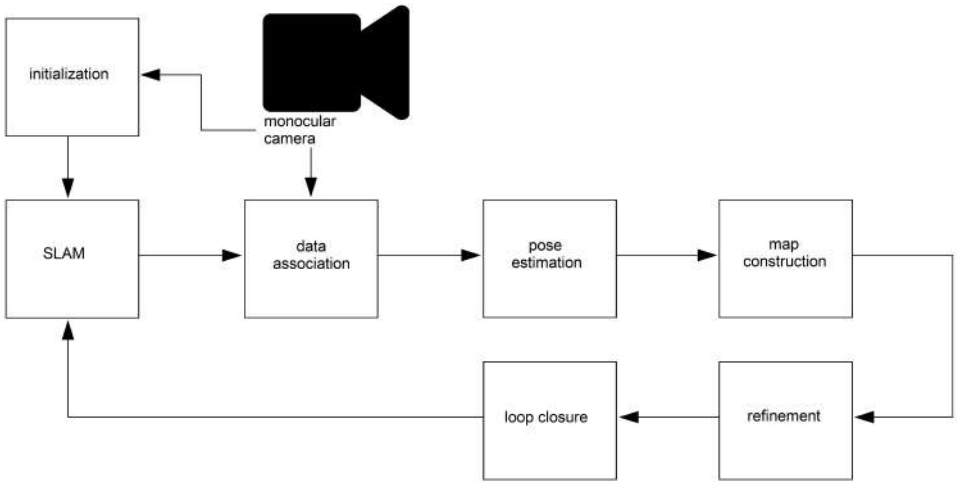


Figure 2.1. Monocular-SLAM: traditional formulation. First, visual features have to be extracted and tracked for at least two consecutive frames form a video sequence. Then, geometry-based techniques are used to compute the camera pose and the structure of the 3D scene simultaneously. In order to ensure the coherency of the map, refinement and loop closure steps should be computed as post-processing step.

2.1 Initialization

To compute the camera pose and the scene structure it is necessary to obtain/estimate the depth in the scene which can be estimated through temporal stereoscopy. In the traditional monocular-SLAM formulation these problems have to be solved in the initialization step. Then, the SLAM problem can be solved by expanding the initial map, while keeping track of the initial camera pose in the map. In early monocular-SLAM systems, such as in MonoSLAM [35], the user has to initialize the SLAM system by keying in the distance separating the camera from the square. However, in recent work, the researches adopted the methods developed by Longuet-Higgins & H Christopher [81] to simultaneously recover the camera pose and the 3D scene structure. In order to recover the scene structure and the camera pose Higgins & Christopher proposed algebraically eliminating the depth from the problem, obtaining the Essential and the Homography matrices. However, the elimination of the depth involves several limitations on the recovered data: on one hand since, the real camera motion cannot be recovered, it is computed in an unknown scale. On the other hand, because of the motion vector between the two views defines the baseline used to triangulate the visual features extracted from these views, the loss of scale is propagated to the recovered data, as a results the scale in the 3D map is unknown.

In **Fig.** 2.2 the flowchart of the generic initialization step which is based on the Longuet-Higgins & H Christopher algorithm is shown. First, visual features have to be extracted and tracked across at least two different viewpoints from the same scene (data association). In order to avoid degenerate matrices within the essential matrix estimation a proper feature matching distribution is necessary. So, any feature matching has to be validated using a key frame validation tread. Given feature matches for at least two frames, those can be used to estimate the Fundamental matrix using a robust model fitting method (RANSAC or MLESAC [133]). The estimated Fundamental matrix is then decomposed as described in [57] into an initial scene structure and initial camera poses. To minimize degenerate cases, a random depth initialization assigns depth values with large variance and then, this random depth has to be updated iteratively over subsequent frames until the depth variance converges. In the following subsections, details about the Fundamental matrix estimation, camera pose estimation and triangulation algorithms which are implemented inside the initialization step are given. For the data association step, please see **Section** 2.2.

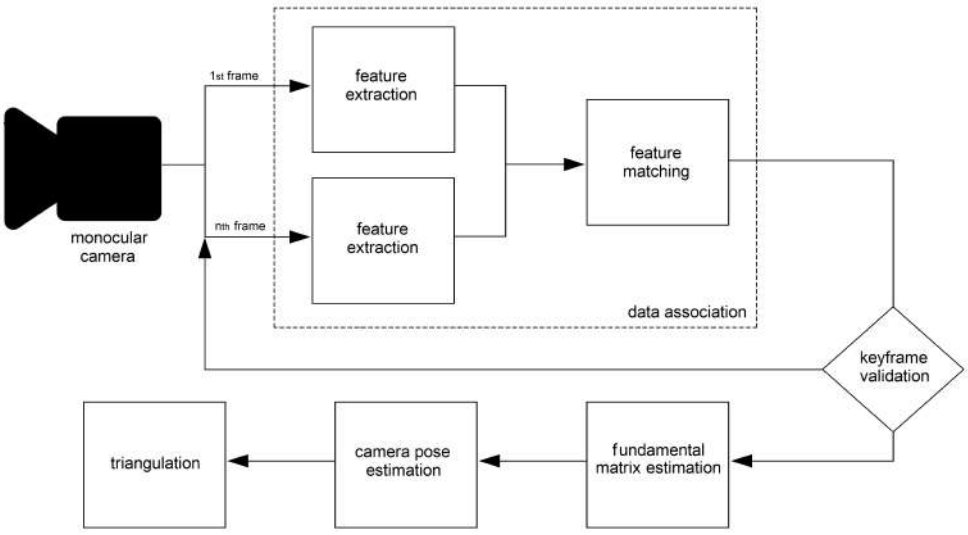


Figure 2.2. Monocular-SLAM initialization step. The first frame captured by the camera is set as the first keyframe. Then, subsequent frames (at least one) are processed by establishing 2D-2D data associations. These associations are used to estimate a Homography or a Fundamental matrix using a robust model fitting method. Finally, the estimated Homography or the Fundamental matrix is then decomposed into an initial scene structure and initial camera poses.

2.1.1 Keyframe validation

Triangulated 3D points are determined by the of the rays projected backwards from 2-D image correspondences of at least two image frames. In perfect conditions, these rays would intersect in a single 3D point. However, because of image noise, the camera model, the calibration errors, and the feature matching uncertainty, they never intersect. Therefore, the point at a minimal distance, in the least-squares sense, from all intersecting rays can be taken as an estimate of the 3D point position. In this scenario the standard deviation of the distances of the triangulated 3D point from all rays gives an idea of the quality of the 3D point. Then, three-dimensional points with large uncertainty will be thrown out. This happens especially when frames are taken at very nearby intervals compared with the distance to the scene points. When this occurs, the 3D points exhibit very large uncertainty. One way to avoid uncertainty consists of skipping frames until the average uncertainty of the 3D points decreases below a certain threshold. The selected frames are called keyframes, while the estimation process is called validation of keyframes. For the mathematical formulation, a simple metric could be the average distance for the matching process, where the mean distance d between $q\{i\}$ and $g\{i\}$ (the feature matching for two different viewpoints of the same scene) is computed as shown in **Eq.** 2.1; where n is the feature matching size. Then, the mean distance d is used to evaluate the robustness for the estimation of the camera pose. This process is illustrated in **Eq.** 2.2; where σ is a threshold value defined by the user.

$$d = \frac{1}{n}\sum_{i=1}^{n} |q\{i\} - g\{i\}| \tag{2.1}$$

$$keyframe = \begin{cases} 1 & if\ d > \sigma \\ 0 & otherwise \end{cases} \tag{2.2}$$

2.1.2 Fundamental matrix estimation

Let $q_n = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$, $g_n = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$ 2D-2D data associations for two different viewpoints from the same scene (**Section** 2.2), a matrix A can be estimated as as illustrated in **Eq.** 2.3. Then, let $[U\ D\ V]$ be the singular value decomposition (**SVD**) of A, the fundamental matrix F is computed as shown in **Eq.** 2.4.

$$A = \begin{pmatrix} q(x_1)g(x_1) & q(y_1)g(x_1) & g(x_1) & q(x_1)g(y_1) & g(y_1) & q(x_1) & q(y_1) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ q(x_n)g(x_2) & q(y_n)g(x_n) & g(x_n) & q(x_n)g(y_n) & g(y_n) & q(x_n) & q(y_n) & 1 \end{pmatrix} \tag{2.3}$$

$$F = U \cdot \mathbf{diag}([D(1,1)\ D(2,2)]\ 0) \cdot V' \tag{2.4}$$

2.1.3 Camera pose estimation

Let F be the fundamental matrix, computed by **Eq.** 2.4, the essential matrix E can be estimated as $E = K'FK$, where K is the calibration matrix for the imager [18]. To estimate the initial camera pose, the Least Squares algorithm [57] estimates the translation $\mathbf{t}$ and rotation $\mathbf{R}$ that minimizes the sum of the squared re-projection error, as shown in **Eq.** 2.5, where $q\{n\}$ and $g\{n\}$ are feature matching for two different viewpoints from the same scene (**Section** 2.2) while E is the esential matrix. To solve **Eq.** 2.5 first, the essential matrix E has to be discomposed via singular value decomposition, $[U\ D\ V] = \mathbf{SVD}(E)$. Then, given $e = (D(1,1) + D(2,2))/2$, and set $D(1,1) = e$, $D(2,2) = e$, $D(3,3) = 0$, the E' matrix ($E' = U \cdot D \cdot V'$) has to be discomposed via singular value decomposition, $[U_2\ D_2\ V_2] = \mathbf{SVD}(E')$. Finally, the translation vector t and the rotation matrix R are computed as shown in **Eq.** 2.6 and **Eq.** 2.7, respectively. In practice, there are four possible solutions: when triangulating $q\{n\}$ and $g\{n\}$, the correct solution is such it minimizes the outliers within the 3D reconstruction.

$$E(\mathbf{R},\mathbf{t}) = \frac{1}{n}\sum_{i=1}^{n} \parallel q\{i\} - \mathbf{R}g\{i\} - \mathbf{t} \parallel^2 \tag{2.5}$$

$$R = U_2 \cdot W \cdot V_2' \tag{2.6}$$

$$t = U_2 \cdot Z \cdot U_2' \tag{2.7}$$

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.8}$$

2.1.4 Triangulation

Let R, t be the rotation matrix and the translation vector, two camera matrices P_1, P_2 are defined as shown in **Eq.** 2.9 and 2.10. c_1, c_2 are defined as $-R_0/t_0$ and $-R/t$, respectively; while, α_1, α_2 are defined as $\alpha_1 = R/q\{n\}$, $\alpha_2 = R/g\{n\}$, where $q\{n\}, g\{n\}$ are 2D-2D data associations for two different viewpoints from the same scene (**Section** 2.2). Finally, $q\{n\}, g\{n\}$ points are tringulated as shown in **Eq.** 2.11 and 2.12.

$$P_1 = [R_0, t_0] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

$$P_2 = [R, t] \tag{2.10}$$

$$\beta = [c_1 - c_2]' \cdot [c_1 - c_2] / [c_1 - c_2]' \cdot [\alpha_1 - \alpha_2] \tag{2.11}$$

$$p = (c_1 + \beta(1) \cdot \alpha_1 + c_2 + \beta(2) \cdot \alpha_2)/2 \tag{2.12}$$

2.2 Data association

Given the monocular-SLAM formulation, the data association step often consists of two different algorithms implemented sequentially. These algorithms are feature extraction and feature matching. On the other hand, three different association types can be estimated, 2D-2D, 2D-3D, and 3D-3D:

- In the 2D-2D association, the 2D feature's location in an I_2 image is sought, given its 2D position in a previously acquired I_1 image. Depending on the type of information available, 2D-2D correspondences can be established in two ways: when a map is not available and the camera transformation between the two frames and the scene structure is not available (i.e. during system initialization), 2D-2D data association is established through a search window that surrounds the feature's location from I_1 in I_2. When the transformation related to I_1 and I_2 is known (i.e. the camera pose is estimated successfully), the 2D-2D data correspondences are established through the epipolar geometry, where a feature in I_1 is mapped to a line in I_2, and the two dimensional search window collapses to a one dimensional search along a line. The

latter case often occurs when the system attempts to triangulate 2D features in 3D features during generation of the map. In both methods, the visual features must be extracted and then each feature have to be associated with a visual descriptor, which can be used to provide a quantitative measure of similarity to other features. The descriptor similarity measure varies with the type of descriptors used; for example, for a local patch of pixels, it is typical to use the Sum of Squared Difference (SSD), or a Zero-Mean SSD score (ZMSSD) to increase robustness against changes in ilumination. For higher order feature descriptors such as, SIFT [82], or SURF [15], the L1-norm, L2-norm, or Hamming distances may be used; however, establishing matches using these measures is computationally intensive and may, if not carefully applied, degrade real-time performance. In practice, it is the binary descriptors that provide the best tradeoff between robustness and computationally requirements.

- In the 3D-2D data association, the pose of the camera and the 3D structure are known, and the aim is to estimate correspondences between the 3D features and their 2D projection in a newly acquired frame, without the knowledge of the new camera pose. This type of data association is typically used during the pose estimation phase of the monocular-SLAM formulation. To solve this problem, the previous camera poses are exploited in order to yield a hypothesis about the new camera pose and, consequently, project the 3D features onto that frame. 3D-2D data association then proceeds similarly to 2D-2D feature matching, by defining a search window surrounding the projected location of the 3D features and searching for matching feature descriptors.

- 3D-3D data association is typically employed to estimate and correct accumulated drift along loops: when a loop closure is detected, 3D feature descriptors are used, visible at both ends of the loop, to establish matches among features that are then exploited to yield a similarity transform between the frames at both ends of the loop.

2.2.1 Visual feature extraction

For the feature extraction step, one algorithm often used in by the traditional monocular-SLAM formulation is the the Harris & Stephens corners detection algorithm [56]. Given an input image $f(x, y)$, horizontal and vertical gradients are given by: $G_x(x, y) = f(x, y) \bullet g_x$, $G_y(x, y) = f(x, y) \bullet g_y$, where the operation $f(x, y) \bullet g$ denotes the 2D spatial convolution between an input image $f(x, y)$ and a fixed convolution kernel g, see **Eq.** 2.13.

Given the image gradients $(G_x(x,y), G_y(x,y)$, image derivatives $(A(x,y), B(x,y), C(x,y))$ are computed as $A(x,y) = G_x(x,y) * G_x(x,y),\ B(x,y) = G_y(x,y) * G_y(x,y),\ C(x,y) = G_x(x,y) * G_y(x,y)$. Then, a Gaussian filtering has to be applied on the image derivatives $(A(x,y),\ B(x,y),\ C(x,y))$ in order to reduce noise and removing fine-scale structures that affect the performance of the corner response. This process is defined as $A'(x,y) = A(x,y) \bullet G,\ B'(x,y) = B(x,y) \bullet G,\ C'(x,y) = C(x,y) \bullet G$, where the operator $\bullet$ denotes the 2D spatial convolution between an input image $(A(x,y), B(x,y), C(x,y))$ and a fixed convolution kernel G. The convolution kernel has to be defined by the discrete gaussian distibution, an example is shown in **Eq.** 2.14. Using the filtered image derivatives, the corner metric response is computed as in **Eq.** 2.15. Finally, using the corner metric response, the corner detection process is computed as shown in **Eq.** 2.16-2.17 where the operation $m' \circ M$ denotes the matrix composition between patches in the corner response image $(m' = m(x-1:x+1, y-1:y+1))$ and the matrix M (suppression matrix), i.e., $a(1,1) = m'(1,1) * M(1,1)$, $a(1,2) = m'(1,2) * M(1,2) \ldots\ a(3,3) = m(3,3) * M(3,3)$. This process is called non-maxima suppression step and its objective is to remove noise pixels detected as corners and retain only one point/pixel at each corner. Finally, a threshold (σ) has to be applied on $m(x,y)$ (the corner metric image), delivering ones at corner points retained after the non-maxima suppression step and zero otherwise, see **Eq.** 2.18.

$$g_x = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, \qquad g_y = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}, \tag{2.13}$$

$$G = \begin{pmatrix} 0.0178 & 0.0306 & 0.0367 & 0.0306 & 0.0178 \\ 0.0306 & 0.0525 & 0.0629 & 0.0525 & 0.0306 \\ 0.0367 & 0.0629 & 0.0753 & 0.0629 & 0.0367 \\ 0.0306 & 0.0525 & 0.0629 & 0.0525 & 0.0306 \\ 0.0178 & 0.0306 & 0.0367 & 0.0306 & 0.0178 \end{pmatrix} \tag{2.14}$$

$$m(x,y) = A'(x,y) \times B'(x,y) - C'(x,y)^2 - 0.04 \times (A'(x,y) + B'(x,y))^2 \tag{2.15}$$

$$a = m \circ M, \qquad b(x,y) = max(a) \tag{2.16}$$

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix} \tag{2.17}$$

$$h(x,y) = \begin{cases} 1 & if\ \sigma < m(x,y) > b(x,y) \\ 0 & otherwise \end{cases} \tag{2.18}$$

2.2.2 2D-2D data association (feature matching)

One feature matching algorithm often used by the monocular-SLAM formulation is BRIEF [26]. In this case, Let $u_i(x,y)$ be the feature points from an image I_1 while $v_i(x,y)$ be the feature points from an image I_2, each feature point have to be evaluated by a binary test defined as shown in **Eq.** 2.19 and 2.20,where, the p_a, p_b variables are defined as $u_i(x+\mu_x, y+\mu_y)$ for I_1 and $v_i(x+\mu_x, y+\mu_y)$ for I_2, where μ_x, μ_y is defined as a random number less than S. Let S be the size of a patch $(S \times S)$ whose center is x, y. Then, let n be the maximum number of binary tests, the **BRIEF** binary descriptor is defined as shown in **Eq.** 2.20, giving as result a binary vector of dimension n. Finally, let $\mathbf{BRIEF}_{(n)}(u_i(x,y))$, $\mathbf{BRIEF}_{(n)}(v_i(x,y))$ be the **BRIEF** descriptors for all points in I_1 and I_2, these can be associated/matched using a ratio distance based on the Hamming distances **H**, as illustrated in **Eq.** 2.23; where γ_1, γ_1 are the first and second minimum Hamming distances, respectively while δ is a threshold value defined by the user (typically $\delta = 1$) and H_n represent the visual descriptor associations between u_i and v_i.

$$\lambda = \begin{cases} 1 & if\ p_a < p_b \\ 0 & otherwise \end{cases} \tag{2.19}$$

$$\mathbf{BRIEF}_{(n)} = \sum_{i=1}^{n} \lambda_n \tag{2.20}$$

$$H_n = \begin{cases} \mathbf{arg\ min}_{\mathrm{r}} & if\ \gamma_1 - \gamma_1 > \delta \\ 0 & otherwise \end{cases} \tag{2.21}$$

$$\gamma_r = \mathbf{H}(\mathbf{BRIEF}_{(n)}(u_i(x,y)), \mathbf{BRIEF}_{(n)}(v_r(x,y))) \tag{2.22}$$

$$\mathbf{BRIEF}_{(n)}(u_i(x,y)) \sim \mathbf{BRIEF}_{(H_{(n)})}(v_i(x,y)) \tag{2.23}$$

2.3 Pose estimation

The flow diagram of the generic pose estimation step is shown in **Fig.** 2.3. Since data association can be computationally expensive, most monocular-SLAM systems assume, for the pose of each new frame, *a prior*, which guides and limits the amount of work required for data association. Estimating this *prior* before is usually the first task in pose estimation: data association between the two frames is not known yet and one seeks to estimate *a prior* on the pose of the second frame (**R,t**), given previously estimated poses. To solve this problem, most systems employ a constant velocity motion model that involves a smooth movement of the camera and uses the pose changes in the two frames followed previously to estimate the previous one for the current frame. Then, the pose of the *prior* frame is used to guide the data association procedure in several ways. The *prior* helps to determine a potentially visible set of map features in the current frame, which reduces the computational expense of blindly projecting the entire map. Furthermore, it helps establish a location of the estimated characteristic in the current frame, such that feature matching takes place in small search regions, instead of across the entire image. Finally, the *prior* serves as a starting point for the minimization procedure, which refines the camera pose.

2.3.1 Constant Velocity Motion Model (CVMM)

The Kalman filter provides an estimate of the state of a discrete-time process defined in the form of a linear dynamical system [25]: $x(t+1) = F \times x(t) + w(t)$, with noise from the process $w(t)$. The Kalman filter operates by observing all or some of the state variables, defined by the observation system $y(t) = H \times x(t) + v(t)$, where $v(t)$ represents measurement noise. It is assumed that the process and measurement noise is independent of each other, white, with normal probability distributions: $w \sim BN(0, Q)$ and $v \sim BN(0, R)$. In practice, the state and observation equations are constructed to define the process and relate the corresponding measurements, and the noise variations are established according to the characteristics of the process and the noise measurements. Then, the model can be plugged into a generic form of the Kalman filter [25], which carries out the resulting algebra to obtain a state estimate for any instance. In previous works, the Kalman filtering has been employed in several fields of image processing such as video restoration, camera motion modeling, image stabilization, etc.

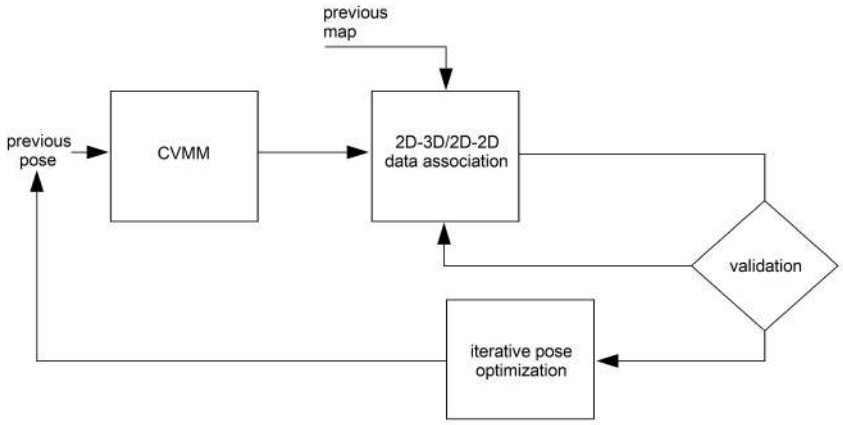


Figure 2.3. Monocular-SLAM: the pose estimation step. First, *a prior*, which guides and limits the amount of work required for data association is computed. To address this problem, most systems employ a constant velocity motion model that assumes a smooth camera motion and use the pose changes across the two previously tracked frames to estimate the *prior* for the current frame. Then, the pose of the *prior* frame is used to guide 2D-3D data association procedure. Finally, a minimization procedure refines the camera pose.

For the Constant Velocity Motion Model (CVMM), let W be the coordinate system of fixed frames in the world, and R, the coordinate system of fixed frames with respect to the camera. A non-minimal representation of 3D orientation and a quaternion is used to define the position state (see **Eq.** 2.24). Then, to model the system, a constant angular velocity model can be used. This means the assumption that the camera moves at a constant velocity over all the time, but that the statistical model of its motion in a time step expects indeterminated accelerations with a Gaussian profile. The implication of this profile is that it imposes a certain smoothness in the camera motion. This model is subtly effective and gives the whole system, robustness even when visual measurements are sparse. So, to model the velocity of the camera, the position state vector with the velocity terms to form the state vector is defined as shown in **Eq.** 2.25; were v^W is the linear velocity and ω^{WR} the angular velocity; where the angular velocity is a vector whose orientation denotes the axis of rotation and whose magnitude the rate of rotation in radians per second (note that the redundancy in the quaternion part of the state vector means that a normalization at each step of the EKF is necesary to ensure that each filtering step results in a true quaternion satisfying $q_0^2 + q_x^2 + q_y^2 + q_z^2 = 1$; this normalization comes with the corresponding Jacobian calculation affecting the covariance matrix.).

Considering that in each time step, unknown acceleration and angular acceleration processes of zero mean and Gaussian distribution can cause an impulse of velocity and angular velocity, as shown in **Eq.** 2.26; the covariance matrix of the noise vector n is diagonal and represents uncorrelated noise in all linear and rotational components. Then, the state update can be modeled with **Eq.** 2.27; where the notation $q(\omega^W + \Omega^W)\Delta_t$ denotes the Quaternion defined by the angle-axis rotation vector $\omega^W + \Omega^W$. Finally, in order to guarantee EKF robustness against noise, the new state estimate $f_v(x_v, u)$ often considers the increase in state uncertainty (process noise covariance) Q_v for the camera after this motion, where the Q_v value is computed by the Jacobian calculation (see **Eq.** 2.28); where P_n is the covariance of noise vector n (note that the EKF implementation also requires calculation of the Jacobian $\frac{\partial f_v}{\partial x_v}$). Therefore, the rate of growth of uncertainty in this motion model is determined by the size of P_n, and setting these parameters to small or large values defines the smoothness of the expected motion. With a small P_n, smooth motion with small accelerations is expected, and it would be well placed to track motions of this type, but will not be able to cope with sudden rapid movements. On the other hand, high P_n means that the uncertainty in the system increases significantly at each time step, and while this gives the ability to cope with rapid accelerations the very large uncertainty means that a lot of good measurements have to be made at each time step to constrain estimates.

$$x_p = \begin{pmatrix} r^W \\ q^{WR} \end{pmatrix} = (x, y, z, q_0, q_x, q_y, q_z)^T \tag{2.24}$$

$$x_v = \begin{pmatrix} r^W \\ q^{WR} \\ v^W \\ \omega^W \end{pmatrix} \tag{2.25}$$

$$n = \begin{pmatrix} V^W \\ \Omega^W \end{pmatrix} = \begin{pmatrix} a^W \Delta_t \\ \alpha^W \Delta_t \end{pmatrix} \tag{2.26}$$

$$f_v = \begin{pmatrix} r^W_{new} \\ q^{WR}_{new} \\ v^W_{new} \\ \omega^{WR}_{new} \end{pmatrix} = \begin{pmatrix} r^W + (v^W + V^W)\Delta_t \\ q^{WR} \times q(\omega^W + \Omega^W)\Delta_t \\ v^W + V^W \\ \omega^W + \Omega^W \end{pmatrix} \tag{2.27}$$

$$Q_v = \frac{\partial f_v}{\partial n} P_n \frac{\partial f_v}{\partial n}^T \tag{2.28}$$

2.3.2 2D-3D/2D-2D data association (feature tracking)

Let $q\{i\}$ be visual features for the previous keyframe (they have to be extracted and associated with a visual descriptor through the initialization step or through the feature matching step), the $g = \{i\}$ visual features in the current frame are tracked by searching a match in the current frame within a small search area around its position at the previous keyframe. This process is called feature tracking and is compared with feature matching (**Section** 2.2.2), feature tracking decreases the computational requirements because one hand, the extraction and description of characteristics are calculated only in one box instead of two and, on the other hand, because the tracking within small areas are less exhaustive than feature matching between two images.

For the mathematical formulation, let $q\{i\}$ be visual features for the previous keyframe, $g\{i\}$ (feature tracking for the current frame), can be obtained by minimizing any local similarity metric between the previous and the current frame, as shown in **Eq.** 2.29 and 2.30; where I_1, I_2 are the previous and the current frame, respectively. s is the search size at the current frame while r is the patch size. Finally, to validate the feature tracking step, visual features of the previous and the current keyframe ($(q\{i\}, g\{i\})$, respectively), should be associated with their corresponding X_i, Y_i, Z_i real world coordinates (2D-3D association) that can be obtained from the previous map. Then, using the camera pose computed by the CVMM algorithm, it is possible to validated the 2D-2D data associations obtained through the feature-tracking step $(q\{i\}, g\{i\})$ by triangulate $q\{i\}$ and $g\{i\}$ (**Section** 2.1.4) and comparing the results with X_i, Y_i, Z_i. In the case of finding enough correspondences, i.e., the motion model is not violated, the camera pose and 3D map are optimized (as shown in **Section** 2.3.3). In the case of not finding enough correspondences, feature matching (**Section** 2.2.2) has to be computed and thereafter, the camera pose and 3D map have to be optimized.

$$\mathrm{SSD}(a,b) = \sum_{a=-s,b=-s}^{a=s,b=s} \sum_{u=-r,v=-r}^{u=r,v=r} (I_1(q\{i\} + (u,v)) - I_2(q\{i\} + (u,v) + (a,b)))^2 \tag{2.29}$$

$$g\{i\} = \textbf{arg min}_{\text{a,b}}(\text{SSD}(a, b)) \tag{2.30}$$

2.3.3 Iterative pose optimization

Let r^{W}_{new} and q^{WR}_{new} be the camera pose for the current frame (these are obtained through the CVMM algorithm, **Section** 2.3.1, and correspond with the transition vector t and the corresponding quaternion which can be converted in a rotation matrix R, respectively), and let $q\{i\}, g\{i\} \sim X_i, Y_i, Z_i$ be 2D-2D/2D-3D data associations (feature matching/feature tracking), traditional monocular-SLAM formulations refines the camera pose R, t and the 3D local map by minimizing the re-projection error of features from the map (X_i, Y_i, Z_i) over the frame's prior pose (r^{W}_{new}, q^{WR}_{new}). The re-projection error is formulated as the distance in pixels between a 3D feature projected in a frame, and its corresponding 2-D position in the image. In order to obtain robustness against outliers, the minimization takes place over an objective function that penalizes the features with big errors. Therefore, the problem of optimization of camera pose is defined as shown in **Eq.** 2.31; where T_k is a minimally represented Lie group [57] of either (X_i, Y_i, Z_i) feature or R, t camera pose. $Ob_i(.)$ is an objective function and e_i is the error defined through the data association for each matching feature $q\{i\}, g\{i\}$ in the image. Finally, the system decides wheter the new frame should be marked as a keyframe or not. Decisive criteria can be classified as significant changes in pose or significant changes in the appearance of the scene; a decision is usually made through a weighted combination of both criteria; i.e., the current frame is flagged as a keyframe only if a significant change in the camera pose measurements (rotation and/or translation) occurs, and when there are a significant number of 2D features that are not observed in the current map.

$$T_k = \textbf{arg min}_{\text{T}_\text{k}} \sum_i Ob_i(e_i) \tag{2.31}$$

2.4 Map construction

The step of the construction of the map is responsible for generating a representation of the unexplored, and recently observed environment. Typically, the map construction step represents the world as a sparse cloud of points. The flow diagram of the construction step of the generic map is shown in **Fig.** 2.4. In general, the different viewpoints of an

unexplored scene are registered with their corresponding camera poses through the pose tracking step (**Section** 2.3.2). The map construction step then re-establishes the data association between the new keyframe and a set of keyframes surrounding it, looking for matches. Then, triangulate 2D feature points into 3D features; it also keeps track of their 3D coordinates and expands the map within what is known to as a metric representation of the scene. In order to guarantee consistency within the map, the global map is optimized each time a keyframe is added.

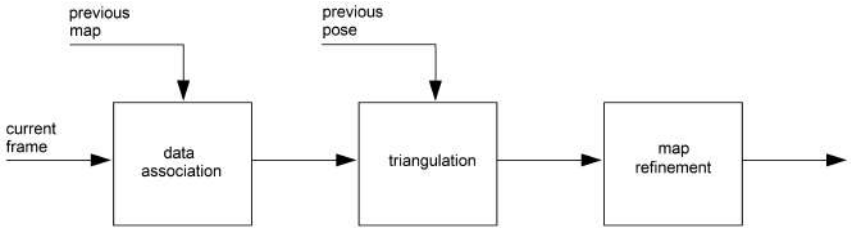


Figure 2.4. Monocular-SLAM: the map construction step. First, the map construction step gets data association between the new keyframe and a set of keyframes surrounding it, looking for matches between the previous map and the current keyframe. Then, 2D feature points for the current kayframe are triangulated into 3D features using the camera poses previously computed. Finally the previous map is expanded via optimization techniques.

2.4.1 Data association

In previous monocular-SLAM formulations, map points have to be created by triangulating feature points in different keyframes. For that, a popular solution consists of triangulating points between adjacent keyframes since parallax level is low and this decreases the computational requirements for the data association process [28, 35, 103]. However, recent works [94] have demonstrated that triangulating feature feature points between N keyframes in the covisibility graph provides higher mapping density than the approach based on adjacent frames. For the mathematical formulation, it is necessary to correspond points between the current keyframe and N neighboring keyframes. For each feature point in the current keyframe, it is necessary to search a match with another feature in other keyframe simply by comparing the descriptors, as illustrated in **Eq.** 2.32 and 2.33; where I is the current keyframe while I_N are N adjacent keyframes in the covisibility graph that share most feature points in I. $x_1\{i\}$ are feature points in I, $x_N\{i\}$) are their 2D-2D data associations for the N neighbor keyframes and s is the search size while r is the patch size.

$$\mathrm{SSD}_N(a,b) = \sum_{a=-s,b=-s}^{a=s,b=s} \sum_{u=-r,v=-r}^{u=r,v=r} (I(x_1\{i\} + (u,v)) - I_N(x_1\{i\} + (u,v) + (a,b)))^2 \quad (2.32)$$

$$x_N\{i\} = \mathbf{arg\ min}_{\mathrm{a,b}}(\mathrm{SSD}_N(a,b)) \quad (2.33)$$

2.4.2 Triangulation

First, two camera matrices P_1, P_2 are defined as shown in the **Eq.** 2.34 and 2.36; where R_1, t_1 and R_2, t_2 are camera poses for the current and the previous keyframe, respectively (these are obtained by the pose estimation step, **Section** 2.3). Then, let $q\{n\}, g\{n\}$ be 2D-2D data associations between the current and the previous keyframe, the current map is obtained via linear triangulation, as described in **Section** 2.1.4).

$$P_1 = [R_1, t_1] \quad (2.34)$$

$$P_2 = [R_2, t_2] \quad (2.35)$$

2.4.3 Map refinement

Due to noise/outliers in the data association step, the projection of the rays from two associated features probably not intersect in the 3D space and this generates inconsistencies within the 3D map. To obtain resilience against outliers and to obtain better accuracy, the triangulation is typically performed over features associated across more than two views [41, 94]. This process is called multiple view triangulation and is illustrated in **Fig.** 2.5). For the mathematical formulation, given N images of i fixed 3D points ($x_{N,i} = P_N \cdot X_i$), the problem is defined as estimate N projection matrices P_N and i 3D points X_i from the N, i correspondences $x_{N,i}$. In order to solve this problem, monocular-SLAM formulation uses the Levenberg-Marquardt algorithm (**Eq.** 2.36); where the variable X_i is solved by minimizing the re-projection error for 2D-2D data associations ($x_{N,i}$) at N different viewpoints from the same scene and P_N camera matrices (obtained by the pose estimation step, described in **Section** 2.3).

$$E(P, X) = \sum_{a=1,b=1}^{a=N,b=i} D(x_{a,b}, P_a \cdot X_b)^2 \tag{2.36}$$

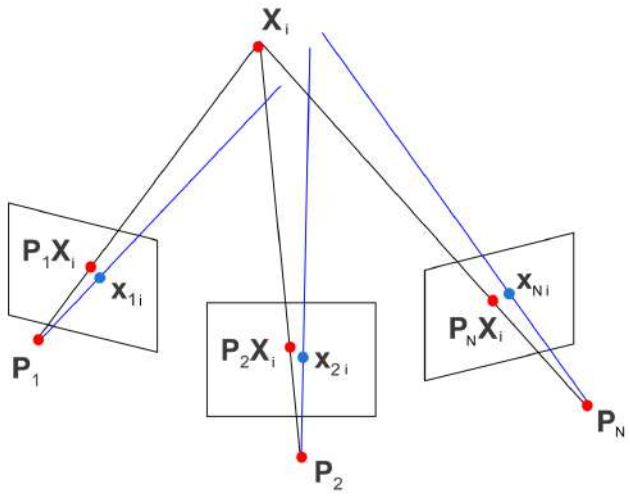


Figure 2.5. Multiple view triangulation. Given N images of i fixed 3D points ($x_{N,i} = P_N \cdot X_i$), the Levenberg-Marquardt algorithm is applied in order to estimate N projection matrices P_N and i 3D points X_i from the N, i correspondences $x_{N,i}$ that minimizes the re-projection error.

2.5 Refinement

The refinement step is responsible for optimizing both the map and the camera poses. For that, the monocular-SLAM formulation typically uses bundle adjustment as an optimization technique. During map exploration, the new 3D features are triangulated according to the camera pose estimates; after some time, the deviation of the system manifests itself in wrong camera pose measurements, due to accumulated errors in previous camera poses that were used to expand the map. The refinement step continues by a establishing data association between the full set of keyframes in the map or a subset of keyframes and performing a global bundle adjustment (GBA) or a local bundle adjustment (LBA) respectively. Outlier features flagged from the optimization are then culled (removed from the map) while the values of the previously calculated camera poses are optimized simultaneously. To reduce the complexity of optimization, redundant keyframes are also culled. Bundle adjustment is the problem of refining a visual reconstruction and camera poses to produce a jointly optimal 3D structure and see parameter estimates (camera pose). This

mean the estimates of the parameters are found by minimizing some cost function that quantifies the error of fit of the model, and that the solution is simultaneously optimal with respect to the variations of the structure and the camera.

For the mathematical formulation of package adjustment, the cost function to be minimized is shown in the **Eq.** 2.37; where T_i is a pose estimate of keyframes and N is the number of keyframes on the map or a subset of the map. X_j corresponds to the 3D pose of a feature and S_i represents the set of 3D features observed in Keyframe i. Finally, $e(T_i, X_j)$ is the reprojection error of a feature X_j in a keyframe T_i, in which it is observed. In order to reduce the computational costs of bundle adjustment, several formulations represent the monocular SLAM map using a Euclidean map for LBA, and a topological map for pose graph optimization that explicitly distributes the accumulated drift along the entire map. Then, the original formulation of Package Adjustment is rewritten as shown in **Eq.** 2.38; where the optimization process takes place only in the keyframe that represents (T_i).

$$\textbf{arg min}_{\mathrm{T,X}} \sum_{i=1}^{N} \sum_{j \in S_i} Ob_j(e(T_i, X_j)) \tag{2.37}$$

$$\textbf{arg min}_{\mathrm{T}} \sum_{i=1}^{N} \sum_{j \in S_i} Ob_j(e(T_i, X_j)) \tag{2.38}$$

2.6 Loop closure

Global localization is required when the camera loses track of its position and is required to be located on a global map. Failure recovery and loop closure are considered a form of global localization. It is worth mentioning that loop closure and fault recovery revolve around the same problem and the solutions presented for any of them could be used for the other.

2.6.1 Relocalization

Either due to improper movement of the user, such as abrupt changes in the camera pose that result in motion blur, or due to observation a region without distinctive features, or for any other reason, the monocular-SLAM formulation may eventually fail. Therefore,

an essential key module for the usability of any monocular-SLAM system is its ability to recover correctly from such failures. To solve this problem, a relocalization algorithm should be implemented. So, let $q\{i\}$ be visual features for the previous keyframe, each visual feature associated with a visual descriptor and in which the camera pose estimation step was correct executed; $g\{i\}$ visual features and visual descriptors are extracted at the frame being processed, then, the system relocalization can be obtained by feature matching between $q\{i\}$ and $g\{i\}$. In order to evaluate the matching quality and therefore, guarantee a non-degenerated result for the camera pose estimation, a threshold operation has to be applied, using **Eq.** 2.39; where $\mathbf{BRIEF}_{(i)}(u_i(x,y)), \mathbf{BRIEF}_{(H_i)}(v_i(x,y))$ are the matched descriptors between $q\{i\}$ and $g\{i\}$ (see **Eq.** 2.23) and Φ is the quality of coincidence which is used to decide if the current frame can be located correctly, i.e., in case the match quality is larger than a threshold value ($\Phi > \sigma$), the camera pose can be obtained; otherwise the system remain as standby mode and process the following frames until the corresponding quality criteria is satisfied. In the last step (after the matching criterion is satisfied), let $q\{i\} \sim g\{i\}$ be 2D-2D data associations for the previous keyframe in which the camera pose estimation step it was correctly executed; the current camera pose and 3D reconstruction (mapping) can be estimated using the formulation presented above in **Sections** 2.3 and 2.4.2.

$$\Phi = \sum_{i=1}^{i=n} \mathbf{H}(\mathbf{BRIEF}_{(i)}(u_i(x,y)), \mathbf{BRIEF}_{(H_i)}(v_i(x,y))) \tag{2.39}$$

2.6.2 Loop closure

Since the traditional monocular-SLAM formulation consist of several optimization steps, it is prone to changes in camera pose estimates. As a result, returning to a certain position after an exploration phase may not generate the same measure of camera pose, as it did at the start of the run. The camera pose drift can also manifest itself in a map scale drift, which will eventually lead the system to erroneous measurements, and fatal failure. To solve this problem, some algorithms detect loop closures in an offline monocular-SLAM session, and optimize the loops track in an effort to correct the drift and the error in the camera pose and in all relevant map data that were created during the loop. For the mathematical formulation, similar to relocalization procedures, one way so solve the loop closure problem is by matched descriptors between two keyframes, using **Eq.** 2.39. However, different to relocalization, in loop closure the keyframes being matched are the

start frame and the current frame (in case of large maps with multiple loops the start frame is replaced by the first frame of each map node). Finally, after a loop closure is detected, all camera poses and features within the map are corrected. In all cases, there are three consecutive step that must be carried out:

1. The first step to correct a loop is to compute the transformation from the current keyframe camera coordinate system to the loop candidate one. This transformation is the only way to know the accumulated drift, which in monocular-SLAM can occur in seven degrees of freedom: three translations, three rotations and scale. Therefore it is necessary to compute a similarity $\mathbf{Sim}(3)$ transformation from current keyframe k to loop candidate l, see **Eq.** 2.40; where $s \in \mathbb{R}^+$ is the scale factor, $\mathbf{R} \in \mathbf{SO}(3)$ is a rotation matrix and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector. The computation of this transformation serves as geometrical verification, if it is successful it is necessary to correct the loop, otherwise the loop candidate is rejected.

$$S_{k,l} = \begin{pmatrix} s_{k,l}\mathbf{R}_{k,l} & \mathbf{t}_{k,l} \\ 0 & 1 \end{pmatrix} \tag{2.40}$$

2. All the poses are converted in their absolute transformation $\mathbf{SE}(3)$ while the keyframe of origin $T_{i,w}$are converted in a similarity $\mathbf{Sim}(3)$; where $S_{i,w}$ maintains the rotation and the translation and establishing the scale to 1. Then, the relative transformation $\Delta S_{i,j}$ is computed, between one pose and the next, closing the loop with the computed similarity transformation between the current keyframe and the loop keyframe $S_{k,l}$. The basis of this process is the minimization of the residual error $r_{i,j}$ between the poses $r_{i,w}$ and $r_{j,w}$ with respect to the constraint $\Delta S_{i,j}$ in the tangent space $\mathbf{Sim}(3)$ and in a minimal representation as illustrated in **Eq.** 2.41; where $log_{\mathbf{Sim}(3)}$ is $\mathbf{Sim}(3) \rightarrow \mathbf{sim}(3)$, that maps from the overparametrized representation of the transformation to the tangent space and $(\cdot)^V_{\mathbf{Sim}(3)}$ is $\mathbf{sim}(3) \rightarrow \mathbb{R}^7$ is an operator that maps from the tangent space to the minimal representation with the same elements as the degrees of freedom of the transformation. Initially all the residuals are zero, except the loop. Then, these residues are optimized to distribute this error along the graph. The cost function to minimize the error is defined in the **Eq.** 2.42; where $\bigwedge_{i,j}$ is the inverse covariance of the residual $\mathbf{r}_{i,j}$ and is established in the identity. Then, for each point x_j a keyframe of origin $T_{i,w}$ is associated and the point is mapped using the optimized using **Eq.** 2.43. The last step is to convert the corrected similarity transformations ($\mathbf{S}^{cor}_{i,w}$) back to 3D rigid body transformations

$T_{i,w}^{cor}$. To eliminate the scale factor in similarity and maintain rotation since it is not affected by the scale; the scale factor of similarity should be scaled again (see **Eq.** 2.44). Finally, the poses and points are optimized together with this seed.

$$r_{j,w} = (log_{\mathbf{Sim}(3)}(\Delta S_{i,j} \cdot S_{j,w} \cdot S_{i,j}^{-1}))_{\mathbf{sim}(3)}^{v} \tag{2.41}$$

$$\chi^2 = \sum_{i,j} \mathbf{r}_{i,j}^T \bigwedge_{i,j} \mathbf{r}_{i,j} \tag{2.42}$$

$$x_j^{cor} = (\mathbf{S}_{i,w}^{cor})^{-1} \cdot T_{i,w} \cdot x_j \tag{2.43}$$

$$\mathbf{S}_{i,w}^{cor} = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \rightarrow T_{i,w}^{cor} = \begin{pmatrix} s\mathbf{R} & \frac{1}{s}\mathbf{t} \\ 0 & 1 \end{pmatrix} \tag{2.44}$$

3. The last step in closing the loop is to inform the frontend about the closure of the loop and correct the location and velocity of the camera. First the relative transformation of the the current camera pose $T_{c,w}$ to the non-corrected pose of the current keyframe $T_{k,w}$ is computed, see **Eq.** 2.45. Then, the translation of $\Delta T_{c,k}$ should be scaled by the scale factor $s_{k,l}$ of the similarity transformation defined in the **Eq.** 2.46; then, by applying $T_{c,k}^{cor}$ to the corrected current keyframe pose $T_{k,w}^{cor}$ the corrected camera poses are recover, see **Eq.** 2.47. Finally the velocity motion for the CVMM algorithm, (**Section** 2.3.1) is corrected dividing the linear speed by $s_{k,l}$.

$$\Delta T_{c,k} = T_{c,w} \cdot T_{k,w}^{-1} \tag{2.45}$$

$$\Delta T_{c,k} = \begin{pmatrix} s\mathbf{R}_{c,k} & \mathbf{t}_{c,k} \\ 0 & 1 \end{pmatrix} \rightarrow \Delta T_{c,k}^{cor} = \begin{pmatrix} s\mathbf{R}_{c,k} & \frac{1}{s_{c,k}}\mathbf{t}_{c,k} \\ 0 & 1 \end{pmatrix} \tag{2.46}$$

$$T_{k,w}^{cor} = \Delta T_{c,w}^{cor} \cdot T_{k,w}^{cor} \tag{2.47}$$

2.7 Summary

In this chapter we have provided an overview of state of the art monocular-SLAM techniques. All parts of the traditional formulation were discussed in detail and were complemented with graphical explanations.

Chapter 3

Monocular-SLAM: a survey

This section details the previous monocular-SLAM systems in the literature. **Table** 3.1 lists all the monocular-SLAM systems that, to our knowledge, exist to date. In the following subsections, more details on all of these approaches are presented.

Table 3.1. Monocular-SLAM systems in the current literature.

Year	Name	Feature extraction	Feature matching	Optimization choice
2006	Real-time Localization and 3D Reconstruction [92]	Harris & Stephens	Local patch	Bundle Adjustment
2007	Parallel tracking and mapping for small AR workspaces [69]	FAST	Local patch	Tukey-biweight
2008	An Efficient Direct Approach to Visual SLAM [117]	Intensity gradient	Local patch	Gauss Newton
2010	Scale Drift-Aware Large Scale Monocular SLAM [125]	FAST	Local patch	Bundle Adjustment
2010	Live dense reconstruction with a single moving camera [96]	FAST	Local patch	Bundle Adjustment
2011	Dense Tracking and Mapping in Real-Time(DTAM) [97]	Intensity gradient	Local patch	Gauss Newton
2011	Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation [110]	Harris & Stephens and Cany	Local patch	Bundle Adjustment

2011	CD SLAM-continuous localization and mapping in a dynamic world [108]	SIFT	SIFT	Bundle Adjustment
2011	Online environment mapping [79]	Harris & Stephens	KLT trackers	Bundle Adjustment
2011	Homography-based planar mapping and tracking for mobile phones [105]	FAST	Local patch	Bundle Adjustment
2013	Robust monocular SLAM in Dynamic environments [127]	SIFT	SIFT	Bundle Adjustment
2013	Handling pure camera rotation in keyframe-based SLAM [106]	FAST	Local patch	Bundle Adjustment
2014	Efficient keyframe-based real-time camera tracking [38]	Harris & Stephens	KLT trackers	Bundle Adjustment
2014	SVO: Fast semi-direct monocular visual odometry [46]	FAST	Local patch	Gauss Newton
2014	LSD-SLAM: Large-scale direct monocular SLAM [41]	Intensity gradient	Local patch	Gauss-Newton
2014	DT-SLAM: deferred triangulation for robust SLAM [60]	FAST	Local patch	Bundle Adjustment
2014	Real-Time 6-DOF Monocular Visual SLAM in a Large Scale Environment [78]	FAST	BRIEF	Bundle Adjustment
2015	Robust large scale monocular Visual SLAM [24]	SURF	SURF	Bundle Adjustment
2015	ORB-SLAM: a versatile and accurate monocular SLAM system [95]	FAST	ORB	Bundle Adjustment
2015	DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence [33]	Intensity gradient	Local patch	Gauss-Newton
2016	Multi-level mapping: Real-time dense monocular SLAM [55]	Intensity gradient	Local patch	Bundle Adjustment

2016	Robust Keyframe-basedMonocular SLAM for Augmented Reality [80]	FAST	Homography	Bundle Adjustment
2017	CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction [129]	-	-	CNN
2017	ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras [94]	FAST	ORB	Bundle Adjustment
2017	PL-SLAM: Real-time monocular visual SLAM with points and lines [111]	FAST/Cany	ORB	Bundle Adjustment
2017	NID-SLAM: Robust Monocular SLAM using Normalised Information Distance [100]	Intensity gradient	NID	NID-based optimization
2018	Polarimetric Dense Monocular SLAM [141]	Intensity gradient	Polarimetric distance	Azimuth-based optimizations
2018	Loosely-Coupled Semi-Direct Monocular SLAM [74]	FAST	ORB	Bundle Adjustment
2018	Direct sparse odometry [43]	Intensity gradient	Local patch	Gauss-Newton
2018	Undeepvo: Monocular visual odometry through unsupervised deep learning [76]	-	-	CNN

Most previous works used as basis of their algorithms the traditional formulation presented in **Chapter** 2. Then they made modifications in some parts of the traditional formulation in order to improve the performance under specific application domains. So, in the following subsections we present a survey based on the processing steps of the traditional formulation and how the previous works modified these processing steps in order to fulfill with their specific constraints.

3.1 Initialization

Initially, PTAM [69] proposed using the five-point algorithm [98] to estimate and decompose a Fundamental matrix in a $SE(3)$ transformation that relates both initializing keyframes. The transformation is then used to triangulate a supposed non-planar initial scene. The initialization of PTAM was subsequently changed to a Homography estimate [45], where the scene is assumed to be composed of 2D planes. Because the 2D-2D matching process is performed through ZMSSD without warping the features, establishing correct matches is susceptible to both motion blur and significant changes in the appearance of the features as a result of camera rotations. Therefore, strict requirements on user's motion during initialization are required. The initial map generated is scaled, for example, the estimated translation between the first two keyframes corresponds to 0.1 units, before the structure of only BA takes place. SVO [106] adopted a Homography for initialization with the same procedure as PTAM. SVO extracts the FAST features and tracks them using KLT [132] (Kanade-Lucas-Tomasi feature tracker) across incoming frames. To avoid the need for a second input by the user, SVO monitors the median of the baseline distance of the features, tracked between the first keyframe and the current frame; and whenever this value reaches a certain threshold, sufficient parallax is assumed, and the Homography can be estimated.

DT-SLAM [60] does not have an explicit initialization phase; rather, it is integrated into its tracking module as an essential matrix estimation method. In LSD-SLAM [41], and later in DSO-SLAM [43], a scene depth randomly initialized from the first viewpoint, both systems use an initialization method that does not require two view geometry. i.e., the initialization step in LSD-SLAM [41] and DSO-SLAM [43] takes place on a single frame: pixels of interest (i.e., image locations that have high intensity gradients) in the first keyframe are given a random depth value with an associated large variance. This results in an initially erroneous 3D map. The pose estimation methods are then invoked to estimate the pose of newly incoming frames using the erroneous map, which in return results in erroneous pose estimates. However, as the system process more frames of the same scene, the originally erroneous depth map converges to a stable solution. The initialization is considered complete when the depth variance of the initial scene converges to a minimum. DPPTAM [33], borrows from LSD SLAM's initialization procedure, and therefore also suffers from the problem of random depth initialization, where several keyframes must be added to the system before a stable configuration is reached.

Finally, ORB-SLAM [95] deals with the limitations arising from all the above methods by computing, in parallel, both a Fundamental matrix and a Homography [45]. In order to select the appropriate model, each model is penalized according to its symmetric transfer error [57]. If the chosen model produces poor tracking quality, and there are very few feature correspondences in the next frame, the initialization is discarded, and the system restarts with a different pair of frames.

3.2 Data association

In the first formulation of monocular SLAM [92] the Harris corners detection algorithm [56] was used as the basis of the feature extraction step. This trend is now considered as the feature-based approaches and in several subsequent works ([79, 110]) the Harris detector was used as the basis of the feature extraction step. Other works, for example, PTAM [69] and DT SLAM [60] use the FAST functions to achieve a high processing speed. Then, these FAST features are associated with a local patch of pixels and the data association is carried out using binary comparisons. In ORB-SLAM [95], ORB features with associated ORB descriptors ([113]) demonstrated a good tradeoff between accuracy and speed processing. Therefore, there are other works which have used the same approach [94, 111]. Finally, in some works such as [108, 127] high order features (SIFT) were used to improve the accuracy of the system when decreasing matching outliers.

In other trend (direct-based approaches) algorithms such as, LSD-SLAM [41] and DPPTAM [33], extract and use all pixels that have a photometric gradient (Intensity gradient). DSO [43] shown that the use of all the pixel information with a photometric gradient introduces redundancy in the system, and requires a step of regularization; therefore, DSO proposed to sub-sample the pixels by dividing the image into blocks, maintaining a fixed number of pixels with the highest gradient in each block. This ensures that, in the first place, the sampled pixels are well distributed in the image, and second, that the sampled pixels have sufficiently high image gradients with respect to their immediate surroundings. The sampled pixels are known as candidate points. Different than other systems, SVO [106] employs a hybrid approach in which it sequentially alternates between direct and feature-based methods. i.e., in a first attempt a direct-based approach tries to establish data associations for the current time. Then, if the data association quality is lower than a threshold value, this result is refined using a feature-based model as keystone of the association process.

For the association of 2D-2D data, PTAM [69] generates a pyramid representation of 4 levels of every incoming frame and uses it to enhance the features robustness to scale changes, and to increase the convergence radius of the pose estimation module. In PTAM, FAST features are extracted at each level with a Shi-Tomasi score [116]. Then, features with a relatively smaller score are removed and then non-maximum suppression takes place. Once 2D features are extracted, the 3D features are projected in the new frame, using a previous camera pose estimation (from motion model). Then, the 3D-2D data association is then employed. The descriptor used for data association is extracted from the 2D image from which the 3D feature was first observed. To take account of changes in views, the local patch of pixel descriptors is deformed through an affine projection, which simulates how it would appear in the current frame. However, this constitutes a limitation in the PTAM, since, for large changes in camera viewpoints, the warping transform fails to accurately reflect the correct distortion, therefore causing data association failure.

For DT-SLAM [60], when a new T_i frame is processed, it estimates a 2D similarity transformation through the image registration with the previous frame T_{i-1}, and transforms, using the estimated 2D similarity, the features extracted from T_i into T_{i-1}. The 3D features are then projected in T_{i-1} and the data association takes place, similarly to how it is done in PTAM. DT SLAM also tracks 2D features, which are features that were previously observed but were not triangulated in 3D features due to the lack of parallax between the different frames observing them (i.e. when the camera undergoes a pure rotation motion). For each 2D feature, the Euclidean distance between its epipolar line and the transformed feature is estimated; if it falls below a threshold, the characteristic is considered as a possible coincidence with the 2D feature. The association of data through Zero Mean Sum of Squared Distance (ZMSSD) attempts to validate the matches. SVO [106] generates a pyramidal representation of five levels of the incoming frame; the data association is first established through the iterative direct image alignment, from the highest level of the pyramid to the third level. Preliminary data association of this step is used as a FAST feature comparison procedure, similar to PTAM's warping technique, with a Zero-Mean SSD score.

ORB-SLAM [94, 95] extracts FAST corners in eight levels of a pyramid. To ensure a homogeneous distribution throughout the entire image, each level of the pyramid is divided into cells and the parameters of the FAST detector are tuned online to ensure a minimum of five corners are extracted per cell. Then, a 256-bit ORB descriptor is computed for each extracted feature. ORB-SLAM discretizes and stores the descriptors in bags of words,

known as visual vocabulary, which are used to speed up image and feature matching by constraining those features that belong to the same node in the vocabulary tree. To deal with viewpoint changes, ORB SLAM proposes to keep track of all the keyframes in which a feature is observed and the algorithm choose the descriptor from the keyframe that has the smallest viewpoint difference with the current frame. Finally, in DSO-SLAM [43], the candidate points, sampled through the image, are represented by eight pixels spread around the target point. Then, the algorithm claims that the use of this number of pixels in a specific pattern was was found empirically to return a good compensation between three objectives: computational time, sufficient information for tracking to take place, and resilience to motion blur. Each of the selected pixels around the candidate point contributes to the energy function, which it seeks to minimize during tracking. Within this formulation, he association of data is still inherent in the direct image alignment scheme; however, use only the candidate points and their selected surrounding pixels, instead of using all the pixels with gradients in an image.

3.3 Pose estimation

In PTAM [69], pose estimation starts by estimating a position before the frame using a decreasing constant velocity motion model, as described in **Section** 2.3.1. Then the previous one is refined using a Small Blurry Image (SBI)–the smallest image resolution in the pyramid representation of the frame—by applying an Efficient Second Order minimization [17]. If the speed is high, PTAM anticipates that a rapid movement is taking place, and hence, the presence of fuzzy motion and, and therefore, the tracking to take place only at the highest pyramid levels (most resilient to motion blur) in what is known as a rough follow-up stage. Otherwise, the coarse tracking stage is followed by a fine tracking stage. However, when the camera is stationary, the thick stage can cause a change in camera posture and, therefore, turn off. The initial camera pose prior is refined by minimizing a tukey-biweight [86] objective function of the re-projection error that reduces the weight of observations with large residuals. To determine the the quality of tracking, PTAM monitors the proportion of features matching successfully in the frame, against the total number of matching attempts of FAST features.

SVO [106] assumes the pose of the new frame to be the same as the previous one; then, it looks for the transformation that minimizes the photometric error of the pixels of the image with the associated depth measurements in the current frame, with respect

to its location in the previous one. The minimization takes place through thirty Gauss Newton iterations of the inverse compositional image alignment method. Once the image alignment is performed, features that are expected to be visible in the current frame, are projected onto the image. To decrease the computational complexity and to maintain only the strongest features, the frame is divided into a grid, and only the strongest feature per grid cell is used. The 2D location of the projected function is adjusted by minimizing the photometric error between its associated patch from its location in the current frame, and a warp of the feature generated from the nearest keyframe observing it. This minimization violates the epipolar constraint for the entire frame, and further processing in the tracking module is required: motion-only bundle adjustment takes place, followed by a structure only bundle adjustment that refines the 3D location of the features, based on the refined camera pose. Finally, a joint (pose and structure) local bundle adjustment adjusts the reported camera position estimate. During this last stage, the tracking quality is continuously monitored and, if the number of observations in a frame, or the number of features between consecutive frames drop, tracking quality is deemed insufficient, and failure recovery methods are initiated.

DT-SLAM [60] maintains a camera position based on three tracking modes: full pose estimation, essential matrix estimation, and pure rotation estimation. When there is a sufficient number of 3D matches, a full pose can be estimated; otherwise, if a sufficient number of 2D matches showing small translations is established, an Essential matrix is estimated; and finally, if a pure rotation is shown, two points are used to estimate the absolute orientation of the matches. Pose estimation aims, in an iterative manner, to minimize the error vector of both 3D-2D re-projections, and 2D-2D matches. When tracking failure occurs, the system initializes a new map and continues to collect data to track on a different map; however, the map making thread continues to look for possible matches between the keyframes of the new map and the old one, and once a match is established, both maps are fused together, thereby allowing the system to handle multiple sub-maps, each at a different scale.

The tracking thread in LSD-SLAM [41] is responsible for estimating the pose of the current frame with respect to the current active keyframe in the map, using the position of the previous frame as before. The required pose is represented by a $SE(3)$ transformation, and is found by an iteratively re-weighted Gauss-Newton optimization that minimizes the residual normalized error of the variance, as described in [40]. A keyframe is considered active if it is the most recent keyframe hosted on the map. To minimize outlier effects, measurements with large residuals are down-weighted from one iteration to the next.

The estimation of the position in ORB SLAM [94, 95] is established through a previous constant velocity movement model (**Section** 2.3.1), followed by a refinement of the posture by optimization. Since the motion model is expected to be easily violated through abrupt motions, ORB SLAM detects such failures by tracking the number of matched features; if it falls below a certain threshold, the points on the map are projected in the current frame, and a wide-range feature search is performed around the projected locations. In an effort to make ORB SLAM operate in large environments, a subset of the global map, known as the local map, is defined by all features corresponding to the set of all keyframes that share edges with the current frame, as well as all neighbors of this set of keyframes from the pose graph. The selected features are filtered out to keep only the features that are most likely to be matched in the current frame. Furthermore, if the distance from the camera's center to the feature is beyond the range of the valid features, the feature is also discarded. The remaining set of features is then searched for and matched in the current frame, before a final camera pose refinement step.

Similar to LSD-SLAM, DPPTAM [43] optimizes the photometric error of high gradient pixel locations between two images, using the ICIA formulation over the $SE(3)$ transform that relates the corresponding points. Minimization is initiated using a constant velocity motion model, unless the photometric error increases after its application. If the latter is true, the motion model is not taken into account and the pose of the last frame followed is used. Similar to PTAM, optimization in DPPTAM takes place in the tangent space $S\xi(3)$ that minimally parametrizes the transformation of the rigid body by six parameters.

In DSO-SLAM [43], all the frames tracked simultaneously and used in the map update process; however, each frame contributes differently, and is treated according to whether a key frame is considered or not. DSO-SLAM uses two parallel threads: a front-end thread, and a mapping thread. Front-end initializes the system at startup using random depth initialization: it computes the intensity gradients, and tracks the current frame with respect to the currently active keyframe. Different than other systems, DSO-SLAM does not use a single frame pose prior; rather, it attempts a direct image alignment by looping over multiple pose guesses, in a pyramidal implementation, and removes guesses that yield higher residuals between iterations. The final pose estimate that yields the smallest residual error is then assigned to the current frame. Finally, in recent works [76, 129], the camera pose estimation is solved via CNN implementations. Different to previous approaches, in [76, 129] the feature extraction and matching steps are avoided; instead a CNN can compute the camera poses and the 3D map in a direct form.

3.4 Map construction

When a new keyframe is added in PTAM [69], all bundle adjustment operations are halted, and the new keyframe inherits the pose from the coarse tracking stage. The potentially visible set of features estimated by the tracker are then re-projected onto the new keyframe, and feature matches are established. Correctly matched features are marked as seen again; this is done to keep track of the quality of the features and to allow for the map refinement step to remove corrupt data. New features are generated by establishing and triangulating feature matches between the newly added keyframe and its nearest keyframe (in terms of position) from the map. Landmarks that are already existent in the map are projected onto both keyframes, and feature matches from the current keyframe are searched for along their corresponding epipolar lines in the second keyframe, at regions that do not contain projected features. The average depth of the projected features is used to constrain the epipolar search, from a line to a segment.

SVO [106] parametrizes the 3D features using an inverse depth parameterization model [31]. When inserting a new keyframe, features possessing the highest Shi-Tomasi scores are chosen to initialize a number of depth filters. These features are referred to as seeds, and are initialized along a line propagating from the camera center to the 2D location of the seed in the originating keyframe. The only parameter that remains to be solved for is the depth of the feature, which is initialized to the mean of the scene's depth, as observed from the keyframe of origin. During the times when no new keyframe is processed, the map management thread monitors and updates map seeds through subsequent observations, similar to [137]. The seed is searched in new frames along an epipolar search line, which is limited by the uncertainty of the seed, and the average depth distribution observed in the current frame. As the filter converges, its uncertainty decreases, and the epipolar search range decreases. If the seeds do not coincide frequently, if they diverge to infinity or, if a long time has passed since their initialization, they are removed from the map. This process however limits SVO to operate in environments of relatively uniform depth distributions. Since the initialization of features in SVO relies on many observations in order for the features to be triangulated, the map contains few, if any, outliers, and hence no outlier deletion method is required. However, this occurs at the expense of a delay time before the features are initialized as features and added to the map.

DT-SLAM [60] aims to add keyframes when enough visual change has occurred; the three criteria for keyframe addition are (1) for the frame to contain a sufficient number

of new 2D features that can be created from areas not covered by the map, or (2) a minimum number of 2D features can be triangulated into 3D features, or (3) a given number of already existing 3D features have been observed from a significantly different angle. The map in DT-SLAM contains both 2D and 3D features, where the triangulation of 2D features into 3D features is done through two view triangulation by optimization, and is deferred until enough parallax between the keyframes is observed.

In LSD-SLAM [41] the map generation module is mainly responsible for the selection and accommodation of new keyframes into the map. Its functions can be divided into two main categories, depending on whether the current frame is a keyframe or not; if it is, depth map creation takes place by keyframe accommodation as described below; if not, the creation of the depth map is done on regular frames. When the system is accommodating a new keyframe, the estimated depth map from the previous keyframe is projected onto it, and serves as its initial guess. Spatial regularization then takes place, by replacing each projected depth value with the average of its surrounding values, and the variance is chosen as the minimal variance value of the neighboring measurements. The $Sim(3)$ of a newly added keyframe is then estimated and refined in a direct, scale-drift aware image alignment scheme with respect to other keyframes in the map, over the seven degree of freedom $Sim(3)$ transform. Due to the non-convexity of the direct image alignment method on $Sim(3)$, an accurate initialization to the minimization procedure is required; for such a purpose, ESM (Efficient Second Order minimization) and a coarse to fine pyramidal scheme with very low resolutions proved to increase the convergence radius of the task.

In ORB SLAM [94, 95] the local mapping thread is responsible for keyframe insertion, map point triangulation, map point culling, keyframe culling, and local bundle adjustment. ORB SLAM incorporates a hybrid map, one metric and two topological maps. However, the two topological maps, referred to as co-visibility and essential graphs, are built using the same nodes (keyframes) however, with different edges (connections) between them. The co-visibility graph allows for as many connections as available between nodes; in contrast to the essential graph that allows every node to have at most two edges, by only keeping the strongest two edges. The mapping thread is responsible for updating the co-visibility and essential graphs with the appropriate edges, as well as computing the bag of words representing the newly added keyframes in the map. The metric map is propagated by triangulating new features from ORB features, which appear in at least two nodes connected to the new keyframe in the co-visibility graph. To prevent outliers, triangulated features are tested to determine positive depth, re-projection error, and scale

consistency in all keyframes they are observed in, before finally incorporating them into the map.

The triangulation of reference points in DPPTAM [43] takes place over several overlapping observations of the scene using inverse depth parametrization; the map maker aims to minimize the photometric error between a high gradient pixel patch in the last added keyframe, and the corresponding patch of pixels, found by projecting the feature from the keyframe onto the current frame. The minimization is repeated ten times for all high gradient pixels, when the frame exhibits enough translation; the threshold for translation is increased from one iteration to the next, to ensure sufficient baseline distance between the frames. The end result is ten hypotheses for the depth of each high gradient pixel. To deduce the final depth estimate from the hypotheses, three consecutive tests are performed, including gradient direction test, temporal consistency, and spatial consistency.

Finally, in DSO [43] all frames are used in the map building process; while keyframes are used to expand the map and perform optimize the window, regular frames (non-keyframe) are used to update the depth of the already existing candidate points. DSO maintains two thousand candidate points per keyframe. The estimated pose of the subsequent regular frames, the location of the candidate points in the active keyframe and their variance, are all used to establish an epipolar search segment in the regular frame. The image location along the epipolar segment, which minimizes the photometric error, is used to update the depth and the variance of the candidate point, using a filter-based triangulation, similar to LSD SLAM [41]. DSO adopts the inverse depth paradigm as a parameterization for the 3D world which reduces the parameters to optimize to one variable; therefore reducing computational cost. This estimated depth is used as a prior for a subsequently activated candidate point in a windowed optimization. In its active window of optimization, DSO maintain seven active keyframes, along with two thousand active points, equally distributed across the active keyframes. As new keyframes and candidate points are accommodated by the system, older ones are marginalized: where the number of active keyframes exceeds 7, the system chooses a keyframe from the active window and marginalizes it. The choice of keyframe is made by maximizing a heuristic designed distance score, which ensures that the remaining active keyframes to be well distributed across the space between the first and last keyframes in the active window, and closer to the most recently added keyframe. Also if ninety-five percent of a frame's points are marginalized, the frame is removed from the system.

3.5 Refinement

When the map making thread is not processing new keyframes, PTAM [69] performs several optimizations and maintenance of the map and camera poses, such as a Local Bundle Adjustment for local map convergence and a Global Bundled Adjustment for the global convergence of the map. The computational cost in PTAM is scaled with the map and becomes intractable as the number of keyframes increases; for this reason, PTAM is designed to work in small workspaces. Finally, the optimization thread applies the refinement of the data by first searching and updating feature observations in all the keyframes, and then by removing all the features that failed, many times, to match the characteristics successfully. For reasons of runtime efficiency, SVO [106] keeps only a fixed number of keyframes on the map and removes the distant ones when new keyframes are added. This is performed so that the algorithm maintains real-time performance after prolonged periods of operation over large distances. DT SLAM [60] employs a third thread that continuously optimizes the entire map and the camera poses in the background through a sparse Global Bundled Adjustment. LSD SLAM [41] runs a third parallel thread that continuously optimizes the map and the camera poses in the background by a generic implementation of a pose graph optimization using the g2o-framework [72]. However, this leads to a low accuracy compared to other methods. Atypical values are detected by monitoring the probability of the projected depth hypothesis at each pixel of being an outlier or not. To make the outliers detection step possible, LSD-SLAM keeps records of each successfully matched pixel during the tracking thread, and increases or decreases accordingly the probability of it being an outlier.

ORB-SLAM [94, 95] employs rigorous feature culling to ensure few outliers in the map. A feature must be correctly matched to twenty-five percent of the frames in which it is predicted to be visible. It must also be visible from at least three keyframes after more than one keyframe has been accommodated on the map, since it was spawned. Otherwise, the feature is removed. To maintain lifelong operation and to counter the side effects of the presence of a high number of keyframes in the map, a rigorous keyframe culling procedure takes place as well. Keyframes that have ninety percent of their associated features observed in three other keyframes are deemed redundant, and removed. The local mapping thread also performs a Local Bundle Adjustment over all keyframes connected to the last accommodated keyframe in the co-visibility graph, and all other keyframes that observe any feature present in the current keyframe.

DPPTAM [43] produces dense maps in real time by employing a dense mapping thread that exploits planar properties of manmade indoor environments. Keyframes are first segmented into a set of 2D superpixels, and all 3D features from the map are projected onto the keyframe, and assigned to different superpixels according to the distance of their projections to the appropriate superpixel in the keyframe. 3D points belonging to contours of the superpixels are used to fit 3D planes to each superpixel. To determine if the superpixel's plane is to be added into the map, three tests are performed: the normalized residual test, the degenerate case detection, and the temporal consistency test. Then, a full dense map is reconstructed, using the depth priors of the 3D planes associated with the superpixels. Finally, DSO [43] performs a windowed optimization on the photometric combination (intensity) and geometric residual of all active points between the set of active keyframes, using six iterations of Gauss-Newton. If the resulting residual of the most recently added keyframe after the optimization is large, the newly added keyframe is dropped. Map maintenance in DSO is also responsible for the detection and management of atypical values at an early stage of the DSO formulation.

3.6 Loop closure

3.6.1 Relocalization

Upon detecting a fault, the PTAM [69] tracker initiates a recovery procedure, where the SBI of each incoming frame is compared to the SBI database (Small-Blurry-Image) for all keyframes. If the intensity difference between the incoming frame and its closest looking keyframe is below a certain threshold, the current frame's pose is assumed to be equivalent to that of the corresponding keyframe. ESM tracking takes place to estimate the rotational change between the keyframe and the current frame. If converged, the tracker attempts to match the features to the features in the frame. If a sufficient number of features are correctly matched, the tracker resumes normally; otherwise, a new frame is acquired and the tracker remains lost. In SVO [106] the first procedure in the recovery process is to apply image alignment between the incoming frame and the closest keyframe to the last known correctly tracked frame. If more than thirty features are correctly matched during this image alignment step, then the re-localizer considers that is converged and continues tracking regularly; otherwise, it attempts to relocalize using new incoming frames.

The LSD-SLAM [41] recovery procedure first chooses, at random, from the pose graph, a keyframe that has more than two adjacent keyframes connected to it. Then, LSD-SLAM tries to align the currently missing frame. If the ratio of outlier to inlier is large, the keyframe is discarded, and replaced by another random keyframe; otherwise, all neighboring keyframes connected to it are verified in the pose graph. If the number of neighbors with a large inlier-to-outlier ratio is larger than the number of neighbors with a large outlier-to-inlier ratio, or if there are more than five neighbors with a large inlier-to-outlier ratio, the neighboring keyframe with the largest ratio is set as the active keyframe, and regular tracking resumes.

Upon running, the ORB SLAM [94, 95] re-localizer transforms the current frame into a bag of words and queries the database of keyframes for all possible keyframes that might be used to relocalize from. The place recognition module implemented in ORB SLAM, used for loop detection and failure recovery, is based on bags of words, since frames that observe the same scene share a large number of common visual vocabulary. In contrast to other bag of words methods that return the best hypothesis consulted from the keyframe database, the ORB-SLAM place recognition module returns all possible hypotheses that have a probability of being a match larger than seventy-five percent of the best match. The combined added value of the ORB features, along with the bag of words implementation of the place recognition module, manifest themselves in a real-time, high recall, and relatively high tolerance to viewpoint changes during relocalization and loop detection. All hypotheses are then tested through a RANSAC implementation of the PnP algorithm [75], which determines the camera pose from a set of 3D to 2D correspondences. The camera pose with the most inliers is then used to establish more matches to features associated with the candidate keyframe, before an optimization over the camera's pose takes place.

Finally, in DSO [43], there is no world-based fault recovery method. When the minimization of DSO pose tracking is diverted, the last successfully tracked camera pose is used to generate multiple arbitrary random rotations around it. The generated poses are used in an attempt to locate at the thickest pyramid level with the most recent active keyframe; if the photometric minimization is successful, regular tracking resumes, otherwise, tracking fails.

3.6.2 Loop closure

Whena tracking error occurs in DT-SLAM [60], a new secondary map begins and DT-SLAM start tracking it while a loop closure thread attempts to establish data associations across different sub-maps. Therefore, the DT-SLAM loop closure module is a modified version of the PTAM failure recovery module [69] employed across DT-SLAM sub maps. When a sufficient number of data associations are successfully established between two keyframes, their corresponding sub-maps are merged together through a similarity transform optimization. When a keyframe is processed by LSD-SLAM [41], loop closures are searched for within its ten nearest keyframes as well as through the appearance based model of FABMAP [51] to establish both ends of a loop. Once a loop edge is detected, a pose graph optimization minimizes the similarity error established at the loop's edge, by distributing the error over the poses of the loop's keyframes.

Loop detection in ORB SLAM [94, 95] takes place via a global place recognition module, that returns all hypotheses of keyframes, from the database that might correspond to the opposing loop end. All features associated with the queried keyframe and its neighbors are projected to, and searched for, in all keyframes associated with the current keyframe in the co-visibility graph. The initial set of inliers, as well as the matches found, are used to update the co-visibility and Essential graphs, thereby establishing many edges between the two ends of the loop. Finally, an optimization of the pose graph is carried out, similar to that of LSD-SLAM [41], which minimizes and distributes the closing error of the loop along the nodes of the loop. Finally, keyframes and marginal points in DSO [43] are permanently removed from the system and never used again.

3.7 Monocular-SLAM: limitations and future trends

Each different solution of monocular-SLAM is favored by different operating conditions. For example, SVO [106] prefers the high frame rate inputs of down-facing cameras, DPPTAM [43] can only operate in indoor environments where most of the observed scene is composed of planar surfaces. DT-SLAM [60] requires that the scene be observed repeatedly. Furthermore, there is no public data set in the literature that allows us to make an impartial experimental comparison in all systems. Therefore, in this section, we discuss and evaluate the ramifications of the decisions made in each component of the different monocular-SLAM systems, providing a theoretical view of the limitations of the different module designs.

3.7.1 Performance and limitations of direct approaches

Direct methods take advantage of all the information available in the image and, therefore, are more robust than the methods based on characteristics in regions with little texture and blur. Nevertheless, direct methods are susceptible to changes in the lighting of the scene, of the assumption of underlying brightness consistency. In an effort to gain resistance to this failure mode, the recently launched direct approaches model the imaging process and attempt to incorporate the irradiance of the scene into functional energy, at the cost of adding a calibrated imaging model that is used to correct images in a preprocessing step. In practice, this model is estimated through an additional offline calibration process described in [43].

During the non-linear optimization process, it is linearized through a first order Taylor expansion. While the linearization is valid when the parameters of the warping transform tends to zero, higher order terms become dominant and the linearization becomes invalid for large transforms. Therefore, a second disadvantage of direct methods is the assumption of small motions between the images (typically not more than 1 pixel). To relax this constraint, direct monocular-SLAM systems employ a pyramidal implementation, where the image alignment process takes place sequentially from the highest pyramid level to the lowest, using the results of every level as a prior to the next level. Several authors also suggest the use of high fame rate cameras to alleviate this issue; some systems employ an efficient second order minimization to estimate a rotation prior that helps increase the convergence radius. Despite these efforts, the tolerated baseline for data association in direct methods is considerably smaller than the tolerated baseline in feature-based methods. Finally, another disadvantage of the direct methods is that the calculation of the photometric error in each pixel is computationally intensive; therefore, until recently, real-time SLAM monocular applications of direct methods were not considered feasible. However, with the recent advancements in parallel processing and with the introduction of semi-dense reverse depth filtering, it was possible to integrate direct methods into real-time solutions [42, 46], unfortunately, the processing is less than if based on characteristics approaches.

3.7.2 Performance and limitations of feature-based approaches

Feature-based methods are relatively robust to changes in illumination and can tolerate wider baselines; however, the extraction processes that make them resilient to these fac-

tors are generally computationally expensive. For real-time operation constraints, most systems employ an exchange between a type of entity to use in one hand, and robustness and resistance to environmental factors in the other. To mitigate this constraint, other systems, such as the work of [127], resort to parallelized GPU implementations for feature detection and extraction. Another disadvantage of feature-based methods is that even the top performing feature descriptors are limited in the amount of scene change (lighting and viewpoint) they can handle before failure. Feature matching is also prone to failure in similar-self repeating texture environments, where a feature in I_1 can be ambiguously matched to multiple other features in I_2. Outliers in the data association module can significantly degrade system performance by inducing errors in both the camera postures and the generated map to the point of failure. Feature-based methods also suffer from lack of features in textureless regions, causing feature-based approaches to fail in texture-deprived environments.

3.7.3 Performance and limitations of the initialization step

Aside from the random depth initialization of LSD-SLAM [41] and DSO [43], all the suggested methods described above suffer from degeneration under certain conditions, such as under low parallax movements of the camera, or when the structure of the scene is assumed is violated. The PTAM [69] initialization procedure is brittle and remains tricky to perform, especially for inexperienced users. Furthermore, it is subject to degeneracies when the planarity of the initial scene's assumption is violated, or when the user's motion is inappropriate; thereby crashing the system, without means of detecting such degeneracies. As is the case in PTAM, the initialization of SVO [106] requires the same type of motion and is prone to sudden movements, as well as to non-planar scenes. Furthermore, monitoring the median of the baseline distance between features is not a good approach to automate the initial keyframe pair selection, as it is prone to failure against degenerate cases, with no means of detecting them. The initialization model of ORB-SLAM [94, 95] attempts to automatically initialize the system by monitoring the baseline and the scene across a window of images. If the observed scene is relatively far away, while the camera slowly translates into the scene, the system is not capable of detecting such scenarios, and fails to initialize. While a random depth initialization from a single image does not suffer from the degeneracies of two view geometry methods, the depth estimation requires that the processing of the subsequent frames converge, resulting in an intermediate follow-up phase in which the map generated is not reliable.

3.7.4 Performance and limitations of the data association step

In general, establishing data associations remains one of the biggest challenges in monocular-SLAM. Systems that limit the search range along the epipolar line using the observed depth information, implicitly assume a relatively smooth depth distribution. The violation of this assumption (that is, when the scene includes significance variance in the observed depth) causes the 2D features corresponding to potential future 3D features to fall outside the boundaries of the epipolar segment, and the system ends up neglecting them. Other limitations for data association arise from large erratic accelerations in the camera's motion, also causing features to fall outside the scope of the search window. Such a scenario is common when the camera is operated by an untrained user. Under the same type of motions, image pollution with motion blur also negatively impacts the performance of data association methods to the point of failure. Erroneous data association is also a very common problem that can cause false positives in self-repeating environments. Most current implementations of data association address this problem through a bottom-up approach, where low level information from image pixels or from features, is used to establish correspondences. To mitigate some of these issues, a number of systems have attempted to use more important geometric features, such as lines [21, 70, 146], superpixels or planar features [32, 88], or priors on 3D shapes in the scene [48]. Recent advances in machine learning are promising alternatives to remedy some of the data association issues by automatically learning to extract and match features [118, 135].

3.7.5 Performance and limitations of the pose estimation step

Systems based on constant motion models, such as PTAM [69] and ORB-SLAM [94, 95] are prone to tracking failure when abrupt changes in the direction of the camera's motion occurs. While both employ a recovery of such failures, the tracking performance of PTAM is exposed to a false positive recovery; as opposed to ORB-SLAM , which first tries to increase the search window before invoking its fault recovery module. Another limitation of feature-based pose estimation is the detection and handling of occlusions. As the camera translates in the scene, some features in the background are prone to occlusions from objects in the foreground. When the system projects the 3D map points onto the current frame, it fails to match the occluded features, and counts them toward the camera tracking quality assessment. In extreme cases, the tracking quality of the system might be deemed bad and tracking failure recovery procedures are invoked even though camera pose

tracking did not fail. Furthermore, occluded points are flagged as outliers and passed to the map maintenance module to be removed, depriving the map from valid useful features that were erroneously flagged due to occlusions in the scene.

Other systems, that use the previously tracked pose as a prior for the new frame's pose, are also prone to the same limitations of constant velocity models. Furthermore, they require small displacements between frames, limiting their operation to relatively expensive high frame rate cameras (typically > 70 fps) such that the displacement limitation is not exceeded. Another limitation of these methods is inherent from their use of direct data association. Their tracking module is susceptible to variations in the lighting conditions. To gain some resilience to lighting changes in direct methods, DSO authors [43] suggest an off-line photometric calibration process to parametrize and incorporate lighting variations within the camera pose optimization process. Finally, a common limitation in most tracking modules is the presence of dynamic objects in the observed environment. The use of multiple cameras (multi-camera visual SLAM) could solve this issue. Since dynamic objects can be detected and removed via multiple-view motion models. However, in the case of monocular-SLAM the limitation of a single moving camera makes it not possible to use these motion models.

As most monocular-SLAM systems assume a static scene, the tracking modules of most systems suffer from tracking failures: a significantly large dynamic object in the scene could trick the system into thinking that the camera itself is moving, while it did not move relative to the environment. Small, slow-moving objects can introduce noisy outlier features on the map and require subsequent processing and handling to be removed. On the other hand, small and fast moving objects do not affect the tracking module as much. Finally, small rapidly moving objects tend to violate the epipolar geometry of the pose estimation problem, and are easily flagged and removed from the camera pose optimization thread; however, they can occlude other features.

3.7.6 Performance and limitations of the map construction step

A major limitation in the method of optimization by triangulation is the requirement of a significant baseline that separates two points of view by observing the same characteristic. Hence, it is prone to failure when the camera's motion is made of pure rotations. To counter such modes of failure, DT-SLAM [60] introduced 2D features that can be used to expand the map during pure rotations, before they are triangulated into 3D features. However, the observed scene during the rotation motion is expected to be re-observed with more

baseline, for the features to transition from 2D to 3D. Unfortunately, in many applications this is not the case; for example, a camera mounted on a car making a turn cannot re-observe the scene, and eventually tracking failure occurs. DT-SLAM addresses such cases by generating a new sub map and attempts to establish connections to previously created sub-maps by invoking a thread to look for similar keyframes across sub-maps, and establish data associations between them. Meanwhile, it resumes tracking in the new world coordinate frame of the new sub-map. However, this makes the pose estimates obsolete; at every tracking failure, the tracking is reset to the new coordinate frame, yielding useless pose estimates until the sub-maps are joined together, which may never occur.

In filter-based triangulation methods, outliers are easily flagged as features whose distribution remains approximately uniform after several observations have been incorporated in the framework. This reduces the need for a subsequent processing step to detect and handle outliers. Also, features at infinity feature parallax values that are too small for triangulation purposes; but still, it can be used to improve the camera's rotation estimates, stay on the map and go from infinity to the metric map, when enough parallax is recorded between the views observing them. However, these benefits come at the expense of increased complexity in implementing a probabilistic framework, which keeps track and updates the uncertainty in the depth distribution of every pixel with a gradient in the system. Furthermore, while the dense and semi-dense maps can capture a much more meaningful representation of a scene than a sparse set of 3D features, the added value is diminished by the challenges of handling immense amounts of data in 3D. Therefore, it is necessary to have additional top-level semantic information to reason about the observed scene and to improve the overall performance of the system. While monocular SLAM systems have been shown to improve the results of semantic labeling [103], the feedback from the latter to the former remains a challenging problem. Previous work on the subject includes, among others, [12, 73, 142].

3.7.7 Performance and limitations of the refinement step

Pose Graph Optimization (PGO) returns inferior results to those produced by GBA (Global Bundle Adjustment), while PGO optimizes only for the poses of keyframes, that is, adjusts the 3D structure of the visual features; GBA and LBA (Local Bundle Adjustment) are optimized together for keyframe poses and the 3D structure. The stated advantage comes at the cost of computational time, with PGO exhibiting a significant speed up compared to the other methods. PGO is often employed during the closure of

the loop since the computational cost of running a full packet adjustment is often unsuitable in large-scale loops; however, pose graph optimization may not yield optimal result if the errors accumulated over the loop are distributed along the entire map, leading to locally induced inaccuracies in regions that were not originally wrong.

3.7.8 Performance and limitations of the loop closure step

For successful re-localization or loop detection, the global location methods employed by PTAM [69], SVO [106] and DT-SLAM [60] require that the camera's pose be close to the recorded keyframe's pose, and would otherwise fail when there is a large displacement between the two. Furthermore, they are highly sensitive to any change in the lighting conditions of the scene, and may yield many false positives when the observed environment is composed of self-repeating textures. Other methods that rely on bags of words representation of high dimensional features are susceptible to failure when the training set of the bag of words classifier is not representative of the working environment in which the system is operating.

3.7.9 Challenges

Although extensive research has been dedicated to the monocular-SLAM formulation, each of the building blocks discussed above could benefit from many improvements of which we list the following:

- Robust data association against illumination changes, dynamic scenes, and occluded environments.
- A robust initialization method that can operate without an initial scene assumption.
- An accurate camera pose estimate that is not affected by sudden movements, blur, noise, large depth variations, nor moving objects.
- A map making module capable of generating an efficient dense scene representation in regions of little texture, while incorporating a higher level of perception.
- A map maintenance method that improves the map, with resilience against dynamic, changing environments.
- A failure recovery procedure capable of recovering the system from significantly large changes in camera viewpoints.

- A mathematical formulation that allows efficient embedded implementation.

These are all desired properties that remain challenging topics in the field of monocular-SLAM. Furthermore, with the recent advancements in machine learning, researchers are moving towards integrating semantic data within the context of monocular-SLAM. While the incorporation of semantic data into SLAM is undoubtedly the next step in the right direction, we argue that such integration requires a hybrid fusion approach that tightly integrates metric, topological and semantic representations in a symbiotic relationship, a research area relatively uncharted.

3.8 Discussion

In previous works, several contributions for the essential building blocks of the generic monocular-SLAM formulation were made; including data association, visual initialization, pose estimation, topological/metric map generation, BA/PGO/map maintenance, and global localization. Although extensive research has been dedicated to improve those blocks and currently monocular-SLAM systems reach high accuracy and a relatively high processing speed, it is our opinion that each of the building blocks discussed above could benefit from many improvements. In our case, we are interested in monocular-SLAM solutions suitable for embedded systems. In this context, in previous works (PTAM, SVO, DT SLAM, LSD SLAM, ORB SLAM, DPPTAM, and DSO) the most accurate and used solution is based on optimization techniques implemented in sequential processors. This makes possible to reach high accuracy for the camera pose estimation but limits the processing speed, embedded capabilities and deliver sparse point clouds. So, in order to reach high efficiency under embedded systems, in this research, we reformulate the monocular-SLAM problem in order to facilitate an FPGA/CUDA implementation, suitable for embedded applications, real-time processing and dense point cloud estimations. For the camera pose estimation, we will explore about a dense feature matching as linear/dependent pattern for the pose estimation. For the feature matching algorithm, we will explore about a new pixel tracking/feature matching algorithm which consists in an extension of the stereo matching problem; for that, we will use a pixel-parallel/window-parallel approach based on a Sum of Absolute Difference and, in order to improve the correlation performance, we will explore the curl of the intensity gradient as preprocessing step. Finally, to recover depth in the scene, we propose the norm of the pixel tracing (optical flow) as linear dependent to depth.

3.9 Summary

In this chapter a detailed discussion about the state of the art was presented. Performance and limitations of the current monocular-SLAM systems and the current open and possible future trends/strategies for the solution of each of these limitations were discussed. In order to address the embedded capabilities limitations (which is the main research interest of this work); in the following chapter, several algorithmic reformulations, new theoretical knowledge for monocular-SLAM and several hardware implantation strategies for FPGA/CUDA architectures will be presented.

Chapter 4

LT-SLAM: Lookup Table-based Monocular-SLAM

In **Fig.** 4.1 an overview of the proposed algorithm is shown. First, feature points (corners) are extracted in the reference image ($f_t(x, y)$), for which a parallel version of the Harris corner detection algorithm was developed [56]. Then, pixel tracking uses a dynamic model to compute 2D pixel displacements between $f_t(x, y)$ and $f_{t+1}(x, y)$ (two consecutive frames from a video sequence). Given pixel tracking for all pixels in the reference image, depth from motion is estimated. Further, feature matching for $f_t(x, y)$ and $f_{t+1}(x, y)$ is computed. Then, search parameters are computed, these parameters are used as searching criteria within a lookup table. Given the search parameters for two consecutive frames, a look up table delivers preliminary pose estimations. Finally, preliminary pose estimations are refined and the map construction is computed in a stream post-processing step.

4.1 Feature extraction

For the feature extraction step, we developed a parallel version of the Harris & Stephens corner detection algorithm [56], see **Fig.** 4.2. Our formulation consists of three steps executing in sequential form. Given an input image $f(x, y)$, first, image derivatives $A(x, y), B(x, y), C(x, y)$ are computed. Then, a corner metric response $m(x, y)$ delivers high pixel values for corner points and low pixel values otherwise. Finally, a thresholding operation delivers a one at corner points retained after a non-maxima suppression step and zero otherwise ($h(x, y)$).

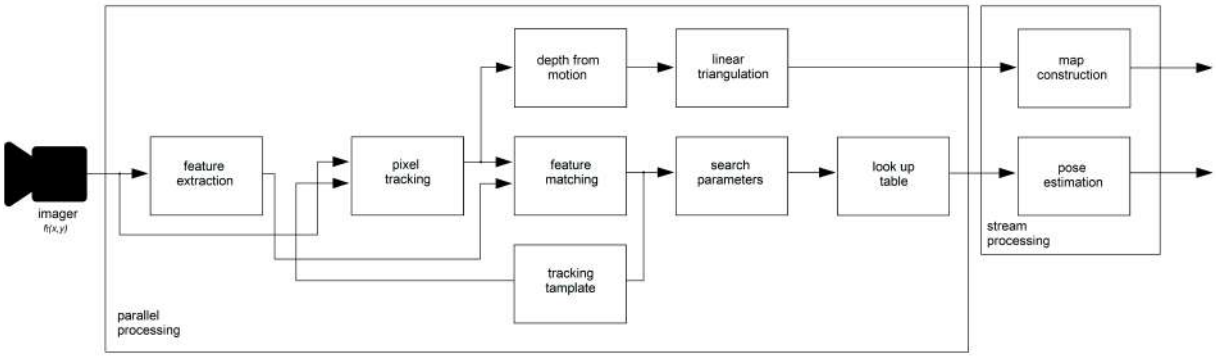


Figure 4.1. Block diagram of the proposed algorithm. First, visual features are extracted and matched. Then, eight search parameters are used as searching criteria within a lookup table, delivering preliminary pose estimations. Finally, preliminary pose estimations are refined and the map construction is computed in a stream post-processing step.

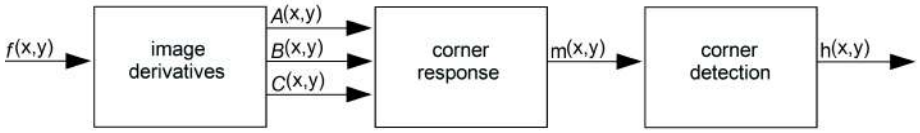


Figure 4.2. Formulation of the feature extraction step. First, image derivatives are computed. Then, a corner metric response delivers high pixel values for corner points and low pixel values otherwise. Finally, a thresholding operation finalizes the feature extraction process.

For the first step: given an input image $f(x, y)$, horizontal and vertical gradients are given by: $G_x(x, y) = f(x, y) \bullet g_x$, $G_y(x, y) = f(x, y) \bullet g_y$, where the operation $f(x, y) \bullet g$ denotes the 2D spatial convolution between an input image $f(x, y)$ and a fixed convolution kernel g. For the convolution kernels, we use the Sobel convolution kernels defined as shown in **Eq.** 4.1. Given the image gradients $(G_x(x, y), G_y(x, y)$, image derivatives $(A(x, y), B(x, y), C(x, y))$ are computed as $A(x, y) = G_x(x, y) * G_x(x, y)$, $B(x, y) = G_y(x, y) * G_y(x, y)$, $C(x, y) = G_x(x, y) * G_y(x, y)$.

$$g_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad g_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \tag{4.1}$$

For the second step, a Gaussian filtering is applied on the image derivatives ($A(x,y)$, $B(x,y)$, $C(x,y)$) in order to reduce noise and removing fine-scale structures that affect the performance of the corner response. This process is defined as $A'(x,y) = A(x,y) \bullet G$, $B'(x,y) = B(x,y) \bullet G$, $C'(x,y) = C(x,y) \bullet G$, where the operator $\bullet$ denotes the 2D spatial convolution between an input image $(A(x,y), B(x,y), C(x,y))$ and a fixed convolution kernel G. The convolution kernel is defined as shown in **Eq.** 4.2. Finally, using the filtered image derivatives, the corner metric response is computed as in **Eq.** 4.3.

$$G = \begin{pmatrix} 0.0178 & 0.0306 & 0.0367 & 0.0306 & 0.0178 \\ 0.0306 & 0.0525 & 0.0629 & 0.0525 & 0.0306 \\ 0.0367 & 0.0629 & 0.0753 & 0.0629 & 0.0367 \\ 0.0306 & 0.0525 & 0.0629 & 0.0525 & 0.0306 \\ 0.0178 & 0.0306 & 0.0367 & 0.0306 & 0.0178 \end{pmatrix} \tag{4.2}$$

$$m(x,y) = A'(x,y) \times B'(x,y) - C'(x,y)^2 - 0.04 \times (A'(x,y) + B'(x,y))^2 \tag{4.3}$$

In the third step, the corner detection process is computed with **Eq.** 4.4; where the operation $m' \circ M$ denotes the matrix composition between patches in the corner response image ($m' = m(x-2:x+2, y-2:y+2)$) and the matrix M (suppression matrix), i.e., $a(1,1) = m'(1,1) * M(1,1)$, $a(1,2) = m'(1,2) * M(1,2) \ldots$ $a(5,5) = m(5,5) * M(5,5)$. This process is called non-maxima suppression step and its objective is to remove noise pixels detected as corners and retain only one point/pixel at each corner. For that, a threshold (ζ) has to be applied on $m(x,y)$ (the corner metric image), delivering ones at corner points retained after a non-maxima suppression step and zero otherwise. In **Fig.** 4.3 an example of the feature extraction step is shown, a threshold $\zeta = 1 \times 10^5$ retained more than 15000 visual features (corners) after the non-maxima suppression step.

$$h(x,y) = \begin{cases} 1 & if\ \zeta < m(x,y) > b(x,y) \\ 0 & otherwise \end{cases} \tag{4.4}$$

where

$$b(x,y) = max(a)$$

$$a = m \circ M$$

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

(a)

(b)

Figure 4.3. Example of the feature extraction process. (a) Input. (b) Using the Harris corner metric response a thresholding operation ($\zeta = 1 \times 10^5$) delivers ones at corner points retained after a non-maxima suppression step.

4.2 Pixel tracking

In previous works, visual descriptors such as, SIFT [82], SURF [15], ORB [113] have been used to compute feature tracking in video sequences. Unfortunately, in order to get a robust feature tracking, these algorithms use high order metrics such as, the Jacobian or Laplacian of the patch that is processed, which limits the processing speed. To solve this problem, feature tracking should be performed only for a few pixels in the input image, generating a sparse tracking. To obtain dense tracking, as established in our hypothesis, in this work we propose a new pixel tracking algorithm which consists in an extension of the stereo matching problem. To achieve high performance for hardware architectures (FPGA/CUDA), a pixel-parallel/window-parallel approach based on a local correlation function is used. In order to improve the correlation performance, the curl of the intensity gradient as preprocessing step is proposed. In **Fig.** 4.4 an overview of the pixel tracking algorithm is shown. It consists of three steps in sequential form. The first step improves the input images robustness: let $(f_t(x,y), f_{t+1}(x,y)$ be two consecutive frames from a video sequence, the curl of the intensity gradient $\frac{df(x,y)}{d\mathbf{x}}$ are computed using **Eq.** 4.5. Let curl be a vector operator that describes the infinitesimal rotation, then, at every pixel the curl of that pixel is represented by a vector whose attributes (length and direction) characterize the rotation at that point. In our case, we use only the norm of $\mathbf{Curl}(x,y)$, as shown in **Eq.** 4.6. For implementation purposes, we divide the curl operation into two parts, first, image gradients ($G_{x(t)}$, $G_{y(t)}$, $G_{x(t+1)}$, $G_{y(t+1)}$) are computed, then, in the second part, the curl operation is completed.

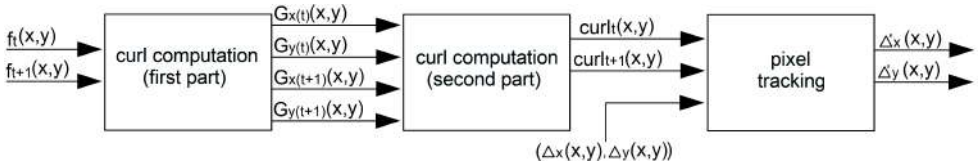


Figure 4.4. Formulation of the pixel tracking step.

$$\mathbf{Curl}(x,y) = \nabla \times \frac{df(x,y)}{d\mathbf{x}} = \frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial x} - \frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial y} \tag{4.5}$$

where

$$\frac{\partial f(x,y)}{\partial x} = G_x(x,y) = f(x{+}1,y){-}f(x{-}1,y), \quad \frac{\partial f(x,y)}{\partial y} = G_y(x,y) = f(x,y{+}1){-}f(x,y{-}1)$$

$$\frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial y} = G_y(x+1,y) - G_y(x-1,y), \quad \frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial x} = G_x(x,y+1) - G_x(x,y-1)$$

$$\overline{\mathbf{Curl}}(x,y) = |\frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial x} - \frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial y}| \tag{4.6}$$

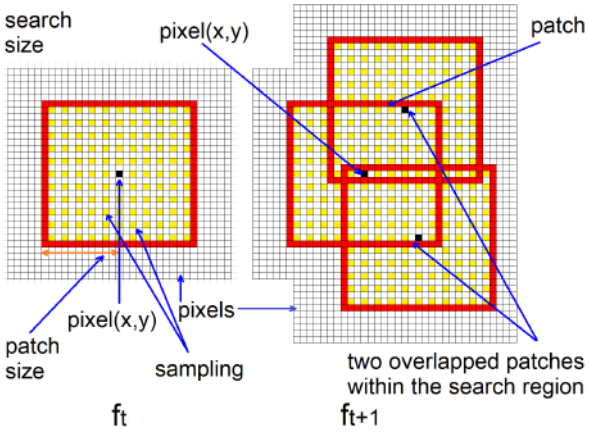
Given the curl images for two consecutive frames $\big(\overline{\mathbf{Curl}}_t(x,y)\ \overline{\mathbf{Curl}_{t+1}}(x,y)\big)$, dense pixel tracking $\big(\Delta'_x(x,y),\ \Delta'_y(x,y)\big)$, illustrated in **Fig.** 4.5b) in the reference image is computed as shown in **Fig.** 4.5a. This process assumes that pixel displacements between frames is such that it exists an overlap on two successive "search regions". A search region is defined as a patch around a pixel to track. Considering that between f_t and f_{t+1}, the image degradation is low, any similarity-based metric have to provide good accuracy. In our case, the similarity is computed by a SAD (Sum of Absolute Difference), **Eq.** 4.7; where r is the patch size (see **Fig.** 4.5a). $(\overline{\mathbf{Curl}}_t(x,y), \overline{\mathbf{Curl}_{t+1}}(x,y))$ are curl images on two consecutive frames. x, y are the spatial coordinates of pixels in f_t and, a, b are the spatial coordinates within a search region constructed in f_{t+1} (see **Eq.** 4.8 and 4.9); where $\Delta''_{x(t-1)}, \Delta''_{y(t-1)}$ are a dynamic search template, computed as shown in **Section** 4.4. k is the search size and s is a sampling value defined by the user. Finally, dense pixel tracking at the current time $(\Delta'_x(x,y),\ \Delta'_y(x,y))$ is computed by **Eq.** 4.10.

$$\mathrm{SAD}(a,b) = \sum_{u=-r,v=-r}^{u=r,v=r} |\overline{\mathbf{Curl}}_t(x+u,y+v) - \overline{\mathbf{Curl}_{t+1}}(x+u+a,y+v+b)| \tag{4.7}$$

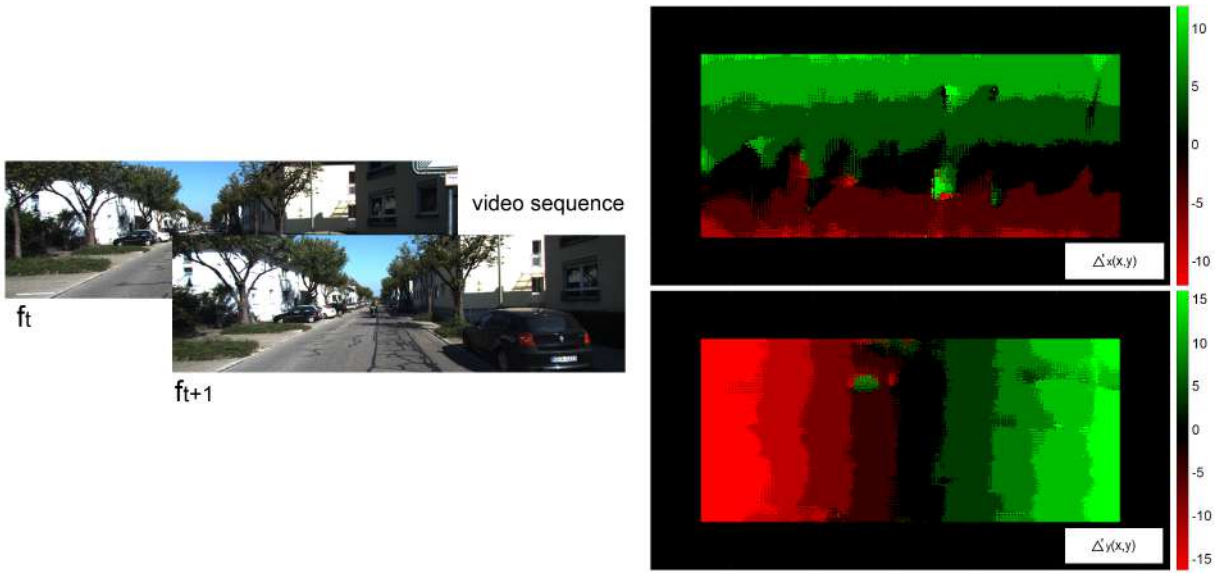
$$a = \Delta''_{x(t-1)}(x,y) - k : 1 : \Delta''_{x(t-1)}(x,y) + k, \tag{4.8}$$

$$b = \Delta''_{y(t-1)}(x,y) - k : 1 : \Delta''_{y(t-1)}(x,y) + k \tag{4.9}$$

$$[\Delta'_x(x,y), \Delta'_y(x,y)] = \mathbf{arg\ min}_{(a,b)} SAD(a,b) \tag{4.10}$$



(a)



(b)

Figure 4.5. The pixel tracking process. (a) The pixel-parallel/window-parallel formulation. For each pixel in the reference image f_t, n overlapped regions are constructed in f_{t+1}, then, n region center that minimizes a SAD correlation fucntion is the tracked position of the pixel(x, y) at f_{t+1}. (b) Pixel tracking example. Let f_t, f_{t1} be two consecutive frames of a video sequence; the pixel tracking components $(\Delta'_x(x, y),\ \Delta'_y(x, y))$ represents the 2D spatial displacements between the origin (f_t) and the current frame (f_{t1}).

4.3 Feature matching

The pixel tracking step is computed for all pixels in the reference image (**Section** 4.2), however, it does not consider the occlusion problem. To solve this problem, a feature matching step is used as filtering outliers. Let $h_{t+1}(x, y)$ be the features extracted in the search image, and $h_t(x, y)$ the features extracted in the reference image (see **Section** 4.1); we propose an outlier filter based on the hypothesis that a "good" feature has to be isolated from other key points (this should avoid the confusion in the matching process). For the mathematical formulation, each feature point and its surrounding neighbors are tested in order to quantify the number of features within a 3×3 neighborhood, see **Eq.** 4.11-4.12. Ten, a "good" feature point has to be associated with a unique feature at the center of the 3×3 neighborhood. On the other hand, to validate the pixel tracking robustness and to ensure not occlusion, we use the hypothesis that a "good" feature in $h_t(x, y)$ should have a corresponding feature in $h_{t+1}(x - \Delta'_x(x, y), y - \Delta'_y(x, y))$. This means that for each feature point in the reference image, there has to exist the same feature point (isolated from other key points) in the tracked position in $h_{t+1}(x, y)$. For the mathematical formulation see **Eq.** 4.13 - 4.14, where $\Delta_x(x, y), \Delta_y(x, y)$ are the feature matching being computed while $\Delta'_x(x, y), \Delta'_y(x, y)$ are the corresponding pixel tracking (**Section** 4.2). In **Fig.** 4.6 an example of feature matching in shown.

$$p_t(x, y) = \sum_{u=-1}^{u=1} \sum_{v=-1}^{v=1} h_t(x + u, y + v) \tag{4.11}$$

$$p_{t+1}(x - \Delta'_x(x, y), y - \Delta'_y(x, y)) = \sum_{u=-1}^{u=1} \sum_{v=-1}^{v=1} h_{t+1}(x + u, y + v) \tag{4.12}$$

$$\Delta_x(x, y) = \begin{cases} \Delta'_x(x, y) & if\ p_t(x, y) == p_{t+1}(x, y) == 1 \\ 0 & otherwise \end{cases} \tag{4.13}$$

$$\Delta_y(x, y) = \begin{cases} \Delta'_y(x, y) & if\ p_t(x, y) == p_{t+1}(x, y) == 1 \\ 0 & otherwise \end{cases} \tag{4.14}$$

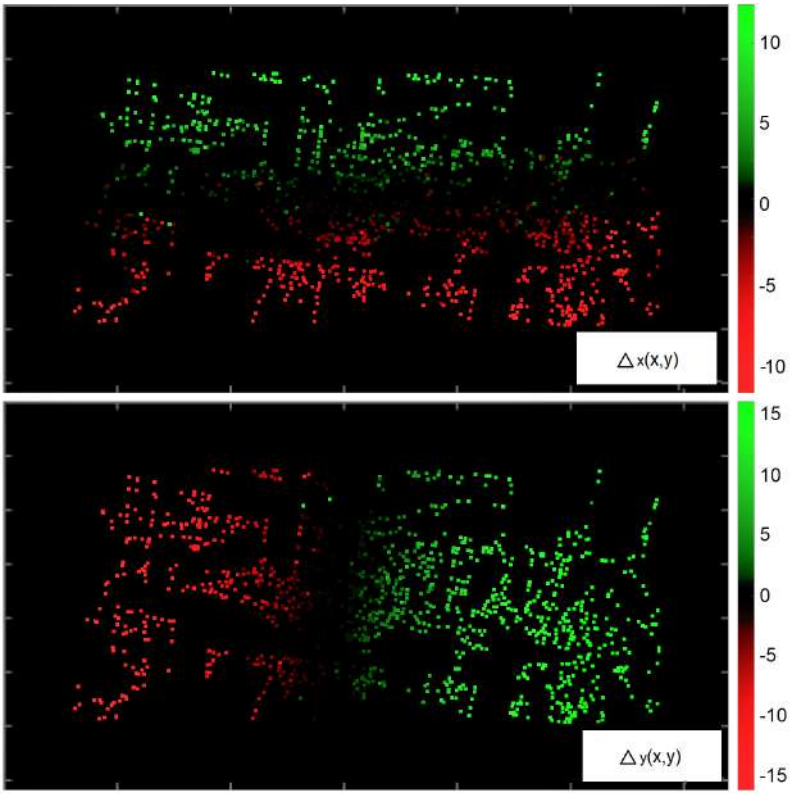


Figure 4.6. The feature matching process. Only pixel tracking for "good" features (corners) extracted in the reference image are retained.

4.4 Tracking template

Let p be a pixel in the reference image $(f_t(x_t, y_t))$ the same pixel in the tracked image $(f_{t+1}(x_{t+1}, y_{t+1}))$ has to satisfy $x_{t+1} \in x_t - k : 1 : x + k$, $y_{t+1} \in y - k : 1 : y_t + k$, where k is the search size for the pixel tracking step. In practice, large search size areas increase the tracking performance since feature tracking could be carried out in both slow and fast camera movements. However, large search windows decrease accuracy. On the other hand, small search size areas reach accurate and fast tracking but it is limited to slow camera movements. To address this problem we use the feedback of the previous feature matching step (**Fig.** 4.7) in a fashion that if camera movement in t_{-1} is slow, a fixed small size search window closer to the pixel being tracked (x_t, y_t) is used. On the other hand, for fast camera movements, a fixed small size search windows far to the pixel being tracked is defined. For implementation purposes we use a 9×9 search window since it provides a good tradeoff between robustness/accuracy and computational resources usage. For the

mathematical formulation, let define $\Delta_x(x,y), \Delta_y(x,y)$ as the feature matching for the time $t-1$ (see **Eq.** 4.13 - 4.14). Search template for the current time is computed as shown in **Eq.** 4.15 - 4.16, where k is the template size, in this case $k=9$, which means a window search area of 19×19.

$$\Delta''_x(x+u,y+v) = \sum_{u=-k,v=-k}^{u=k,v=k} (\textbf{mean} \sum_{u=-k,v=-k}^{u=k,v=k} \Delta_x(x+u,y+v)) \tag{4.15}$$

$$\Delta''_y(x+u,y+v) = \sum_{u=-k,v=-k}^{u=k,v=k} (\textbf{mean} \sum_{u=-k,v=-k}^{u=k,v=k} \Delta_y(x+u,y+v)) \tag{4.16}$$

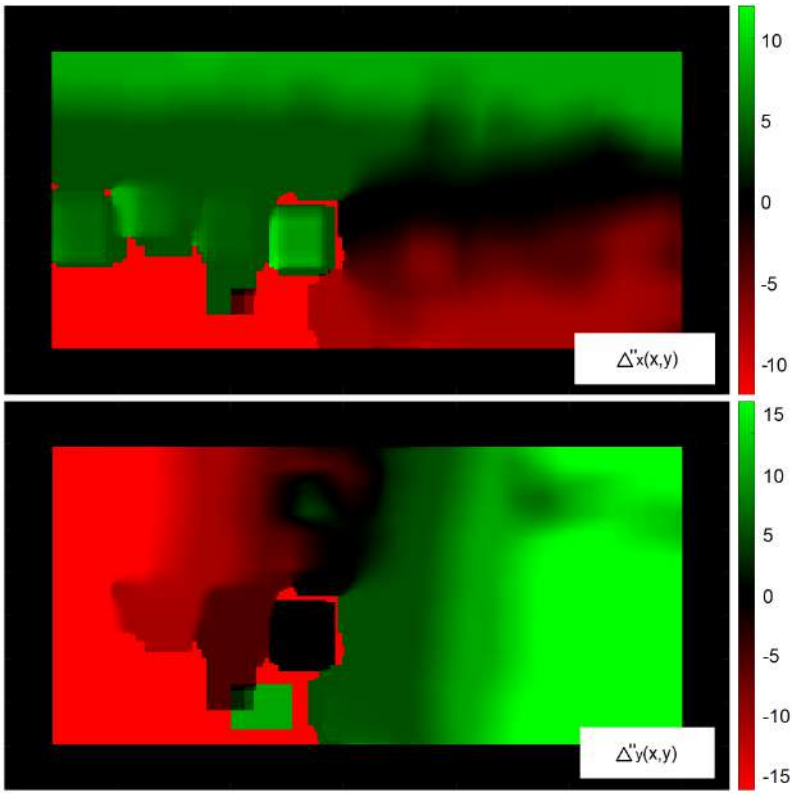


Figure 4.7. The tracking template process. This template is used as *a prior* knowledge for the current pixel tracking step. Then, pixels in the reference image are searched based on the predicted locations at the search image. This decreases the tracking confusion and guarantees an efficient hardware resources usage for FPGA/CUDA architectures.

4.5 Search parameters

In previous works the classical solution for the problem of visual odometry consisted of geometric methods applied to 3D points of the scene and its projection in the plane of the image [66, 102, 128, 138]. However, recent works [46, 54, 67, 71] have shown that there is some information such as optical flow, motion vectors, etc., that can be successfully used to develop simpler solutions. In this work, we propose dense feature matching as a linear/dependent pattern for the ego-motion estimation, as shown in **Fig.** 4.8. In previous works, geometric algorithms required feature matching for at least 5 characteristic points that were no closer and non-coplanar, and then, iterate until a "good" result for a geometric minimization function is achieved. Our approach uses the combination of dense features (more than 1000 features per image are matched) and this makes it possible to estimate the camera ego-motion without an iterative behavior and without geometric constrains. In practice, dense feature matching (based on previous works such as KTL [132], KL [83], Horn Schunck [63]) involves exhaustive operations and iterative behavior that limits its implementation into dedicated hardware. Similar to KTL and KL, our approach involves exhaustive operations, i.e., it makes relatively complex operations for all pixels in the input images, however, instead of the iterative optimization criterion used within the KL or KLT search windows, our algorithm is an extension of the stereo matching problem, that is, we compute a local correlation function instead of a local optimization. This makes possible for an efficient prallellization in FPGA/CUDA architectures. For the mathematical formulation, let $\Delta_x(x,y)$, $\Delta_y(x,y)$ be the feature matching for two consecutive frames (see **Fig.** 4.6), we propose eight different motion parameters $Q = [s_1, s_2 \dots s_8]$. These parameters are defined as the median value within the discrete histogram at eight different regions, as shown in **Fig.** 4.9 and, as defined in **Eq.** 4.17 and 4.18, where $H(\Delta(x,y)_R)$ are feature matching histograms for each regions in $(\Delta_x(x,y)$, $(\Delta_y(x,y))$.

$$\text{median}(H) = \frac{H_{\lfloor \#k/2 \rfloor} - H_{\lfloor \#k/2+0.5 \rfloor}}{2} \tag{4.17}$$

$$s_R = \text{median}(H(\Delta_x(x,y)_R)) \tag{4.18}$$

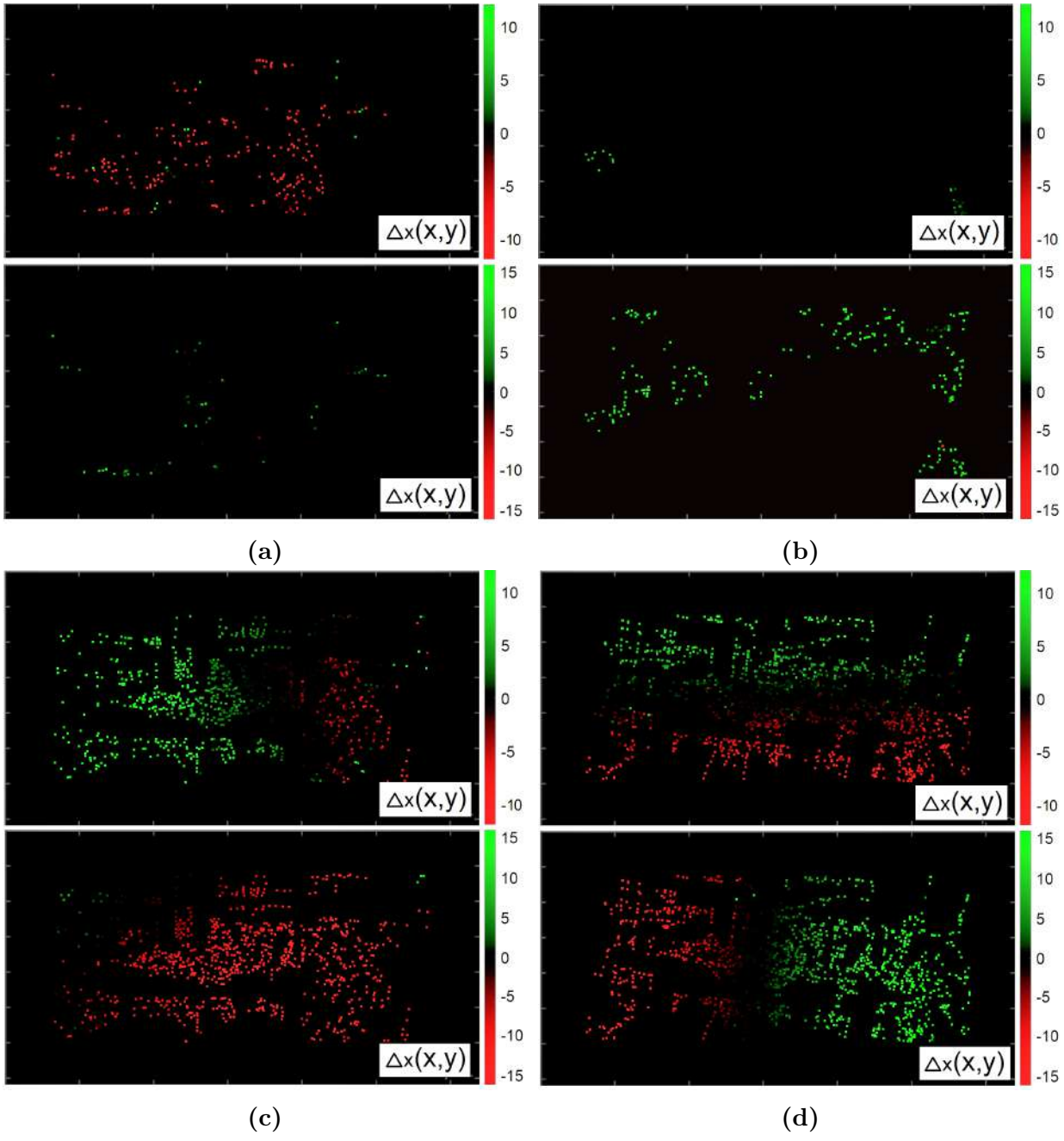
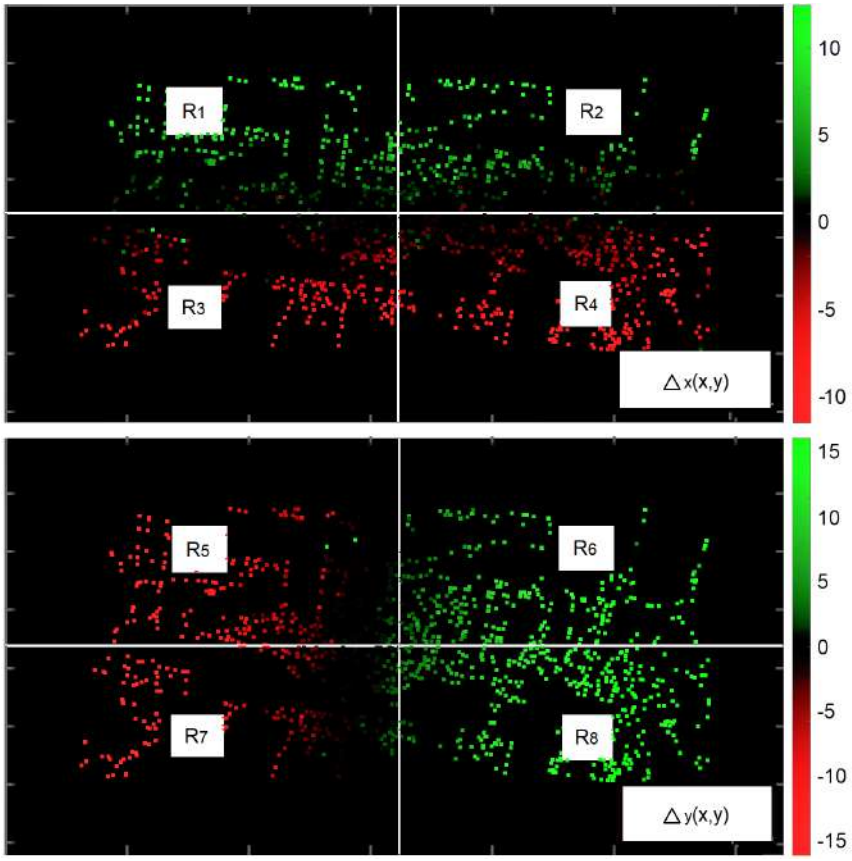
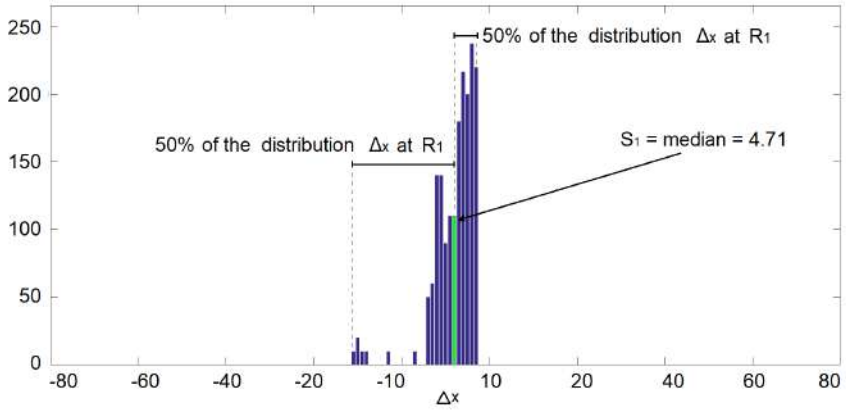


Figure 4.8. Hypothesis of the search parameters. Given dense feature matching as input, in all cases, unique motion patterns could be recover, for example: (a) Camera movement in the x axis deliver high (positive or negative) $\Delta_x(x, y)$ values and at the same time low $\Delta_y(x, y)$ values. (b) Camera movement in the y axis deliver low $\Delta_x(x, y)$ values and high $\Delta_y(x, y)$ values. (c) (d) Even for the rotation movements (α, γ) unique motion patterns can be recover



(a)



(b)

Figure 4.9. Computation of the motion parameters. (a) Input data for the motion parameters computation. In all cases, the feature matching images are divided in eight different regions ($R_1, R_2 \cdots R_8$). (b) Example of the motion parameter computation at R_1. The corresponding motion parameter (S_1), is defined as the median (the middle value in distribution) within the discrete histogram at R_1.

4.6 Lookup table

Let $Q = [s_1, s_2 \dots s_8]$ be the motion parameters that are linearly dependent to the camera movement, j the known camera displacements (obtained from datasets) that can be associated with its corresponding Q_j parameters using **Eq.** 4.19; where $C_j\{x, y, z, \alpha, \theta, \beta\}$ is the camera movement with six degree of freedom (known camera movements). Then, given Q_j, (C_j) non repeated elements in a lookup table, any unknown camera movement can be estimated as shown in **Eq.** 4.20 and 4.22; where Q' is the Q parameter for the pose being computed and C_k is the camera pose of the element that minimizes the absolute difference between the Q' and Q_j. Finally, in order to construct the lookup table, an element Q_j is included only if it satisfies $|Q(k) - Q_i(k)| < \sigma_1$, $|\alpha - \alpha_i < \sigma_2|$, $|\theta - \theta < \sigma_2|$, $|\beta - \beta < \sigma_2|$, where σ_1, σ_2 are threshold values defined by the user. High threshold values result in small lookup tables and this makes the search faster, however, accuracy due to the drift error is increased. On the other hand, low threshold values deliver accurate results but the search time is increased. For practical purpouses we recomend $\sigma_1 = 4, \sigma_2 = 0.5$ because these values deliver a good tradeoff between accuracy and speed. These values were empirically obtained and may be different for particular scenarios. As example, given the first training sequence of the KITTI dataset [50], which consist of a video sequence with 4541 frames, the full lookup table size has to be 4541. Using the proposed reduction and setting ($\sigma_1 = 4$ and $\sigma_2 = 0.5$), the lookup table size decreases to 1151 (near to 25% of the full size), as illustrated in **Table** 4.1.

$$Q_j \mathrel{\widehat{=}} C_j\{x, y, z, \alpha, \theta, \beta\} \tag{4.19}$$

$$\mathrm{T}_j = |Q_j - Q'| \tag{4.20}$$

$$k = \arg\min_j(\mathrm{T_j}) \tag{4.21}$$

$$\mathrm{pose}\{x, y, z, \alpha, \theta, \beta\} = C_k \tag{4.22}$$

Table 4.1. Example of a lookup table by applying the proposed approach. Setting $\sigma_1 = 4$ and $\sigma_2 = 0.5$, the lookup table size is 1151 (near to 25% of the full size).

	Q_j				C_j					
j	S_1	S_2	$\cdots$	S_8	x	y	z	α	θ	β
1	-1.87	7.37	$\cdots$	2.62	-0.04	-0.02	0.85	-0.02	-0.01	-0.11
2	-1.50	9.50	$\cdots$	2.50	-0.04	-0.02	0.85	-0.02	-0.01	-0.11
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1151	-6.12	1.25	$\cdots$	4.62	-0.05	-0.03	1.13	0.08	-0.04	0.04

4.7 Pose estimation

Let $pose_t$ be the camera ego-motion for the time t, computed by **Eq.** 4.22. This result is refined using a mean filter (**Eq.** 4.23-4.25). ν, μ are threshold values defined by the user. In order to improve the algorithmic implementation, **Eq.** 4.24 is implemented as the average value within a shift window, so, mean values are computed as stream. Finally, the current camera ego-motion Δ_m, is computed as in **Eq.** 4.26.

$$k = \min(\mathrm{T_j}) \tag{4.23}$$

$$pose_f = \frac{1}{4}\sum_{t=-4}^{t=-1} pose_t = mean(pose_{t-4:t-1}) \tag{4.24}$$

$$\{\alpha', \beta', \gamma'\} = |\{\alpha_{t-1}, \beta_{t-1}\gamma_{t-1}\} - \{\alpha_t, \beta_t\gamma_t\}| \tag{4.25}$$

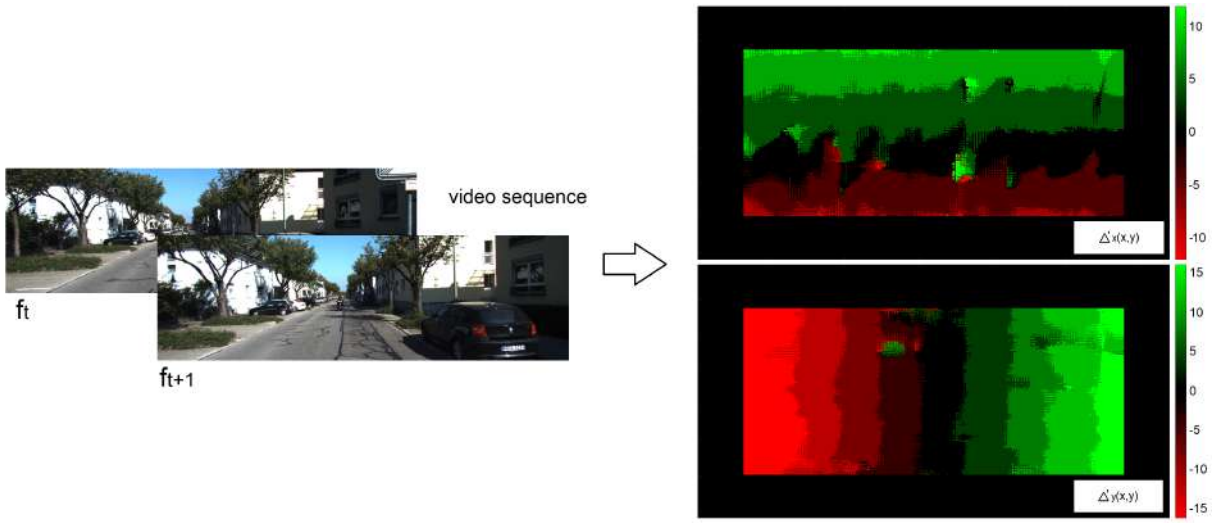
$$\Delta_m = \begin{cases} pose_t & k < \nu,\ \alpha' < \mu,\ \beta' < \mu,\ \gamma' < \mu \\ pose_f & otherwise \end{cases} \tag{4.26}$$

4.8 Depth from motion

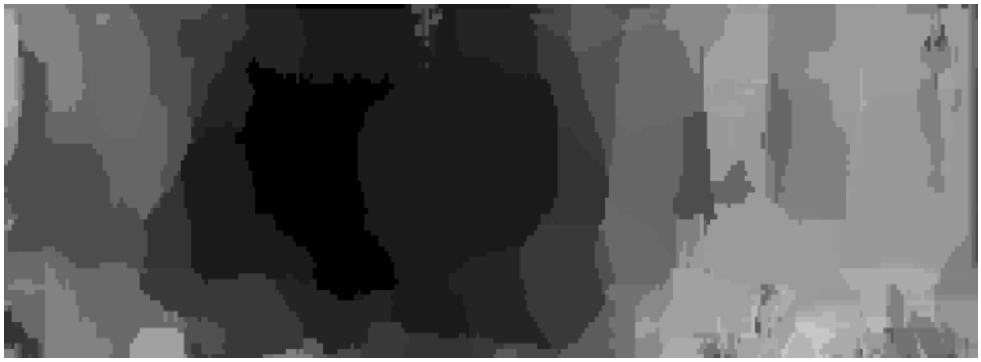
In the last decade, several works have demonstrated that depth information is highly useful for embedded robotic applications, such as intelligent surveillance, autonomous navigation for unmanned aerial vehicles, etc. [22, 59, 124]. In recent years, the most popular solution is the use of active vision to estimate the depth information of the scene [84, 114, 145], that is, LIDAR sensors or RGBD cameras that can deliver accurate depth maps in real time. However, those sensors increase the systems size and cost and are limited to indoor scenarios, in which the objects distribution and controlled illumination guarantees the correct propagation for the structured light. In order to reach high performance for embedded applications and high robustness for indoor/outdoor scenarios, in this work, we introduce a tracking/depth transformation inspired in the epipolar geometry. However, in order to recover the depth in the scene it is necessary to have assumptions about the scene and its 2D images. In the case of the epipolar geometry, it is assumed that the scene is rigid and they epipolar geometry error is close to zero.

In our case, the unique assumption is that the environment within the scene is rigid, then, given the pixel tracking for two consecutive frames (**Section** 4.2), we lay down the hypothesis that depth in the scene is proportional to the 2D pixel displacements (pixel tracking) between frames. That is, far objects must be associated with a low displacement values, while closest objects are associated with high displacement values. This could be considered as an extension of the epipolar geometry in which disparities values are proportional with the depth in the scene, as shown in **Fig.** 4.11. For the mathematical formulation, let $\Delta_x(x,y)$, $\Delta_y(x,y)$ be the pixel tracking at t time. Depth ($\mathbf{depth}(x,y)$) in the scene is computed using **Eq.** 4.27, where $\mathbf{depth}(x,y)$ is the norm of the pixel tracking, as illustrated in **Fig.** 4.10. This is Inspired by the Euclidean vector operations in which the Z component is proportional to norm of the X,Y components. Then, for a single moving camera, the depth in the scene z is proportional to the norm of the x,y components of the pixel tracking step.

$$\mathbf{depth}(x,y) = ||[\Delta_x(x,y), \Delta_y(x,y)]|| = \sqrt{\Delta_x(x,y)^2 + \Delta_y(x,y)^2} \tag{4.27}$$



(a)



(b)

Figure 4.10. The depth from motion process. (a) Input data for the depth from motion computation. We lay down the hypothesis that depth in the scene is proportional to the $\Delta'_x(x,y), \Delta'_y(x,y)$ of the pixel tracking computed from two adjacent frames (f_t, f_{t+1}). (b) Example of depth estimation. We propose the norm of the $\Delta'_x(x,y), \Delta'_y(x,y)$ components as linear dependent of the depth in the reference image (f_t).

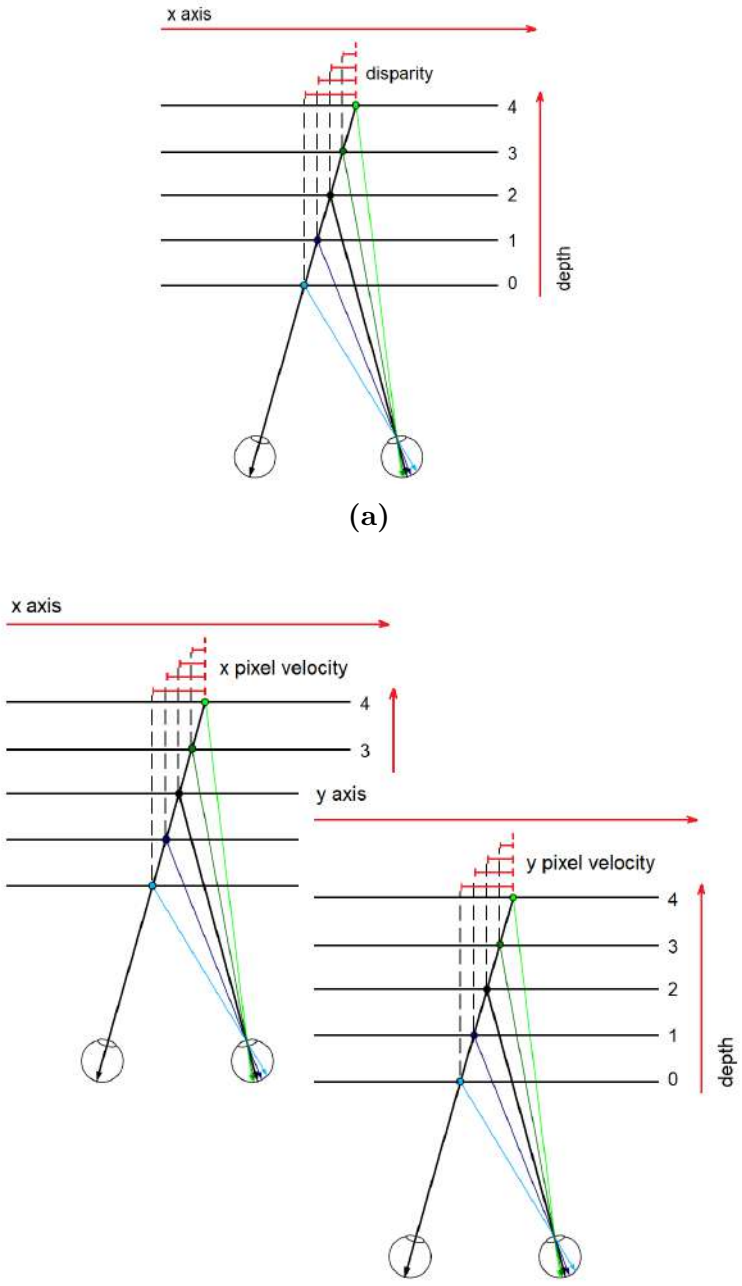


Figure 4.11. Formulation of the depth from motion step. (a) Epipolar geometry: depth in the scene is proportional to the disparity value, i.e., far objects have low disparity values while closer objects are associated with high disparity values. To compute the disparity map (disparities for all pixels in the image) a stereo pair (two images with epipolar geometry) are needed. (b) Single moving camera: in this work we suppose that depth in the scene is proportional to the pixel velocity across the time. To compute the pixel velocity, optical flow across two consecutive frames has to be computed.

4.9 Linear triangulation and map construction

In order to get the scene reconstruction (map of the SLAM problem), it is necessary to triangulate points/pixels whose spatial location for at least two different viewpoints (feature matching) are known or points/pixels whose relative depth (depth with unknown scale factor) is known. In our case, we have the relative depth for all pixels in the scene (obtained via depth from motion, **Section** 4.8), then, dense scene reconstruction can be estimated. Let $\mathbf{depth}(x, y)$ be the depth from motion for the current frame, while x, y be the spatial locations within the frame, all corresponding pixels within the frame are triangulated using **Eq.** 4.28 - 4.34. First, the undistorted pixel coordinates are computed as shown in **Eq.** 4.30 - 4.31, where K is the camera calibration matrix (it can be obtained using camera calibration algorithms [93] within the monocular-SLAM initialization step). The scale correction matrix is computed by **Eq.** 4.33, where the scale factor (β) has to be estimated at the monocular-SLAM initialization step. Finally, the real world coordinates (3D reconstruction) are obtained via the linear triangulation algorithm [58], using **Eq.** 4.34. Then, given local 3D reconstructions (for two consecutive frames) the map construction step bundles them in a single global 3D reconstruction. For that, we use the Iterative closest point (ICP) algorithm [19]. Finally, the global 3D reconstruction (map) can be displayed.

$$xy_1(x, y) = [x, y, 1] \tag{4.28}$$

$$xy_2(x, y) = [x - \mathbf{depth}(x, y), y, 1] \tag{4.29}$$

$$K_1(x, y) = K^{-1} \times xy'_1(x, y) \tag{4.30}$$

$$K_2(x, y) = K^{-1} \times xy'_2(x, y) \tag{4.31}$$

$$A(x,y) = [K_1, -K_2] \tag{4.32}$$

$$\mathbf{sF}(x,y) = (A'(x,y) \times A(x,y)^{-1} \times (A(x,y) \times [\beta, 0, 0]') \tag{4.33}$$

$$xyz(x,y) = \begin{cases} K_L(x,y) \cdot \mathbf{sF}(x,y)[1], & \mathbf{depth}(x,y) > 0 \\ 0 & otherwise \end{cases} \tag{4.34}$$

4.10 Performance of the proposed algorithm

In this section, simulation results of the proposed algorithms are shown. We present results for the proposed camera pose estimation algorithm (localization) and for the linear triangulation (mapping) step. For that, we implement our algorithm in MATLAB 2017b. In practice, the simulation results are not feasible and suitable for any real world application since our algorithm was formulated for parallel implementation using hardware architectures, such as FPGA or CUDA. Then, any CPU-based implementation, could deliver low processing speed, limiting the real-rime processing of the whole system. So, the aim of this section is to obtain early results for the proposed algorithm and compare them with the current state of the art; demonstrating the effectiveness of the proposed approach regarding to localization accuracy and mapping density. In addition, the results presented in this sections will be used as reference for the expected performance (in terms of localization accuracy and mapping density) of the developed hardware implementations (based on FPGA and CUDA, respectively) which will be presented in the following chapters.

For the visual odometry estimation, the KITTI dataset [50] provides 11 training sequences (00-10) with public truth while another 11 sequences (11-21), without public ground truth, that will be used for our evaluations. In an early experiment, we carried out a cross validation for the training sequences, i.e., we built the lookup table using all sequences in the training set, except the sequence that is being evaluated, (see **Table** 4.2). Given the training sequences and given their corresponding ground truth, 21732 Q_j elements are available. For $\sigma_1 = 4, \sigma_2 = .5$, a lookup table of 13791 elements is used.

This configuration delivers accuracy around 97%. For $\sigma_1 = 8, \sigma_2 = 1$, average accuracy of 95% is achieved, in this case, the lookup table was reduced to 7371 elements. Finally, for $\sigma_1 = 15, \sigma_2 = 2$, a lookup table with 1749 elements was used, average accuracy around to 84% For all cases, a relatively high accuracy was obtained. In particular, for $\sigma_1 = 4, \sigma_2 = .5$, $\sigma_1 = 8, \sigma_2 = 1$, accuracy around 96% is possible. However, for all cases, a low speed processing is required (more than 11 hours per video sequence). In **Fig.** 4.12, we present qualitative results for the "Sequence00" of the KITTI dataset. In **Table** 4.3 quantitative comparisons with the current state of the art are shown. In most cases, our algorithm outperforms previous works in terms of localization accuracy which demonstrates the effectiveness of the proposed approach.

Table 4.2. Early results for the KITTI dataset. Large look-up tables ($\sigma_1 = 4, \sigma_2 = .5$) deliver error lower than 3% but the look-up table size is high. Look-up tables using 10-15% of the training data ($\sigma_1 = 8, \sigma_2 = 1$), deliver a good tradeoff between accuracy and look-up table size (mean error of 5%).

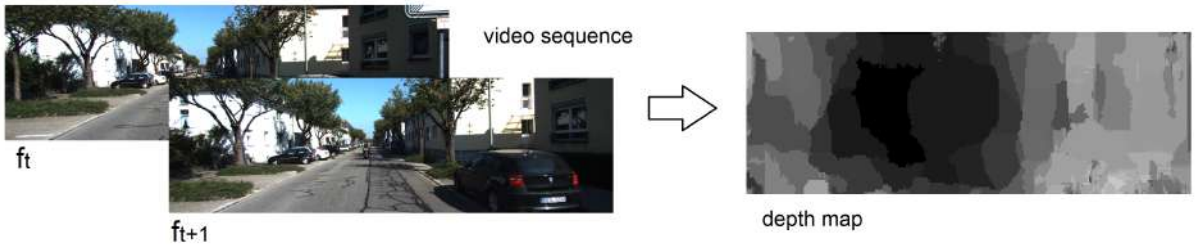
	Accuracy		
	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
00	97.47	95.83	84.65
01	97.48	95.13	84.07
02	97.16	96.35	85.99
03	97.58	95.73	84.15
04	97.92	95.22	84.88
05	97.27	95.62	85.21
06	97.21	95.46	84.92
07	97.09	96.12	84.00
08	97.13	95.11	85.54
09	97.35	95.93	85.63
10	97.83	96.35	84.73

Figure 4.12. Performance for the KITTI dataset. Sequence 00, $\sigma_1 = 4, \sigma_1 = .5$, accuracy = 97.47 %.

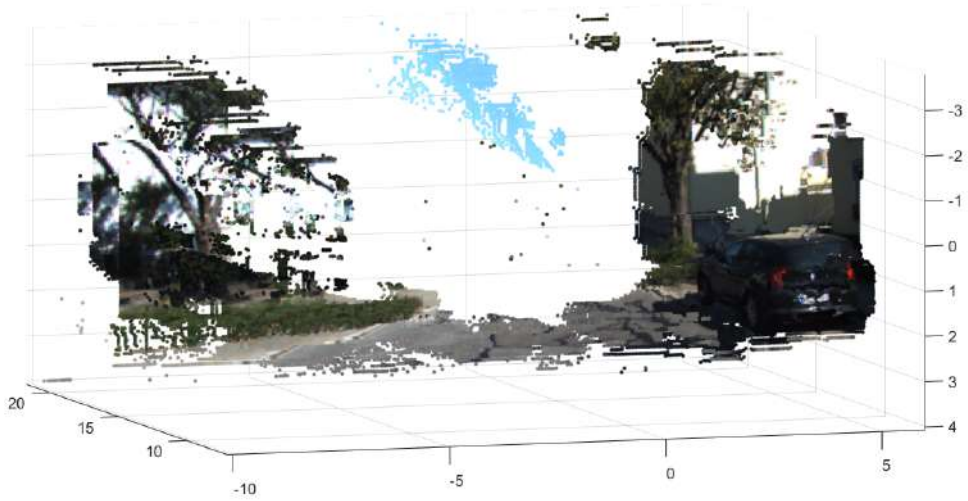
Table 4.3. Early results of the proposed algorithm compared with previous works. In most cases, our algorithm outperforms previous works in terms of localization accuracy.

Algorithm	**Accuracy**
Geiger et al (2011) [49]	83.71%
Ciarfuglia et al (2014) [30]	85.56%
Costante et al (2016) [34]	91.04%
Costante et al (2016) [34]	91.04%
Mohanty et al(2016) [91]	94.50%
Holzmann et al (2016) [61]	91.94%
Weber et al (2017) [139]	88.53%
Pillai and Leonard (2017) [104]	99.72%
This work	97.37%

For the mapping performance, **Fig.** 4.13 shows an example of 3D reconstruction using our approach. Previous works triangulate the depth of 2 to 7% of all image pixels, while ours triangulate 80% of the image pixels. Then, our algorithm improves about 15 times (in terms of mapping density) the current state of the art which demonstrates the effectiveness of the proposed approach.



(a)



(b)

Figure 4.13. Performance of the mapping step. The KITTI dataset: Sequence 00. (a) Our depth from motion algorithm (**Section** 4.8) provides dense depth maps and this improves the mapping density. (b) 3D reconstruction by the proposed approach. Compared with previous works, our algorithm improves them about 15 times in terms of mapping density.

4.11 Summary

In this Chapter, details about the proposed algorithms were presented. For the feature extraction step, we have developed a parallel version of the Harris corners detection algorithm suitable for embedded applications. Further, we have proposed a novel camera pose estimation approach that consists of using look-up tables and a new feature matching algorithm.

For the for the estimation of the pose., we have proposed dense feature matching as linear/dependent pattern for the estimation of the pose. In previous works, geometric algorithms require feature matching for at least 5 no-closer and non-coplanar feature points, and then, iterate until a "good" result for a geometric minimization function is achieved. In this work, our approach uses dense feature matching (more than 1000 features per match) and this makes it possible to estimate the camera pose in a direct form, that is, without iterative behavior and without geometric constrains.

For the feature matching algorithm, a new feature tracking / pixel tracking algorithm which consists in an extension of the stereo matching problem was proposed. A pixel-parallel/window-parallel approach based on a Sum of Absolute Difference was proposed. Further, in order to improve the correlation performance, the curl of the intensity gradient as preprocessing step was proposed. Finally, to recover depth in the scene, we propose the norm of the pixel tracing (optical flow) as a linear dependent to the depth. This was Inspired by the Euclidean vector operations in which the Z component is proportional to the norm of the X, Y components. So, for a single moving camera, we supposed that depth in the scene z is proportional to the norm of the x, y pixels velocities across the time.

Chapter 5

LT-SLAM: GPU implementation

A general description of the developed GPU architecture is shown in **Fig.** 5.1. Feature extraction, pixel tracking, feature matching, template tracking, search parameters, lookup table, depth from motion and linear triangulation steps are parallelized and computed inside GPU dedicated hardware, while map construction and pose estimation steps are computed in a sequential processor (CPU). For all the experiments, we implemented our algorithm in an Alienware 15 laptop, Intel Core i7-4710HQ @ 2.5 GHz (CPU), GTX 970M (GPU) with 1280 CUDA cores and 1024 max treads per block.

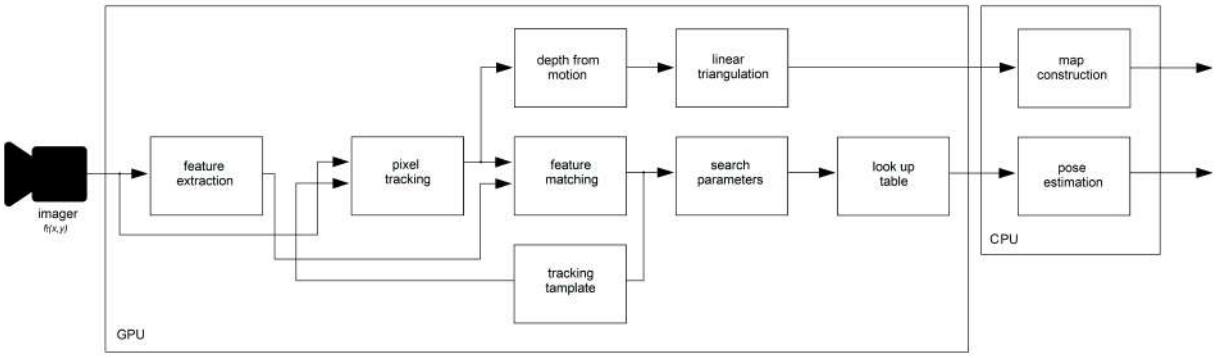


Figure 5.1. Block diagram of the GPU implementation. All exhaustive operations are parallelized and computed inside GPU dedicated hardware, while non parallelizable operations, such as map construction and pose estimation are computed in a sequential processor (CPU).

5.1 Definitions

In order to simplify the mathematical formulation, all the equations are presented in the classical form (using matrices and 2D dimension notation), however, some definitions regarding to the GPU parallelization have to be established:

1. In all cases the maximum parallelization level is defined as $N = \lfloor (Y/\sqrt{T}) \rfloor * \lfloor (X/\sqrt{T}) \rfloor$; where X, Y is the input image resolution and T is the maximum threads per block (defined by the GPU used for implementation).

2. In all cases, 2D image coordinates (x, y) are mapped into a single dimension (K), suitable for CUDA implementations. For this purpose we define $v(K) = v(x + X * y)$ as the mapping into a single dimension K for a X, Y input dimensions; where X is the horizontal 2D resolution and x, y are 2D spatial coordinates.

5.2 Feature extraction

For the feature extraction step, our GPU implementation consists of three CUDA kernels executing in sequential form, as shown in **Fig.** 5.2. For the first CUDA kernel: given an input image $f(x, y)$, horizontal and vertical gradients $(G_x(x, y),\ G_y(x, y))$ and image derivatives $(A(x, y),\ B(x, y),\ C(x, y))$ are parallelized in all threads in the grid, as shown in **Fig.** 5.3a. For the second kernel, the Gaussian filtering and the corner metric response steps are parallelized as shown in **Fig.** 5.3b. Finally, in the third CUDA kernel, corner detection process is parallelized in all threads in the grid, as shown in **Fig.** 5.3c. In all cases, 2D images (x, y) are mapped into a single dimension vector (N), suitable for CUDA implementation. For the detailed mathematical formulation see **Section** 4.1. For the parallelism level, in all cases N pixels are convolved and multiplied in parallel, please see the mathematical definitions lay down at the beginning of this section.

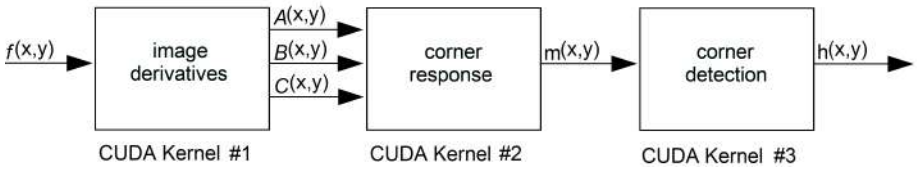


Figure 5.2. GPU implementation for the feature extraction step. Three CUDA kernels are executing in sequential form. Each kernel parallelize a different part of the feature extraction formulation.

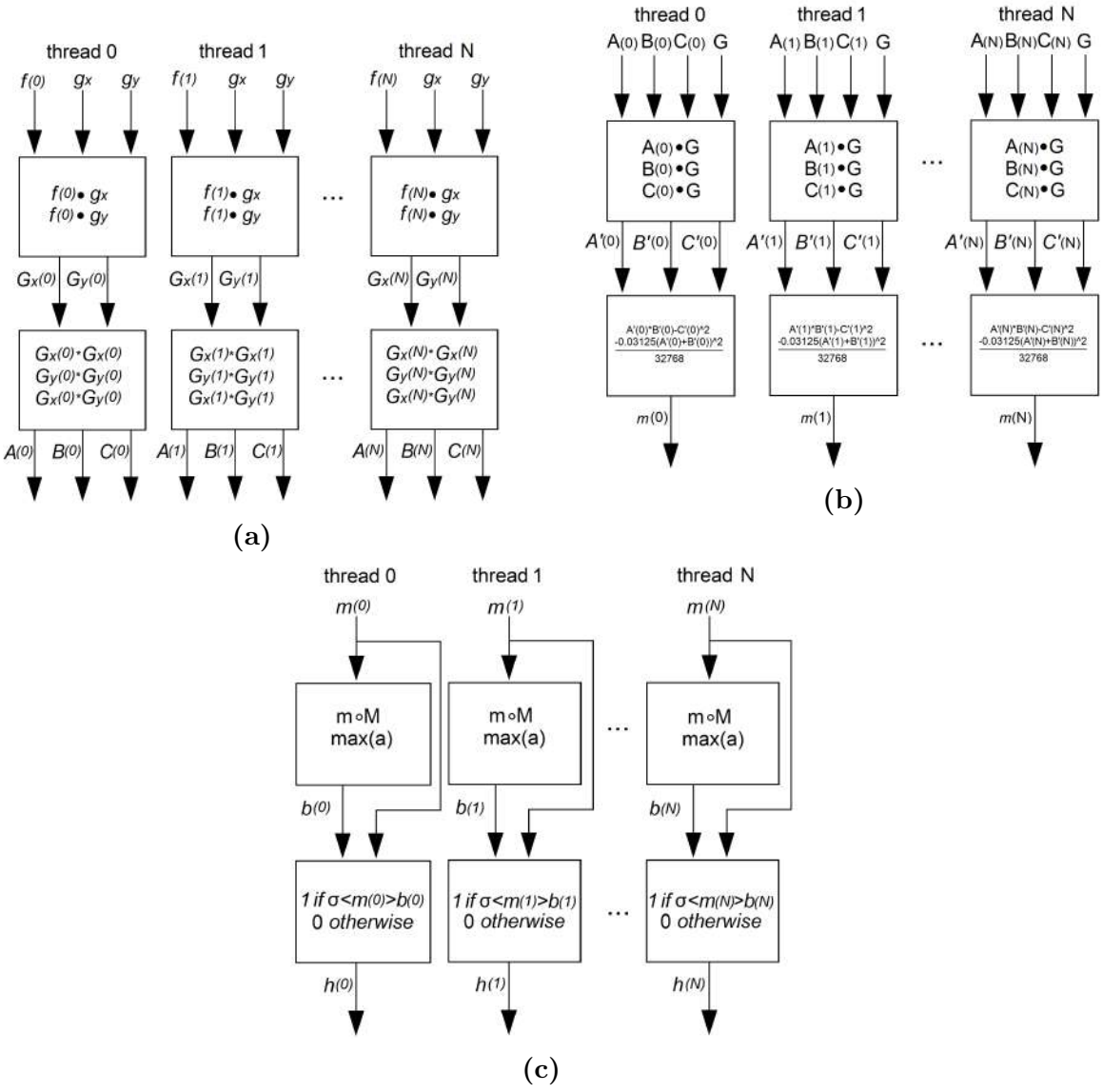


Figure 5.3. GPU parallelization for the feature extraction step. For all CUDA kernels, N pixels are processed in parallel (please see the mathematical definitions lay down at the beginning of this section.). That means, image derivatives for N different pixels are computed in parallel (a), same for the Corner metric response (b) and the corner detection process (c). In all cases, the 2D convolution operation is computed sequentially.

5.3 Pixel tracking

In **Fig.** 5.4 an overview of the developed GPU architecture is shown. It consists of three CUDA kernels launched in sequential. We divide the curl operation into two CUDA kernels, first, image gradients ($G_{x(t)}$, $G_{y(t)}$, $G_{x(t+1)}$, $G_{y(t+1)}$) are computed. Then, in the second CUDA kernel, the curl operation is completed. For the parallelism level, this operation is parallelized in all threads in the grid, as shown in **Fig.** 5.5a and 5.5b. Finally, in the third CUDA kernel, the pixel tracking process (**Fig.** 4.5a) is parallelized in all threads in the grid, as shown in **Fig.** 5.5c. In all cases, 2D images (x, y) are mapped into a single dimension vector (N), suitable for CUDA implementation. For the detailed mathematical formulation see **Section** 4.2. For the parallelism level, in all cases N pixels are convolved and multiplied in parallel, please see the mathematical definitions lay down at the beginning of this section.

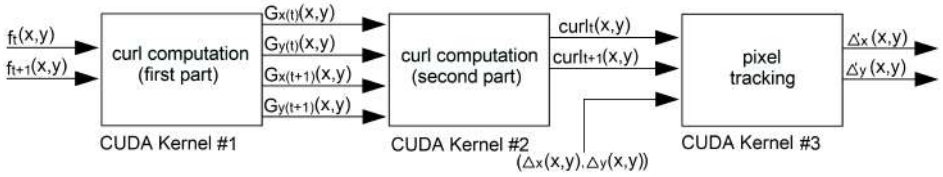


Figure 5.4. GPU implementation for the pixel tracking step. Three CUDA kernels are executing in sequential form. Each kernel parallelize a different part of the feature extraction formulation.

5.4 Feature matching & Tracking template

For the feature matching and tracking template steps, our mathematical formulations (**Sections** 4.3 - 4.4) are parallelized using all threads in the grid, as shown in **Fig.** 5.6a-5.6b, respectively.

5.5 Search parameters & Lookup table

We use CUDA atomic operations in order to get the discrete histogram for each of the eight motion parameter previously proposed (**Fig.** 4.9), see **Fig.** 5.6c. Then, the motion parameters value (the median value of the k discrete histograms) are computed using a **mean** function. For the k variable, it defines the index for the k motion parameter and

it is computed by an **if** statement inside each parallel thread. On the other hand, for the lookup table step, (**Section** 4.6) we parallelized it as shown in **Fig.** 5.6d. First, absolute differences between the motion parameters and the N elements in the look up table are computed in parallel, then, the minimum value of the N absolute differences is computed. For that, we use the MATLAB **min** function since for a small lookup table size (lower than 10000 elements), it has a "good" performance (in terms of processing speed but without GPU hardware requirements). Considering that in most cases our lookup table size is between 5000 to 8000 elements, the MATLAB **min** function should deliver a good tradeoff between computational resources consumption and processing speed. Of course, for a larger lookup table size ($> 10,000$), a CUDA parallel reduction should be more efficient (in terms of processing speed). Finally, the camera motion associated with the minimum value (C_{min}) is the current camera ego-motion (the relative camera pose for the current frame). For the detailed mathematical formulation see **Sections** 4.5 and 4.6.

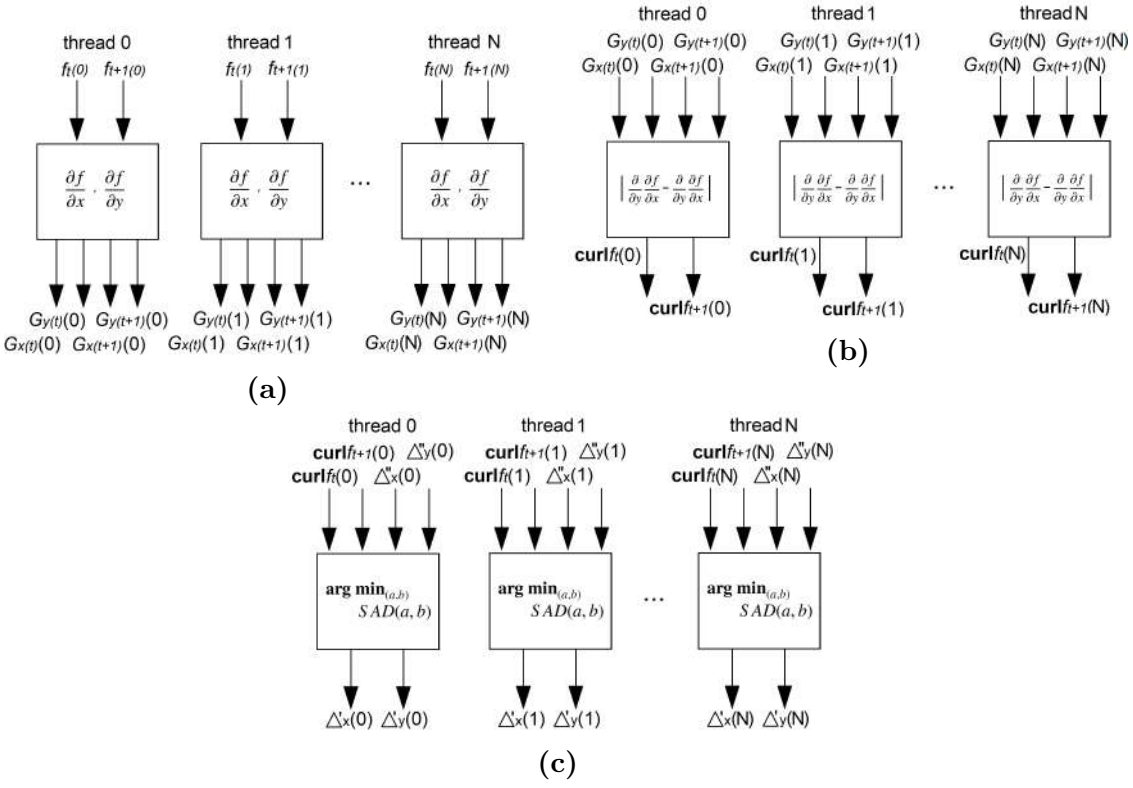


Figure 5.5. GPU parallelization for the pixel tracking step. For all CUDA kernels, N pixels are processed in parallel. i.e., image derivatives and **Curl** for N different pixels are computed in parallel (a) and (b), respectively. Same for the pixel tracking process in which N different pixels are processed in parallel.

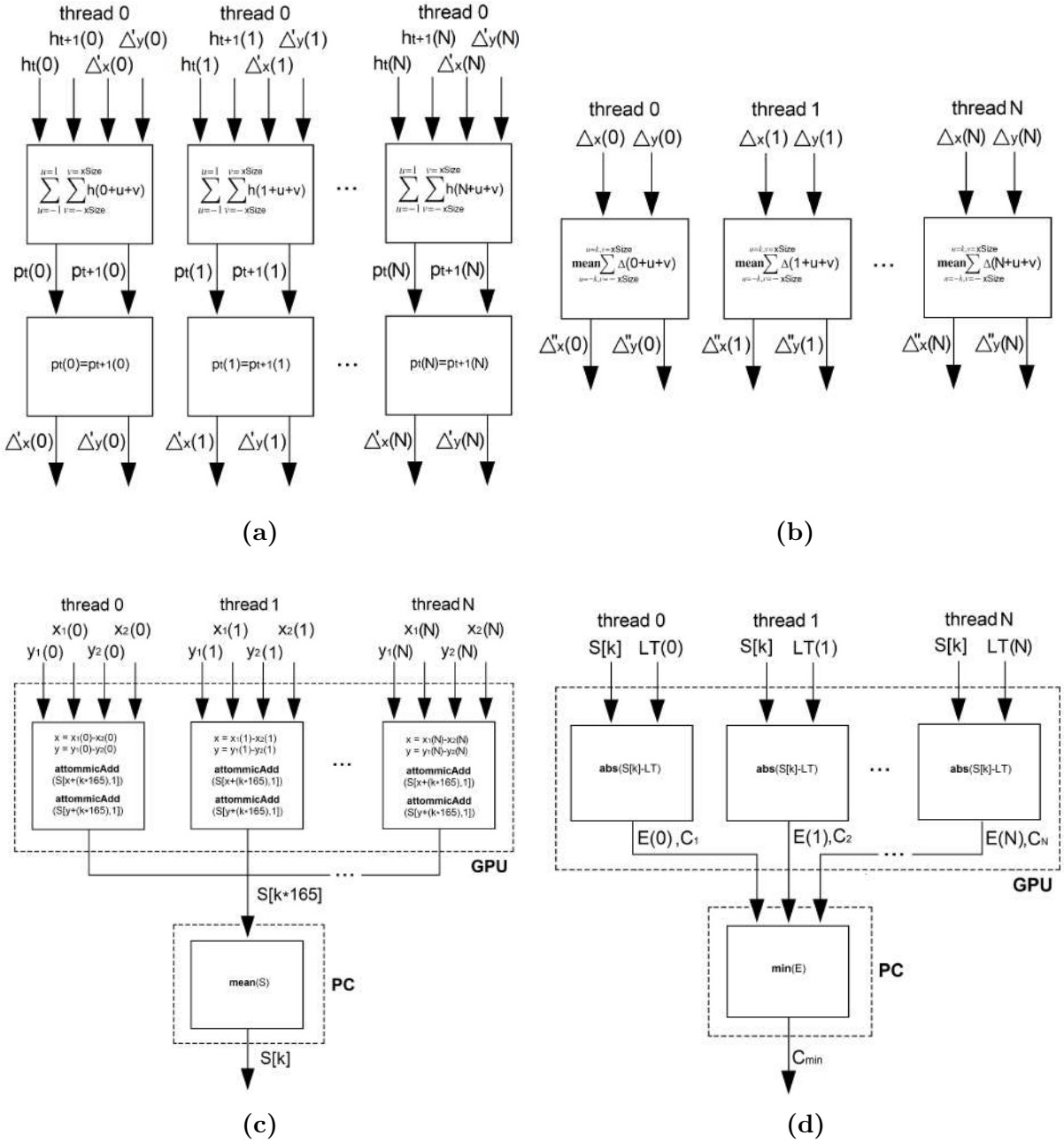


Figure 5.6. GPU parallelization for the feature matching, tracking template, search parameters and lookup table. For the feature matching and tracking template ((a) and (b), respectively), N pixels are processed in parallel. For the search parameters (c), CUDA atomic operations and a CUDA-based **mean** function compute the eight search parameters proposed by this work. Finally, for the lookup table (d), absolute parallel differences and a **minimum** MATLAB function compute the current camera ego-motion.

5.6 Depth from Motion

For the Depth from Motion step, our mathematical formulation (**Sections** 4.8) is parallelized using all threads in the grid, as shown in **Fig.** 5.7. that is, **Eq.** 4.27 is evaluated N times in parallel and, therefore, the depth in the scene for N pixels are estimated in parallel.

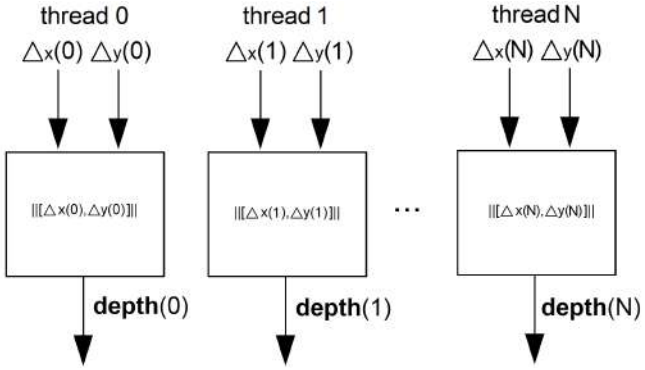


Figure 5.7. GPU implementation for the depth from motion step. N pixels are processed in parallel, therefore, depth for N different pixels are computed simultaneously.

5.7 Linear triangulation

In **Fig.** 5.8 an overview of the developed GPU architecture is shown. First, the undistorted pixel coordinates are computed. Then, the scale correction matrix is computed, for that several matrix multiplications and inverse operations are computed sequentialy. Finally, the real world coordinates are obtained by multiplying the correction matrix with the spatial 2D coordinates. For the detailed mathematical formulation see **Section** 4.9

5.8 Performance and limitations

In this section, the results of the GPU implementation of the proposed algorithms are shown. First, we present results for our pixel tracking algorithm. The results for the feature matching step are shown below. Finally, we present the results for the proposed camera pose estimation algorithm (localization) and for the linear triangulation (mapping) step.

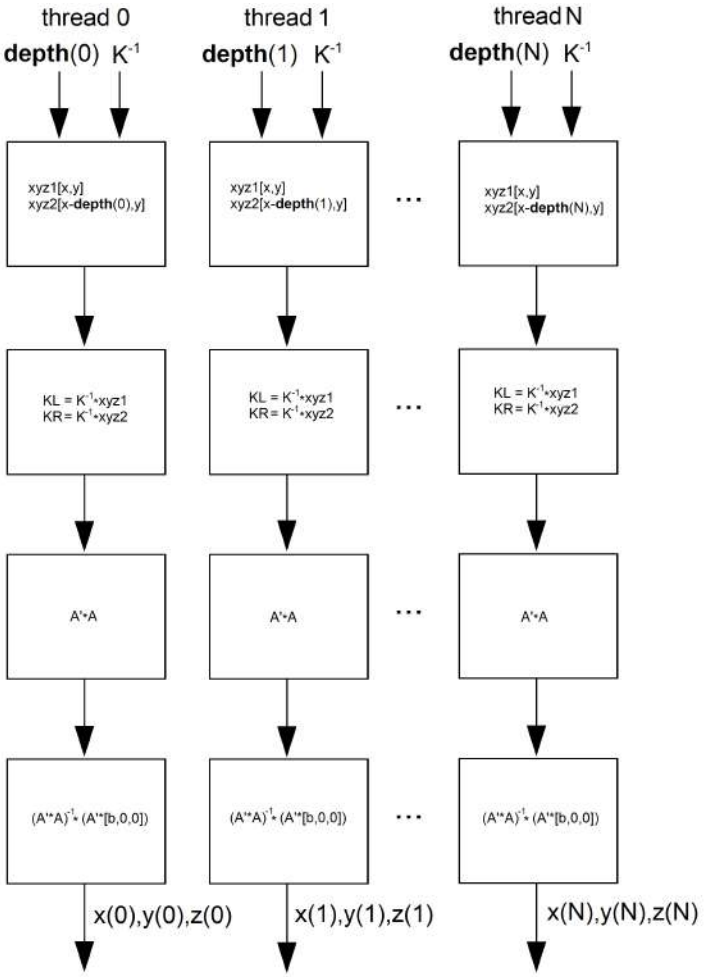


Figure 5.8. GPU implementation for the linear triangulation step. N pixels are processed in parallel, then, N different pixels are triangulated in parallel.

5.8.1 The pixel tracking step: performance and limitations

In **Fig.** 5.9, quantitative and qualitative results for our pixel tracking algorithm compared with previous work are shown. To carried out these comparisons, we use the KITTI optical flow dataset [134] since previous GPU-based algorithms [36, 65, 90, 109, 115, 149] used this dataset as reference. In order to estimate the error, we codified the pixel tracking result as a flow map [14], then we compute and compare the RMS error (our algorithm reaches an RMS error equal to 4.91%). When compared with previous work (**Fig.** 5.9), our algorithm provides high performance under real world scenarios, it reach similar accuracy then several previous work [65, 90, 149] and outperforms other GPU-based feature tracking algorithms [36, 109, 115].

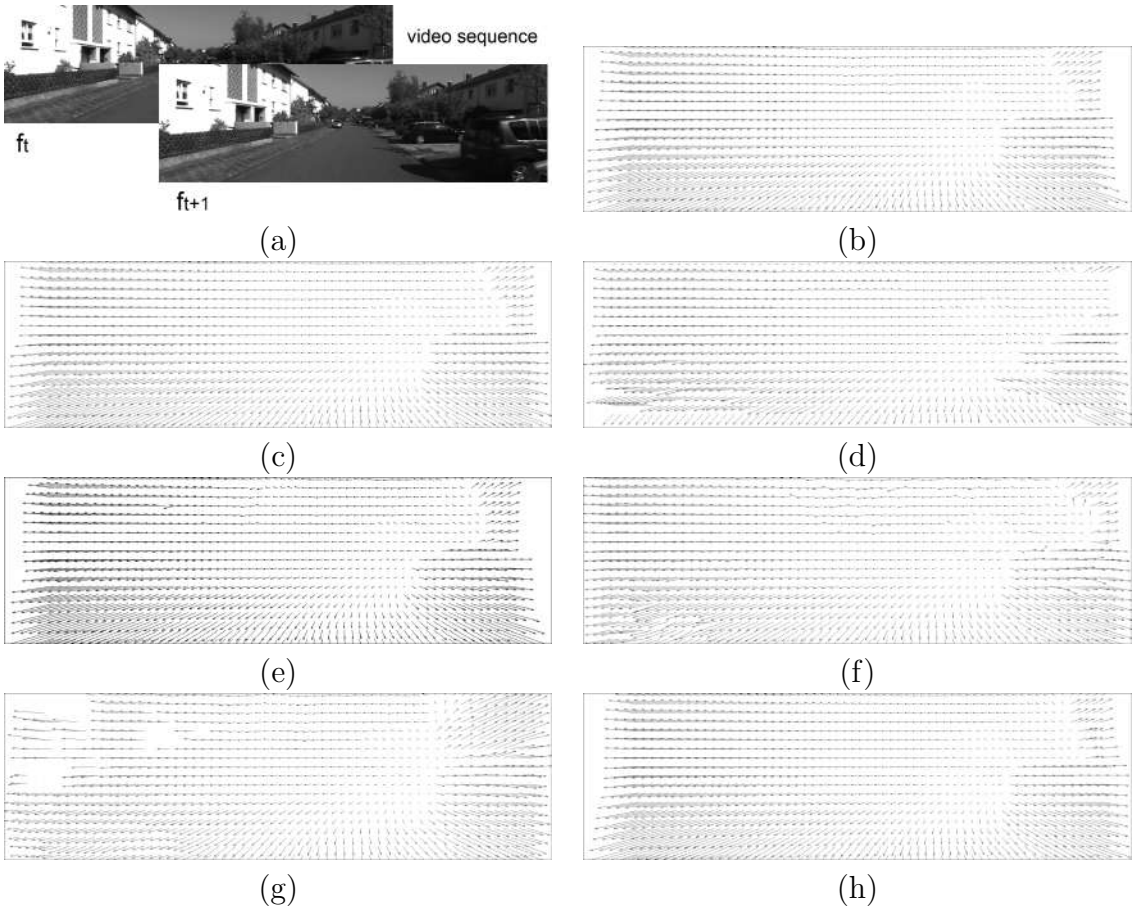


Figure 5.9. Accuracy performance for different GPU-based pixel tracking algorithms. (a) Input data. (b) Ground truth. (c) Hui et al. [65] (error = 3.27 %). (d) Meister et al. [90] (error = 4.28%). (e) Zweig and Wolf [149] (error = 4.94%). (f) Demetz et al. [36] (error = 6.52%). (g) Plyer et al. [109] (error = 19.31%). (h) This work (error = 4.91%).

5.8.2 The feature matching step: performance and limitations

In **Table** 5.1, accuracy comparisons with several feature-matching algorithms previously used in SLAM formulations are shown. In the case of the SURF/ORB [15, 113] algorithms, the image degradation between viewpoints introduces data inconsistences that introduce erroneous matches. For the KLT algorithm, accurate tracking can be reached, however, previous works have demonstrated that iterative operations inside the original KLT formulation limits the processing speed [28]. For our algorithm, it allows high accuracy, superior to SURF/ORB algorithms and with the capability to be implemented inside GPU devices. This makes possible to reach real-time processing, with higher processing speed than KLT-based algorithms.

Table 5.1. Accuracy of feature-matching algorithms used in SLAM formulations (errors are measured in pixels).

TUM dataset [126]	**SURF [15]**	**ORB [113]**	**KLT [83]**	**proposed**
fr1/room	79.38	76.24	0.21	1.97
fr2/desk	81.12	73.63	0.45	1.76
fr1/plant	77.74	75.24	0.39	1.83
fr1/teddy	83.53	76.73	0.47	1.94
fr2/coke	80.78	75.28	0.32	1.73
fr2/dishes	78.25	74.19	0.01	1.67
fr3/cabinet	79.87	74.02	0.42	1.71
fr3/teddy	79.10	75.21	0.46	1.83
mean error =	79.97	75.08	0.34	1.63

5.8.3 The pose estimation step: the proposed dataset

For evaluation purposes, several sets of reference data, such as KITTI [50], TUM [126] etc. are available, however, most of them focused on a particular application of the real world (autonomous vehicle navigation, indoor navigation, etc.). Then, the movement of the camera is limited between two or three degrees of freedom. Therefore, to test our algorithm under more complex movements, a new benchmark dataset is required. To address this problem in this work we introduce a benchmark dataset which consists of indoor video sequences for each possible combination of movements given six degree of freedom (130 different video sequences with ground truth where all possible movement (65) were recorded at two different locations). For more details about our benchmark dataset please see [9].

In **Table** 5.2 quantitative results for the developed GPU implementation, tested on our dataset are shown. For practical purposes we present 33 possible movements (from a total of 65) so, in the first column we indicate the sequence within the proposed dataset, then, the next 6 columns (Movement), indicate which kind of movement is addressed in than sequence. Finally, the last three columns show quantitative results for different look up tables. When $\sigma_1 = 4, \sigma_2 = .5$ are applied, the full lookup table is reduced to 7343 elements (19.59% of the full lookup table without thresholding). This reduced lookup table includes all the basic movements in the dataset, achieving average accuracy of 96%. Furthermore, the maximum processing speed is 77 fps. In other experiment, $\sigma_1 = 8, \sigma_2 = 1$, an average accuracy of 95% is obtained, however, in this case, the lookup table can be reduced to

5394 elements (14.59% of the full lookup table without thresholding), this makes possible to increase the processing speed to 81 fps. Finally, in another test, when $\sigma_1 = 15, \sigma_2 = 2$ were applied, a lookup table with 3749 elements (10.59% of the full lookup table) delivers an average accuracy around 80% combined with processing speed of 97 fps. For all cases, it was demonstrated that a relatively high accuracy for complex movements is possible. In particular, given $\sigma_1 = 4, \sigma_2 = .5$ or $\sigma_1 = 8, \sigma_2 = 1$, accuracy around 95% and processing speed between 77-81 fps are achieved. In **Fig.** 5.10, we present qualitative results and finally, in **Fig.** 5.11 the drift error is shown.

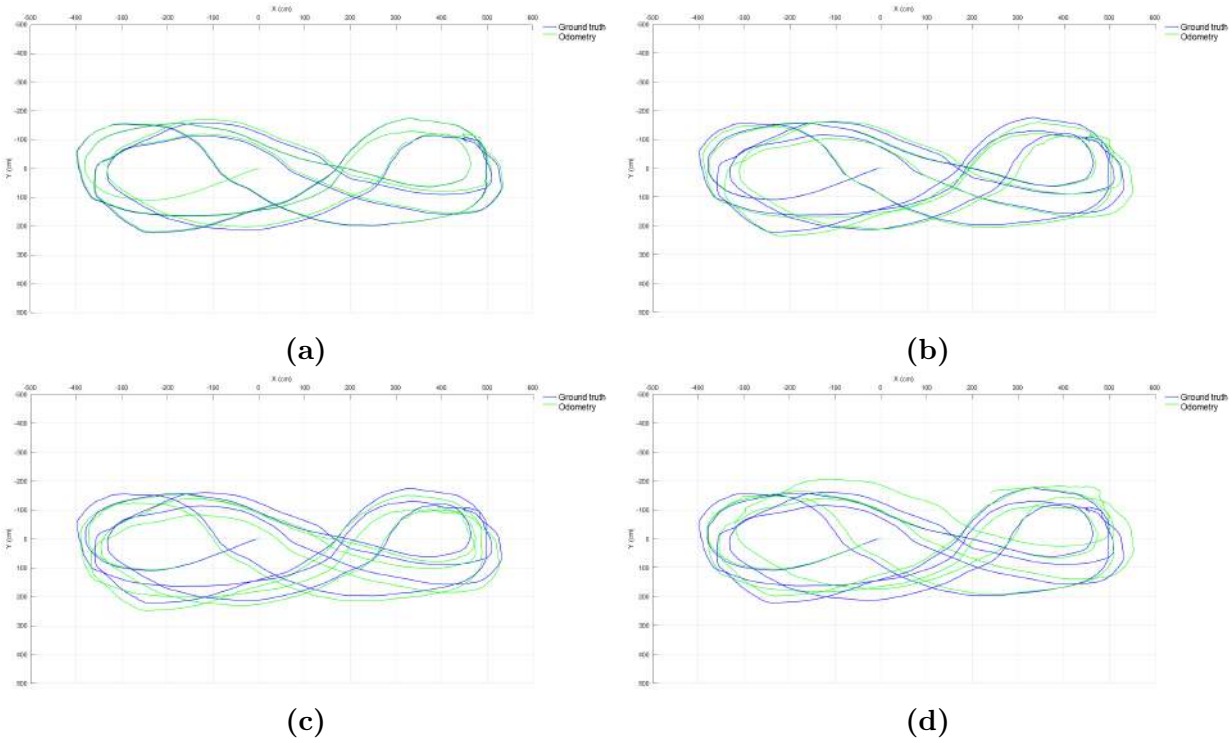


Figure 5.10. Performance for the pose estimation step under the proposed dataset. Sequence 48 which consists of a x, y, α camera movement and validates the performance under loop trajectories. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). (a) $\sigma_1 = 4, \sigma_2 = .5$: Large look-up tables (using 20% of the training data) makes possible accurate ego-motion estimation (close to the ground truth) but processing speed decrease. (d) $\sigma_1 = 20, \sigma_2 = 4$: Small look-up tables (using 5% of the training data) reach high processing speed but accuracy is low. In practice, look-up tables using 10-15% of the training data ($\sigma_1 = 8, \sigma_2 = 1$, $\sigma_1 = 15, \sigma_2 = 2$) deliver a good tradeoff between accuracy and processing speed, as shown in (b) and (c).

Table 5.2. Quantitative results for the pose estimation step under the proposed dataset. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). In all cases, high accuracy (close to 95%) is possible, using between 15 to 20% of the training data ($\sigma_1 = 4, \sigma_2 = .5$). Even with a small look-up table ($\sigma_1 = 15, \sigma_2 = 2$), which uses 10.59% of the training data, a relatively high accuracy (about 80%) can be obtained.

	Movement						**Accuracy**		
	x	**y**	**z**	α	β	γ	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
0	-	-	-	-	-	-	99.58	97.25	79.35
3	-	-	-	-	×	×	95.5	97.76	82.05
7	-	-	-	×	×	×	98.99	96.35	82.46
12	-	-	×	×	-	-	98.55	97.38	81.89
15	-	-	×	×	×	×	97.41	94.68	82.21
19	-	×	-	-	×	×	98.68	97.7	81.27
24	-	×	×	-	-	-	94.07	95.75	78.28
27	-	×	×	-	×	×	98.77	95.63	77.2
31	-	×	×	×	×	×	97.61	96.85	81.99
36	×	-	-	×	-	-	96.7	95.7	77.29
39	×	-	-	×	×	×	99.48	95.05	77.67
43	×	-	×	-	×	×	99.98	96.92	78.05
48	×	×	-	-	-	-	94.02	97.86	82.61
51	×	×	-	-	×	×	99.22	94.92	82.78
55	×	×	-	×	×	×	98.8	95.58	79.74
60	×	×	×	×	-	-	94.87	97.66	80.69
63	×	×	×	×	×	×	97.48	95.05	81.4

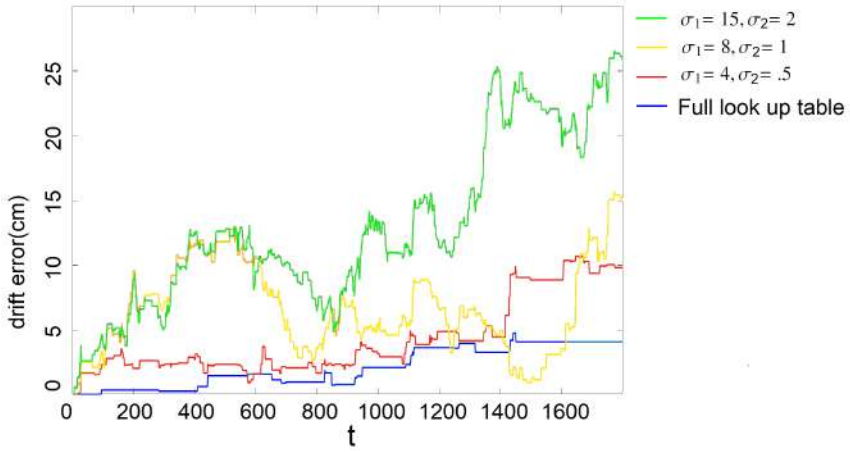


Figure 5.11. Drift error for different reductions of the look up table. The proposed dataset, sequence 48. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). For all cases, it was demonstrated that a relatively high accuracy for complex movements is possible.

5.8.4 The pose estimation step: the KITTI dataset

For the visual odometry estimation, the KITTI [50] provides 11 training sequences (00-10) with public truth while another 11 sequences (11-21), without public ground truth, are used for evaluation. In a first experiment, we carried out a cross validation for the training sequences, i.e., we built the lookup table using all sequences in the training set, except the sequence that is being evaluated, (see **Table** 5.3). Given the training sequences and given their corresponding ground truth, 21732 Q_j elements are available. For $\sigma_1 = 4, \sigma_2 = .5$, a lookup table of 13791 elements is used. This configuration delivers accuracy around 96% while camera egomotion is computed at 79 fps. For $\sigma_1 = 8, \sigma_2 = 1$, average accuracy of 95% is achieved, in this case, the lookup table was reduced to 7371 elements, increasing the processing speed up to 88 fps. Finally, for $\sigma_1 = 15, \sigma_2 = 2$, a lookup table with 1749 elements was used, average accuracy around to 80% combined with possessing speed of 107 fps was achieved. For all cases, a relatively high accuracy was obtained. In particular, for $\sigma_1 = 4, \sigma_2 = .5$, $\sigma_1 = 8, \sigma_2 = 1$, accuracy around 95% and processing speed between 80-90 fps are possible. In **Fig.** 5.12, we present qualitative results for the data presented in **Table** 5.3.

In a second experiment, performance comparisons with previous work were made. For previous work, we obtained the performance data from the corresponding references. For the proposed algorithm, we have submitted to the KITTI benchmark suite our results for the test sequences (11-21), reaching an average error of 4% (please see [8]). For the leader table we are ranked in the 82th place, however, most algorithms in the leader table are stereo-based approaches, RGB D-based approaches (single camera combined with a depth sensor) or SLAM algorithms in which re-localization and loop closure improves the accuracy. For the monocular VO algorithms, the most accurate approach is the FVO algorithm [134], ranked in the 43th place of the leader table. In **Table** 5.5 quantitative comparisons are shown. For traditional approaches [30, 49], our algorithm outperforms previous work in both, accuracy and processing speed, our algorithm is $\times 2$ more accurate and $\times 15$ faster than those works. This is because most of the work uses binary-based feature description and matching techniques and, these are sensible to image degradations. In our case, the proposed pixel tracking/feature matching steps (**Sections** 4.2 and 4.3) provide high robustness for image degradations (it is possible to recover $\times 10$ more features than in previous works) and this improves the performance.

For recent works [34, 91, 104, 139], our algorithm reach a good tradeoff between accuracy and processing speed, similar accuracy and higher processing speed than those works. Compared to CNN-based approaches [34, 139] our algorithm outperforms those works in terms of accuracy, 4% more accurate than [34] and 7% more accurate than [139]. For processing speed, we achieve a speed up of 2 times (67 fps) compared with [34]. For [139] outperforms our algorithm in terms of processing speed (+71 fps), however, the GPU used in that work is highly powerful (2888 CUDA cores) compared than used in this work (1664 CUDA cores). For [91], our algorithm reaches similar accuracy and processing speed than that work, nevertheless, lower hardware resources are required, 1664 CUDA cores compared with the 1280 CUDA cores used in this work. Finally, [104] outperforms our algorithm in terms of accuracy (4% more accurate than that work), but with lower processing speed because the reported value (333.3 fps), does not consider the image readout, feature extraction and feature tracking steps, and these are the heaviest operations in the VO formulation. In **Fig.** 5.13, we present qualitative results for test sequences 11-14 of the KITTI dataset.

Table 5.3. Quantitative results for the KITTI dataset. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). Large look-up tables ($\sigma_1 = 4, \sigma_2 = .5$) deliver error lower than 4% while small look-up tables ($\sigma_1 = 15, \sigma_2 = 2$) reach high processing speed. Look-up tables using 10-15% of the training data ($\sigma_1 = 8, \sigma_2 = 1$), deliver a good tradeoff between accuracy and processing speed (mean eror of 7%).

	Accuracy		
	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
00	95.77	93.16	82.65
01	95.93	93.60	82.07
02	95.12	93.26	82.99
03	95.56	93.65	81.15
04	95.46	93.68	81.88
05	95.01	93.74	81.21
06	95.33	93.45	82.92
07	95.16	93.08	81.00
08	95.79	93.22	82.54
09	95.31	93.91	82.63
10	95.52	93.15	82.73

For hardware resources usage, in **Table** 5.4 the average resources consumption is shown. For all the experiments presented in this work, low hardware requirements (around 26 % of a GPU GTX970M) is needed. The GPU GTX970M is a portable version of the popular GPU GTX970 and given the proposed algorithm it requires 26 % of the maximum processing clock. We believe that an efficient implementation into embedded GPU processors such as the NVIDIA TX2 (around 256 CUDA cores) is feasible and will be part of our future work.

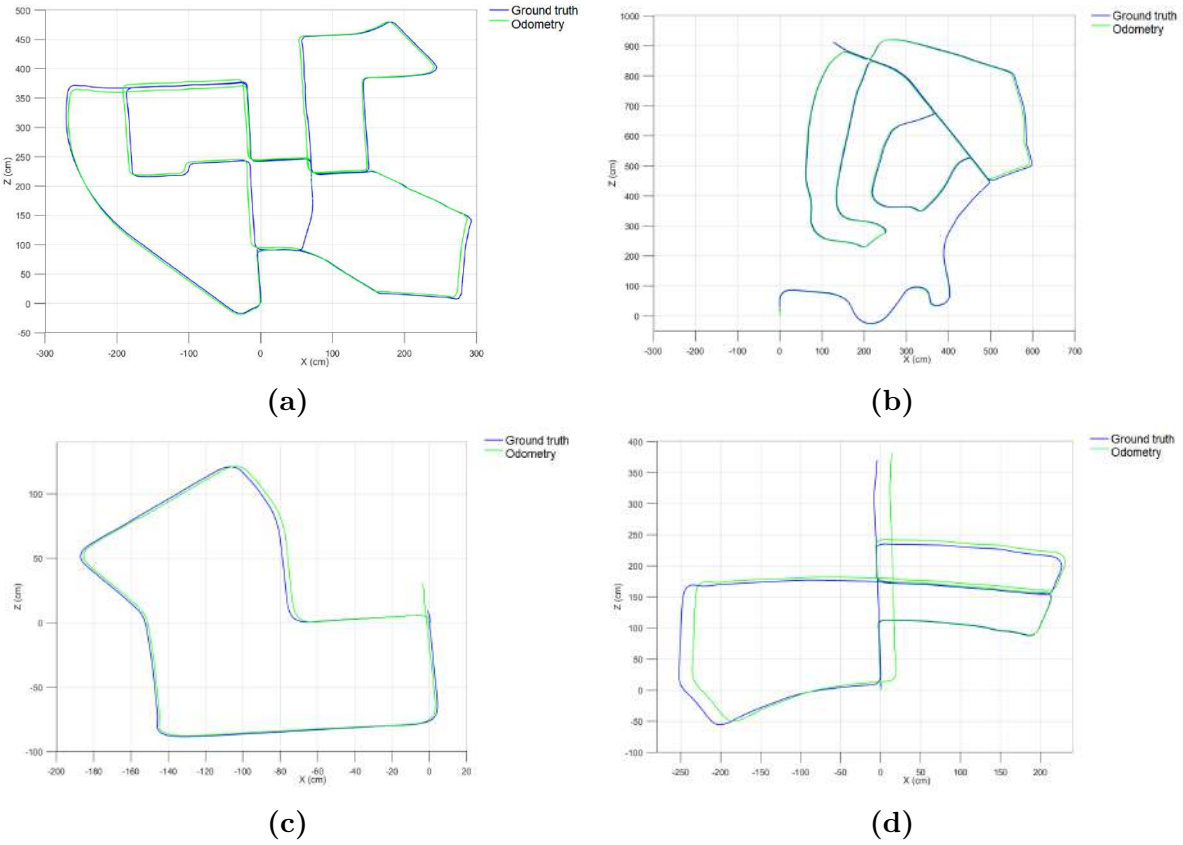
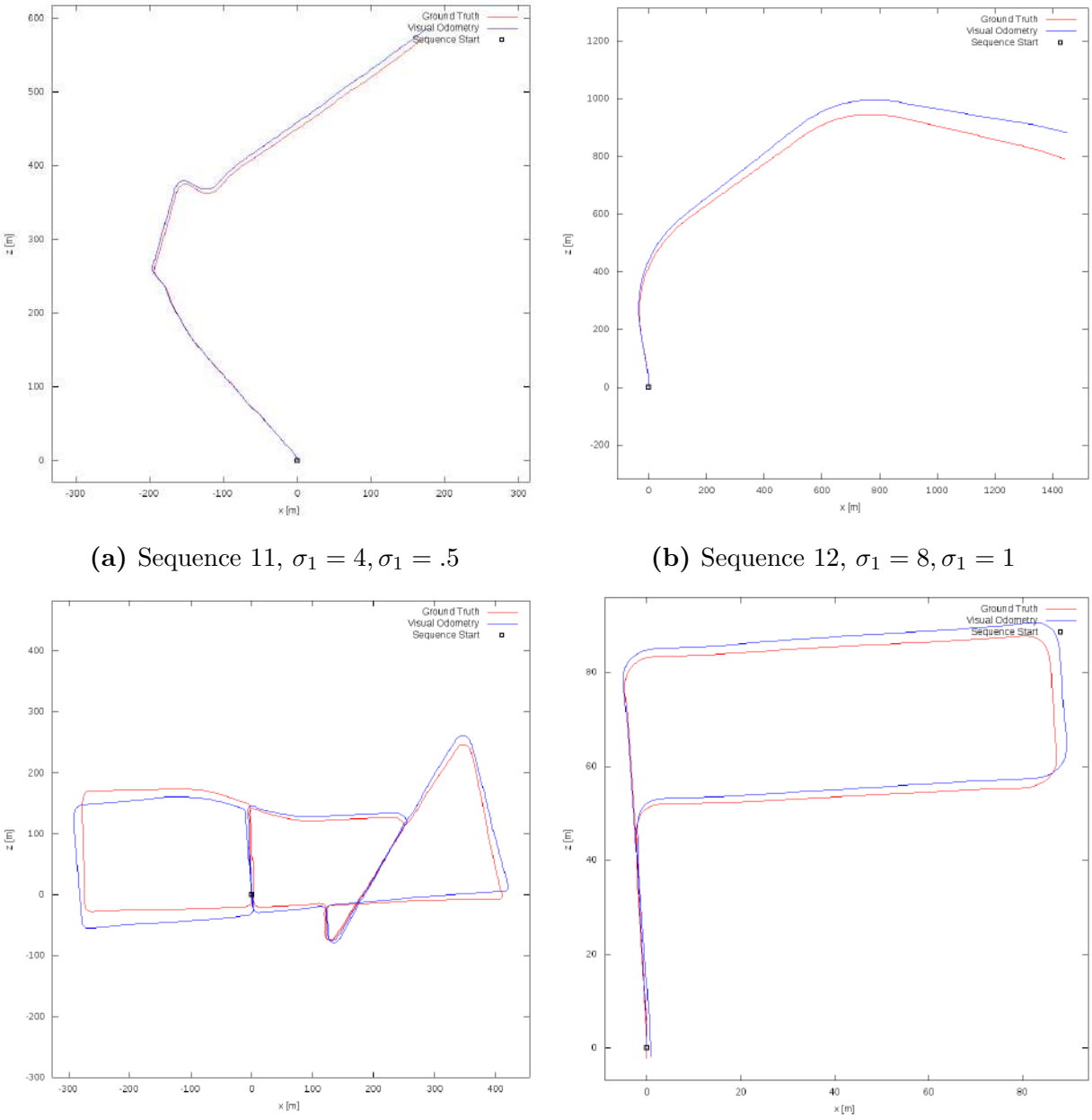


Figure 5.12. Performance for the KITTI dataset (training sequences). σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). Setting $\sigma_1 = 4, \sigma_2 = .5$: For video sequences in which most of the time the environment is rigid, high accuracy (near to 96%) can be reached as in (a) (b) and (c). For video sequences with dynamic objects (d) the accuracy level decreases (accuracy of 87% can be reached).

Table 5.4. Hardware resource consumption for the developed implementation. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). For the GPU GTX970M, the proposed algorithm requires near to 26% of the hardware resources. i.e., our algorithm requires around the 26% of the maximum processing clock.

Resource	Consumption/σ_1, σ_2		
	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
GPU Core	26%	27%	29%
GPU Memmory	15%	17%	21%
GPU Video engine	0%	0%	0%
GPU Memmory controller	7%	7%	7%



(a) Sequence 11, $\sigma_1 = 4, \sigma_1 = .5$

(b) Sequence 12, $\sigma_1 = 8, \sigma_1 = 1$

(c) Sequence 13, $\sigma_1 = 15, \sigma_1 = 2$

(d) Sequence 14, $\sigma_1 = 20, \sigma_1 = 4$

Figure 5.13. Performance for the KITTI dataset (test sequences). σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). Setting $\sigma_1 = 4, \sigma_2 = .5$: For video sequences in which most of the time the environment is rigid, high accuracy (near to 96%) can be reached (a) For video sequences with dynamic objects (b) the accuracy level decreases (accuracy of 91% can be reached). In most of the cases (c) and (d), high level of accuracy can be reached (near 94%).

Table 5.5. Quantitative results for the proposed algorithm compared with previous works. In most cases, our algorithm outperforms previous works in terms of accuracy and processing speed.

Algorithm	Accuracy	Speed	Hardware
Geiger et al (2011) [49]	83.71%	16.39 fps	CPU (i7-4720HQ)
Ciarfuglia et al (2014) [30]	85.56%	9.09 fps	CPU (i7-4720HQ)
Costante et al (2016) [34]	91.04%	3.27 fps	CPU (i7-4720HQ)
Costante et al (2016) [34]	91.04%	20.83 fps	GPU (Tesla K40)
Mohanty et al (2016) [91]	94.50%	111.11 fps	Intel Xeon @4 + GPU (GTX 970)
Holzmann et al (2016) [61]	91.94%	20.30 fps	CPU (i7-4820K)
Weber et al (2017) [139]	88.53%	158.73 fps	GPU (GTX 970)
Pillai et al (2017) [104]	99.72%	333.3 fps*	CPU (i7-3920XM)
This work	95.07%	87.34 fps	GPU (GTX 970M)

*This value is only for the ego-motion computation step, other steps such as feature tracking via the KLT algorithm are not considered.

5.8.5 The depth from motion step: performance and limitations

In **Fig.** 5.14, quantitative and qualitative results (RMS error and depth maps, respectively) for the KITTI dataset [50], are shown. In order to compute the error, we use the ground truth depth map as reference and then we computed the RMS error. In all cases our algorithm provides rough depth maps compared with stereo-based or deep learning approaches but with real-time processing and with the capability to be implemented in hardware, suitable for smart cameras or embedded robotic applications.

5.8.6 The linear triangulation step: performance and limitations

Finally, in **Fig.** 5.15 an example of 3D reconstruction using our approach is shown. Previous works such as the ORB-SLAM [95] or LSD-SLAM [41] compute motion and depth in 2 to 7% of all image pixels, while ours compute 80% of the image pixels. Then, our algorithm improves about 15 times the current state of the art, making 3D reconstructions possible in real time and with the capability to be implemented within FPGA devices, suitable for smart cameras.

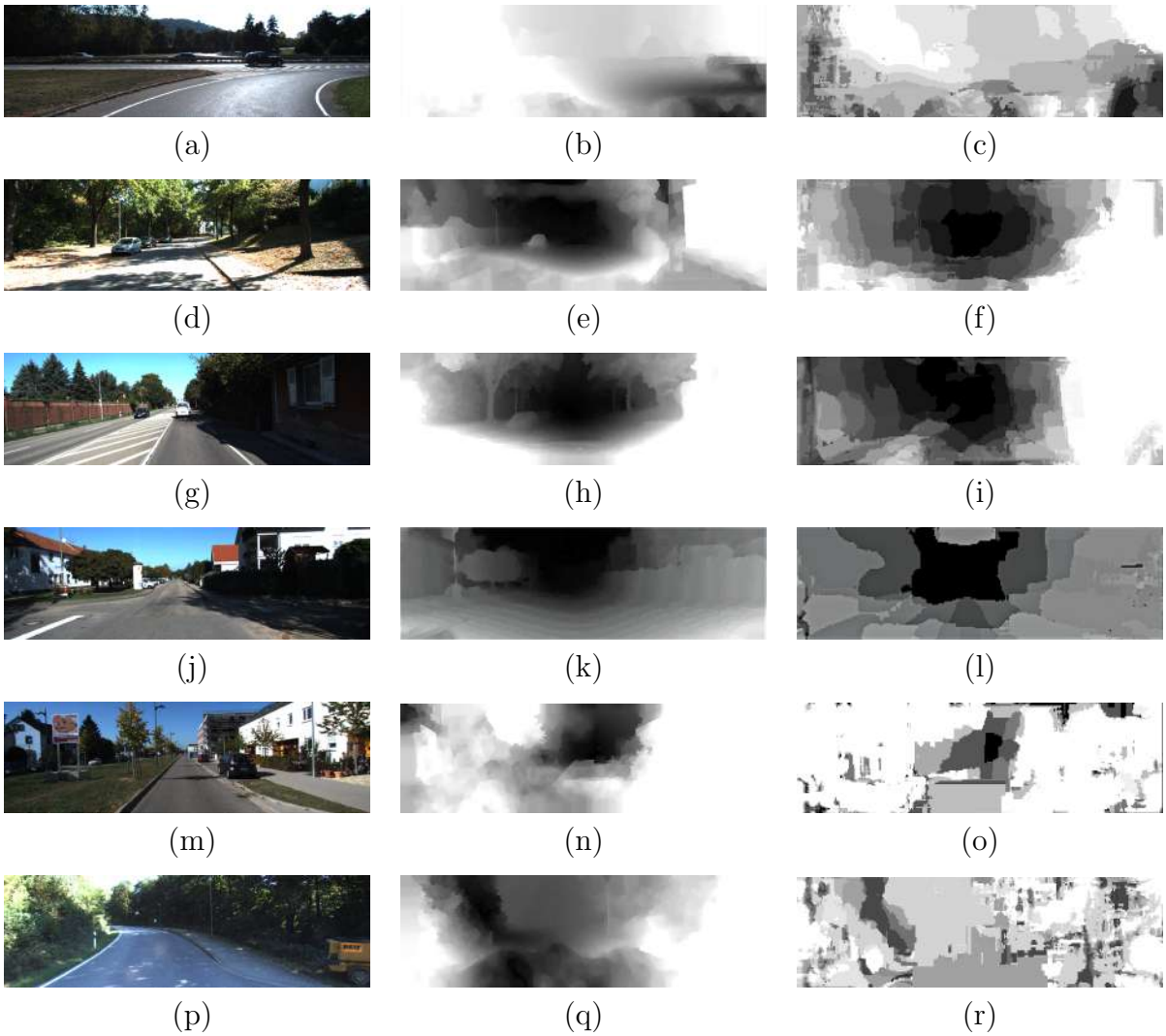
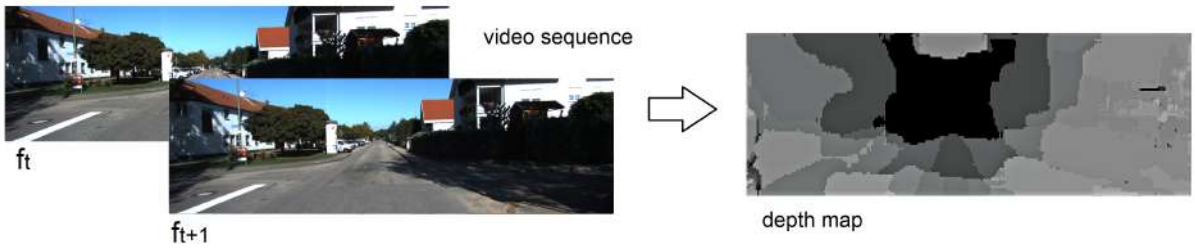
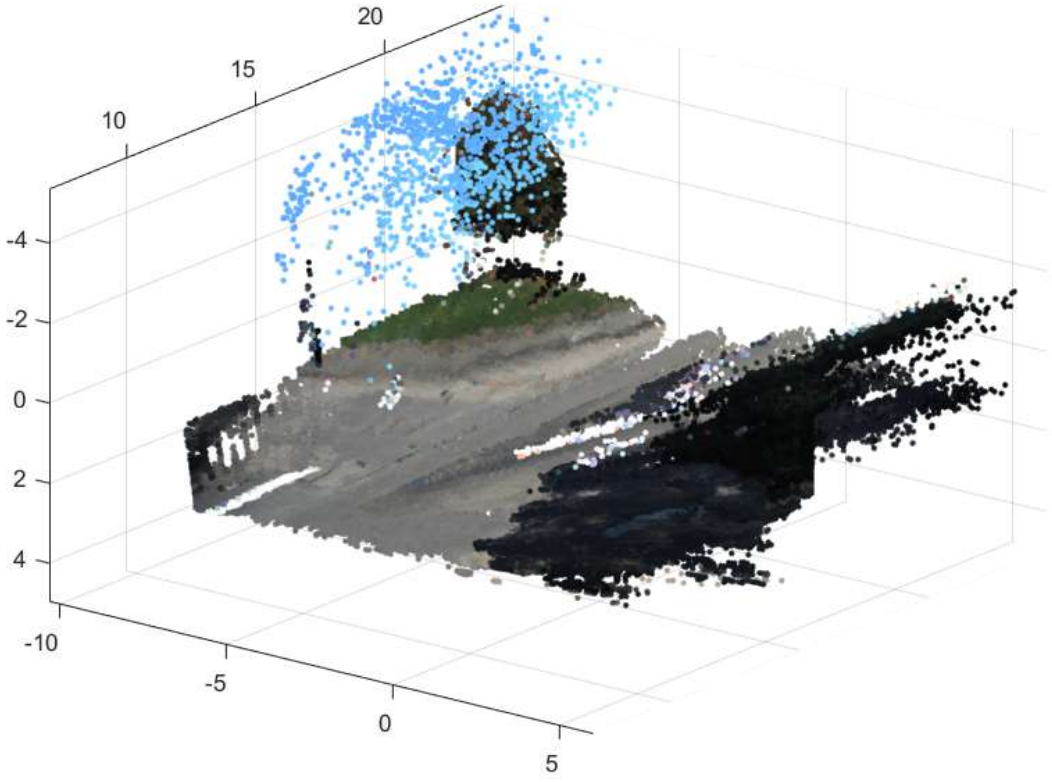


Figure 5.14. Depth from motion: results for the KITTI dataset. (a) Sequence 02, reference image. (b) Ground truth. (c) Depth estimation (error = 22%). (d) Sequence 03, reference image. (e) Ground truth. (f) Depth estimation (error = 21%). (g) Sequence 04, reference image. (h) Ground truth. (i) Depth estimation (error = 22%). (j) Sequence 05, reference image. (k) Ground truth. (l) Depth estimation (error = 21%). (m) Sequence 06, reference image. (n) Ground truth. (o) Depth estimation (error = 21%). (p) Sequence 09, reference image. (q) Ground truth. (r) Depth estimation (error = 22%). In all cases our algorithm provides rough depth maps with the capability to be implemented in hardware, suitable for smart cameras or embedded robotic applications.



(a)



(b)

Figure 5.15. Performance of the mapping step. The KITTI dataset: Sequence 06. (a) Our depth from motion algorithm (**Section** 4.8) provides rough depth maps (lower accuracy compared with previous algorithms) but with real-time processing and with the capability to be implemented in embedded hardware. (b) 3D reconstruction by the proposed approach. Compared with previous works, our algorithm improves them about 15 times (in terms of mapping density), as a result, real-time dense 3D reconstructions can be obtained and, these can be exploited by several real world applications such as, augmented reality, robot vision and surveillance, autonomous flying, etc.

5.9 Summary

In this chapter, we have presented the results of the GPU implementation of our monocular-SLAM formulation. Our GPU-based feature tracking/matching algorithm delivers dense tracking (more feature points than previous algorithms) and without outliers. This avoid the use of RANSAC outliers filtering and allows full parallelization in dedicated GPU hardware. We have developed a GPU-based monocular-SLAM system which deliver high efficiency in terms of algorithmic parallelization. Therefore, unlike previous work, the camera ego-motion (localization) can be estimated without iterative behavior and without geometric constraints, suitable for embedded applications. The experimental results shown that our algorithm reaches high accuracy (95.07%), in comparison with previous monocular VO algorithms such as CNN and depth learning-based algorithms (which reach 90% of accuracy) and with a processing speed up to 17 times faster than previous works.

We have presented a new set of reference data for visual odometry (VO). The dataset can be used for researchers to test and evaluate their VO algorithms in complex movements. This dataset provides 144 video sequences with public ground truth and they were recorded considering all possible motions given six degree of freedom. For download: images, ground truth, documentations and scripts, please see [9].

We have developed a new depth from motion/linear triangulation algorithm whose GPU implementation deliver high efficiency in terms of algorithmic parallelization. So, unlike previous works, the depth information is estimated in real time inside a compact GPU device. Faster and with lower hardware resources than previous works. It makes it possible to reach dense 3D reconstruction, improving by around 15 times the current state of the art.

Chapter 6

LT-SLAM: FPGA implementation

In **Fig.** 6.1, a general description of the developed FPGA architecture is shown. The architecture focuses on an FPGA implementation where all recursive/parallelizable operations are accelerated in the FPGA fabric. First, the "feature extraction" unit reads the pixel stream (pix [7:0]) delivered by the imager and extracts the visual features (corners) by applying a parallelized version of the Harris algorithm [56]. The "frame buffer" unit reads the pixel stream (pix [7:0]) delivered by the imager. In this block, frames captured by the imager are fed to/from an external DRAM memory and delivers pixel streams for two consecutive frames in parallel (pix1 [7:0], pix2 [7:0]). Then, the pixel stream for two consecutive frames (pix1 [7:0], pix2 [7:0]) are used to computed pixel tracking for all pixels in the reference image (Δ_x [7:0], Δ_y [7:0]). In the next step, feature matching ($\Delta_x y$ [7:0]) and depth from motion are computed in parallel, for that, (Δ_x [7:0], Δ_y [7:0]) are used to compute the depth in the scene (d [7:0]). In the last step, feature matching ($\Delta_x y$ [7:0]) is used to compute the relative camera pose for the current frame. "Circular buffers" implemented inside the local processors (feature extraction, pixel tracking, feature matching, etc.) are used to hold local sections of the frames that are being processed and allow for local parallel access that facilitates parallel processing. In the following subsections, details about the algorithm parallelization are presented.

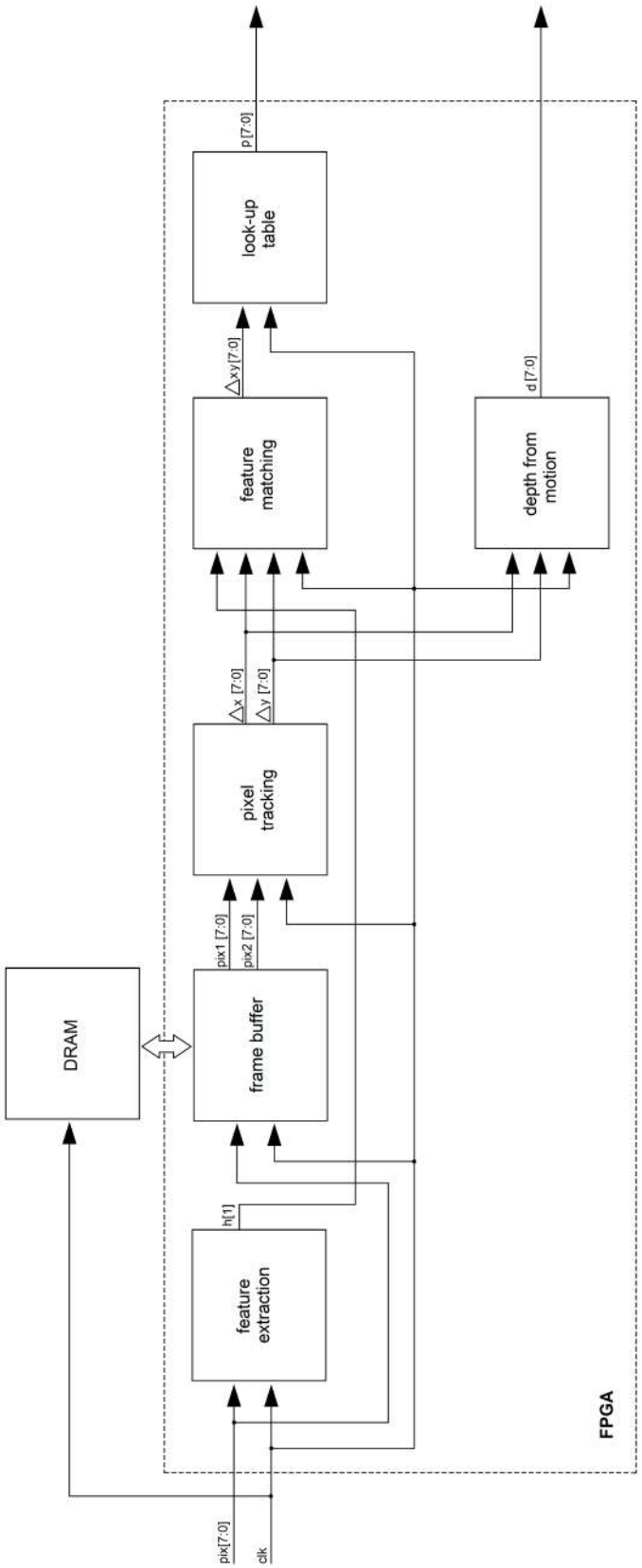


Figure 6.1. Block diagram of the FPGA implementation. All exhaustive operations are parallelized and computed inside an FPGA hardware architecture, while non parallelizable operations, such as map construction and pose estimation are computed in a sequential processor (CPU).

6.1 Feature extraction

A general description of the developed FPGA architecture is illustrated in **Fig.** 6.2. The structure of the architecture consists of four steps: image gradients, derivatives, corner response and corner detection. First, data/parts of the frames are stored in circular buffers that can hold rows temporarily as cache, store image rows from the input images, and that can deliver parallel data to the image preprocessing module. In the next step the FPGA architecture computes the vertical and horizontal gradients. Then, given the image gradients the image derivatives are computed. For the next step, circular buffers delivers image pixels for the smoothing operations and, reconfigurable convolution units (see [3]) compute the smoothing operation. In the next step, the FPGA architecture computes the corner metric response. Finally, corner detection delivers ones at corner points retained after a non-maxima suppression step and zero otherwise. For the detailed mathematical formulation see **Section** 4.1.

6.2 Circular buffer

In [3] we proposed a circular buffer scheme in which input data from the previous n rows of an image can be stored using memory buffers (block RAMs/BRAMs) up to the moment when a $n \times n$ neighborhood is scanned $n \times n$ in the subsequent rows. In this work, we follow a similar approach to achieve high data reuse and high level of parallelism. Then, our algorithm is processed in modules where all the image patches can be read in parallel. First, a shift mechanism "control" unit manages the read/write addresses of $n+1$ BRAMs. In this formulation, n BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, the data, inside the read mode BRAMs can be accessed in parallel and each pixel within a $n \times n$ region is delivered in parallel a $n \times n$ buffer, as shown in **Fig.** 6.3, where the "control" unit delivers control data (address and read/write enable) for the BRAM modules. Then, one entire row is stored in each BRAM, finally, the "data" unit delivers $n \times n$ pixels in parallel. In our implementation, there is 1 circular buffer of 13×13 pixels/bytes, 1 circular buffer of 17×17 and 2 circular buffers of 3×3. For more details on the formulation and behavior of the circular buffer see [3].

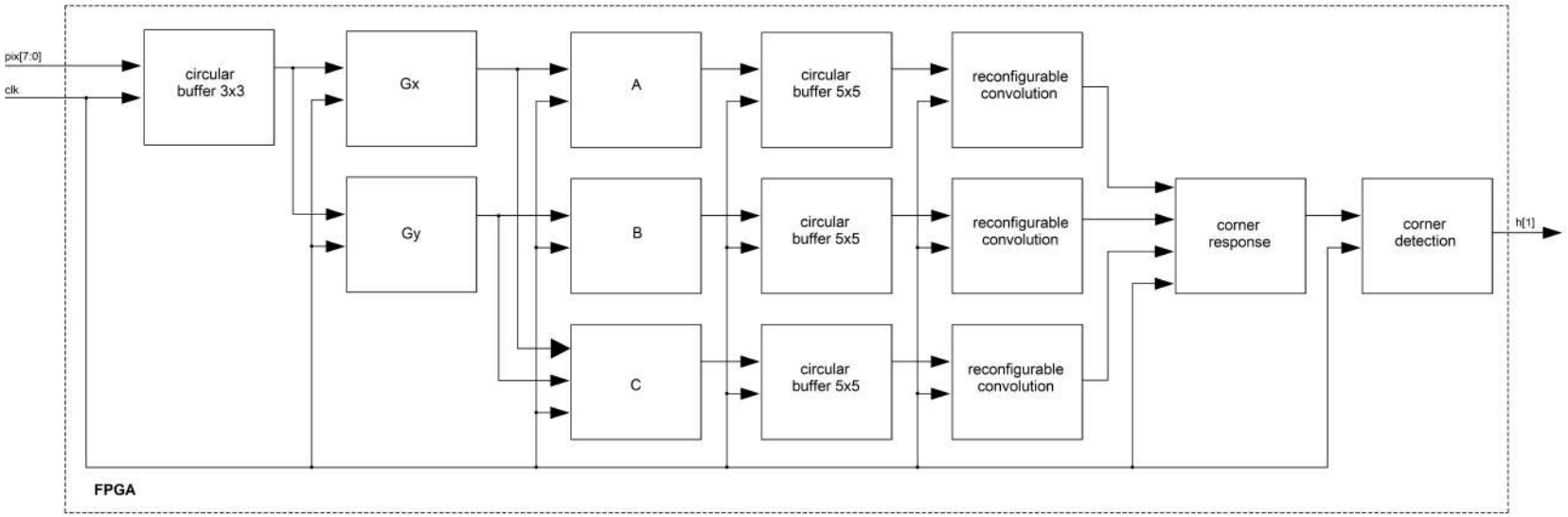
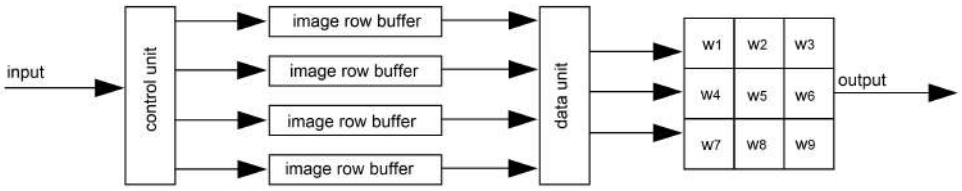
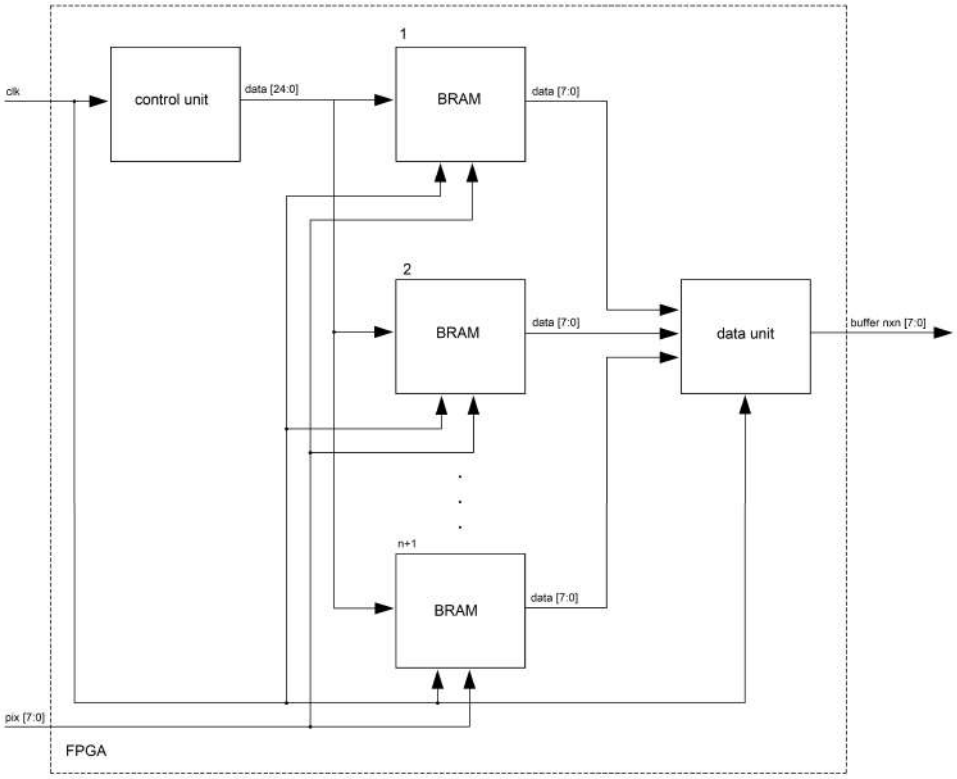


Figure 6.2. FPGA implementation for the feature extraction step. First, the vertical/horizontal gradients and the image gradient derivatives are computed in parallel. Then, the corner metric response is computed. Finally, using the corner metric response, the corner detection process is carried out. Circular buffers attached to the local processors hold temporarily as cache and deliver parallel data to the processors.



(a)



(b)

Figure 6.3. The circular buffers architecture. (a) General formulation of a 3×3 circular buffer. (b) FPGA architecture for the circular buffers. For a $n \times n$ patch, a shift mechanism "control" unit manages the read/write addresses of $n+1$ BRAMs. In this formulation n BRAMs are in read mode and one BRAM is in write mode in each clock cycle. Then, the $n \times n$ buffer delivers logic registers with all pixels within the patch in parallel.

6.3 Pixel tracking

For the "pixel tracking" unit, we consider that the tracking/flow estimation problem can be a generalization of the dense stereo matching problem. That is, stereo matching algorithms track (searching on the horizontal axis around the search image window), all the pixels in the reference image window. Pixel tracking aims to track all the pixels between two consecutive frames from a video sequence (searching around spatial coordinates of the pixels in the search image). Then, it is possible to extend previous stereo matching FPGA architectures to fulfil with our application domain. In this work, we extended the FPGA architecture presented in [101], since it has low hardware requirements and high parallelism level. In **Fig.** 6.4, the developed architecture is shown. First, the "curl" units compute the curl (**Section** 4.2) and deliver curl pixel stream of the reference/search images in parallel. More details about the FPGA architecture of this unit are shown in **Section** 6.3.1. The "circular buffer" units are responsible for data transfers in segments of the image. So, the core of the FPGA architecture are the circular buffers attached to the local processors that can hold temporarily as cache, image sections from two frames, and that can deliver parallel data to the processors. Then, given optical flow previously computed, 121 search regions (defined by the search size defined by the user) are constructed in parallel (see **Fig.** 4.5a). For our implementation, the search region size is equal to 10, therefore, the center of the search regions are all the sampled pixels within the reference region. Given the reference region in $f_t(x, y)$ and 121 search regions in $f_{t+1}(x, y)$, the search regions are compared to the reference region in parallel. For that, a pixel-parallel/window-parallel scheme is implemented. Finally, in the "flow estimation" unit a multiplexer tree can determine the indices a, b that minimize the corresponding correlation function and, therefore, the tracking of pixels for all the pixels in the reference image. For the detailed mathematical formulation see **Section** 4.2.

6.3.1 Curl estimation

In **Fig.** 6.5, the curl architecture is shown. First, one "circular buffer" holds 3 rows of the frame being processed and allows for local parallel access of a 3×3 patch that facilitates parallel processing. Then, image gradients ($\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial f(x,y)}{\partial y}$) are computed. Another "circular buffer" holds 3 rows of the gradient image previously computed and delivers a 3×3 patch for the next step. Second derivatives ($\frac{\partial}{\partial y}\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial}{\partial x}\frac{\partial f(x,y)}{\partial y}$) are computed inside the "derivative" unit. Finally, the curl of the input image is computed by the "curl" unit.

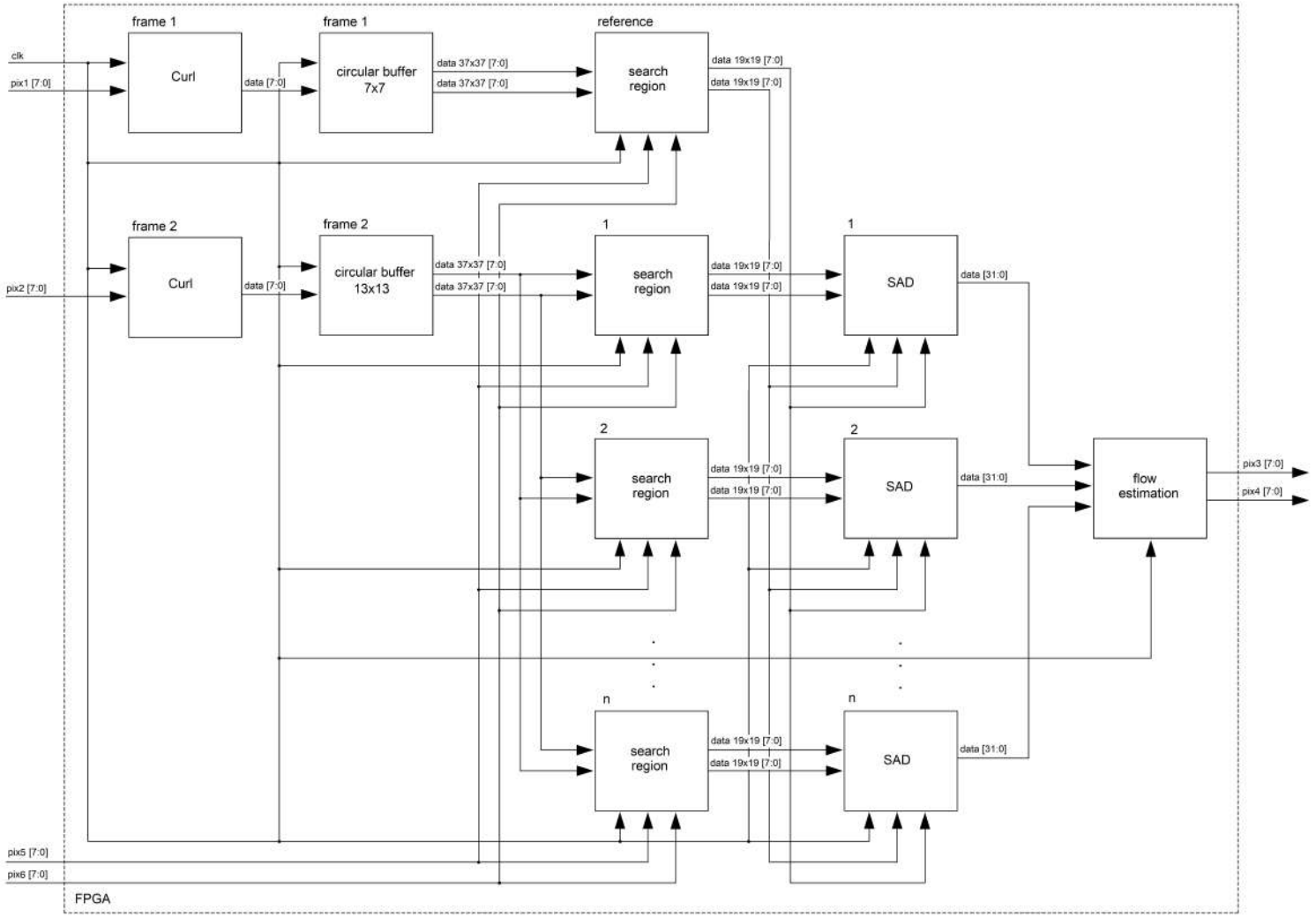


Figure 6.4. FPGA implementation for the pixel tracking step. In order to achieve high performance for hardware architectures, an FPGA-based pixel-parallel/window-parallel approach is used.

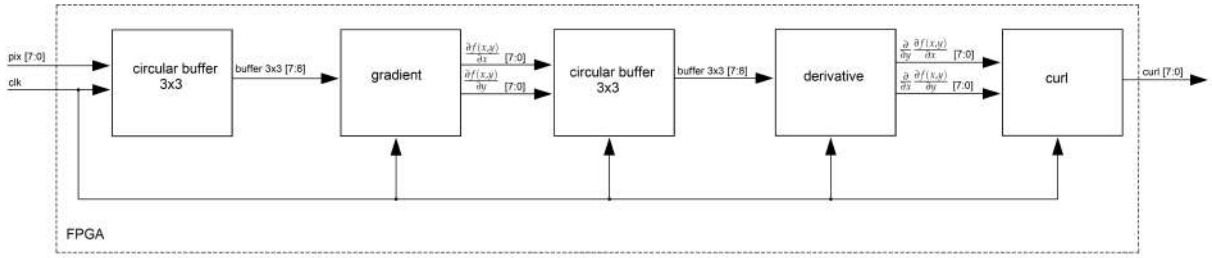


Figure 6.5. FPGA architecture for the "Curl" unit. Each block parallelize a different part of the original "curl" formulation.

6.4 Feature matching

In **Fig.** 6.6, the feature matching architecture is shown. Let pix1 [7;0], pix2 [7:0], pix3 [7:0] be the pixel tracking at current frame Δ_x, Δ_y and the feature extraction for the search frame $(f_{t+1}(x, y))$, respectively; first, the "circular buffer" unit holds 3 rows of the feature extraction at $(f_{t+1}(x, y))$ and allows for local parallel access of a 3×3 patch that facilitates parallel processing. Then, the "feature matching" unit carries out the feature matching process by using comparators and a shift window approach; delivering pix4 [7;0], pix4 [7:0] which corresponds with $\Delta_x(x, y)$, $\Delta_y(x, y)$ respectively. For the detailed mathematical formulation see **Section** 4.3.

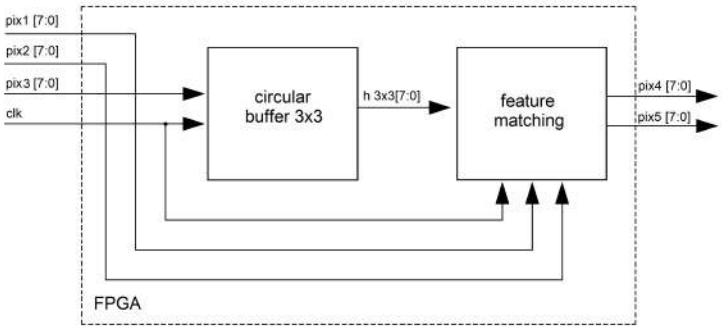


Figure 6.6. FPGA implementation for the feature matching step. First, a circular buffer delivers a 3×3 window centered in the pixel being processed. Then, the "Feature matching" block carries out the matching operation using comparators and a shift window approach.

6.5 Look-up table

In **Fig.** 6.7, the FPGA architecture for Camera ego-motion is shown. Let pix1 [7;0], pix2 [7:0] be the feature matching at current frame Δ_x, Δ_y, respectively; first, the "discrete histogram" unit computes the discrete histogram for the eight motion parameters previously proposed (**Section** 4.5). Then, the "MAX" unit computes the motion parameters as the maximum value within the eight regions of the discrete histogram, as follows: $s_R = \text{MAX}(H(\Delta_x(x,y)_R))$. This equation simplifies the previously proposed formulation (**Eq.** 4.18) since the "MAX" function is implemented as a multiplexer tree, decreasing the hardware resources during FPGA implementation, facilitating parallel/pipeline design and with low compromise compared with the original formulation. Finally, a "CASE" structure searches within the lookup table in order to get the current camera ego motion. For the detailed mathematical formulation see **Sections** 4.5 and 4.6.

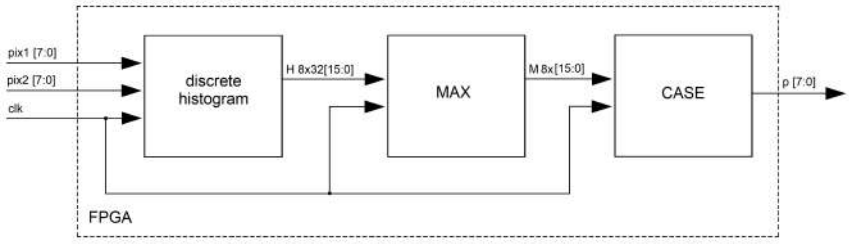


Figure 6.7. FPGA implementation for the camera ego-motion step. A "MAX" function implemented as a multiplexer tree computes the maximum value within the discrete histogram. Then, a "CASE" structure searches within the lookup table and gets the camera ego motion.

6.6 Depth from Motion

In **Fig.** 6.8, the depth estimation architecture is shown. Let pix1 [7;0], pix2 [7:0] be the pixel stream for the pixel tracking at current frame (Δ'_x, Δ'_y); first, the "multiplier" unit computes the square value of the input data. Then, the "adder" unit carries out the addition process for both components $(\Delta_x^{2'}, \Delta_y^{2'})$. Finally, the "sqrt" unit computes the depth in the scene by computing the square root for $\Delta_x^{2'} + \Delta_y^{2'}$. In order to achieve high efficiency in the square root computation, we adapted the architecture developed by Yamin Li and Wanming Chu [77]. This architecture uses a shift register mechanism and compares the more significant/less significant bits to achieving the root square operation without using embedded multipliers.

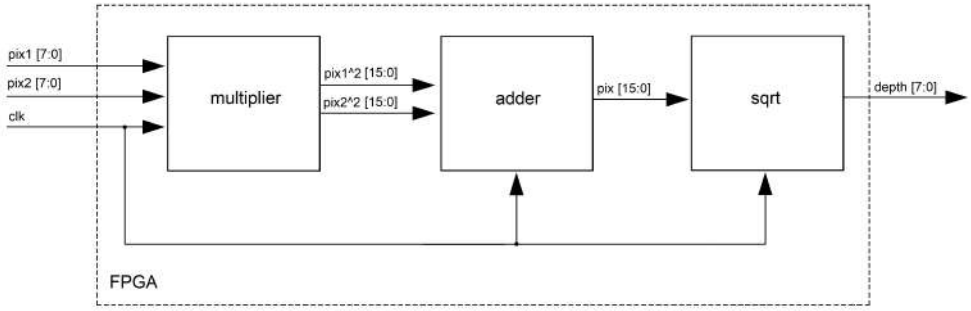


Figure 6.8. FPGA implementation for the depth from motion step.

6.7 Pose estimation & map construction

Our FPGA architecture delivers two outputs the current camera ego-motion and the current depth map, p[7:0] and d[7:0], respectively. In order to estimate the camera pose (localization) and the scene reconstruction (mapping), we use GPU archuitectures previously presented, **Sections** 5.5 and 5.5; where p[7:0] is mapped with $pose_t$, **Eq.** 4.24 and 4.26 while d[7:0] is mapped with **depth**(x, y), **Eq.** 4.29 and 4.34. The reason of a GPU implementation instead of an FPGA architecture is due to the mathematical complexity of the proposed formulation (matrix multiplications, inverse matrix, decimal operations, etc.) so, a GPU implantation is more suitable for this purpose. All operations could be implemented in CPU; maintaining high processing speed and requiring only an FPGA device to solve the monocuar-SLAM problem.

6.8 Performance and limitations

In this section, the results of the implementation of the proposed algorithms are shown. First,we present the results of the implementation of our pixel tracking algorithm. The results for the feature matching step are shown. Finally, we present the results of the implementation of the proposed camera ego-motion estimation algorithm and for the depth from motion algorithm. For all experiments the developed FPGA architecture was implemented in an FPGA Cyclone IV EP4CGX150CF23C8 of Altera. All modules were designed via Quartus II Web Edition version 10.1SP1 and GPStudio [27]. All modules were validated via post-synthesis simulations performed in ModelSim Altera. In **Fig.** 6.9 an overview of the developed FPGA architecture implemented inside the DreamCam [20] via GPStudio is shown.

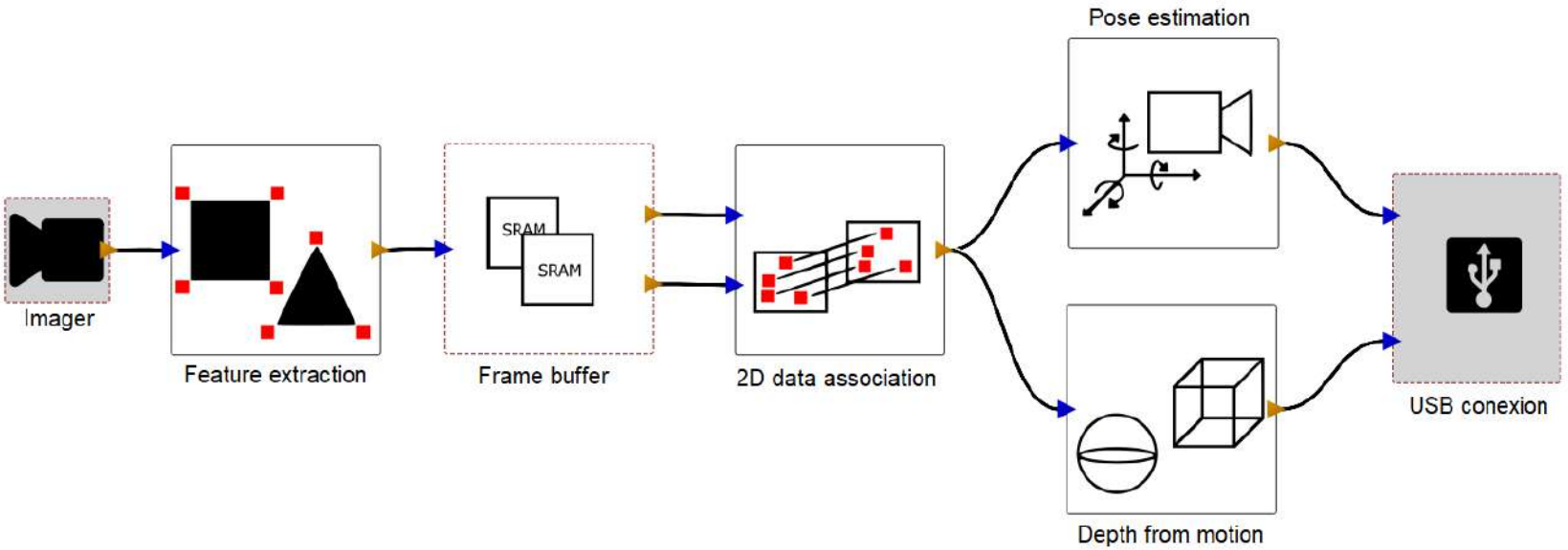


Figure 6.9. DreamCam/GPStudio implementation for the developed FPGA architecture. An MT9M031 imager: 1.2-mega pixel (1280 960) CMOS image sensor manufactured by Aptina, provides full 1280 960-pixel resolution at 45fps. The "featureExtractor" unit extracts the corners in the current image. Two 16 M-bit static RAMs are used inside the "frameBuffer" unit. Then, Feature tracking/feature matching are carried out inside the "featureTracking" unit. Finally, the "cameraPose" and "depthMotion" units estimate the current camera ego-motion and depth map respectively.

The full hardware resource consumption of the architecture is shown in **Table** 6.1. Our algorithm formulation allows for a compact system design, it requires 66% of the total logic elements of the FPGA Cyclone IV EP4CGX150CF23C8. For memory bits, our architecture uses 74% of the total resources, this represents 26 block RAMs consumed mainly in the circular buffers. This hardware utilization enables to target a relatively small FPGA device and therefore could be possible a small FPGA-based smart camera, suitable for real-time embedded applications. In the following subsections comparisons with previous work are presented. For pixel tracking, comparisons with previous FPGA-based pixel tracking/optical flow algorithms are presented. For depth estimation, previous FPGA-based approaches are limited; there are several CPU-based approaches but in these cases most of the effort was for accuracy improvements and real-time processing or embedded capabilities were not considered, therefore, proper comparisons are not possible so, only qualitative results are presented. Finally, for the camera ego-motion, simulation results using the KITTI dataset [50] and validations under real world scenarios are presented.

Table 6.1. Hardware requirements for the developed FPGA architecture.

Resource	Consumption/image resolution		
	640×480	**320×240**	**256×256**
Total logic elements	69,879 (59%)	37,059 (31%)	21,659 (18%)
Total pins	16 (3%)	16 (3%)	16 (3%)
Total Memory Bits	618,392 (15%)	163,122 (4%)	85,607 (2%)
Embedded multiplier elements	0 (0%)	0 (0%)	0 (0%)
Total PLLs	1 (25%)	1 (25%)	1 (25%)

6.8.1 The pixel tracking step: performance and limitations

Compared with the previous work, in **Table** 6.2 we present the utilization of hardware resources between our FPGA architecture and previous FPGA based optical flow algorithms. There are several works [13, 37, 87, 140] whose implementations of FPGA aim to parallelize all recursive operations in the original mathematical formulation. Unfortunately, most popular formulations such as those based on KTL [83] or Horn-Schunck [63], have iterative operations that are hard to parallelize. As result, most previous works have relatively high hardware occupancy/implementations compared with a full parallelizable design approach. Compared with previous works, our FPGA architecture outperform

most previous works, for similar image resolution, less logic elements and memory bits than [37, 62]. And less logic elements and memory bits than [13]. [13] decreases the memory usage by a multiscale coding which makes possible to store only half of the original image, however, this reduction involves pixel interpolation for some cases and this increases the logic elements usage. For, [87], the authors introduced an iterative-parallel approach; this makes possible to achieve low hardware requirements but processing speed is low. Finally, for [140] a filtering-based approach makes it possible to achieve low hardware requirements with relatively high accuracy and high processing speed but the algorithmic formulation requires to store several entire frames, which requires a large external memory (near 250 MB for store 3 entire frames), this increase the system size and cost.

Table 6.2. Hardware requirements compared with previous FPGA-based approaches. In most cases, our FPGA architecture outperforms the current state of the art.

Method	Logic elements	Memory bits	Image resolution
Martín et al. [87] (2005)	11,520	147,456	256×256
Díaz et al. [37] (2006)	513,216	685,670	320×240
Wei et al. [140] (2007)	10,288	256 MB (DDR)	640×480
Barranco et al. [13] (2012)	82,526	573,440	640×480
Honegger et al. [62] (2012)	49,655	1,111,000	376×240
Our work*	69,879	624,244	640×480
Our work*	37,059	163,122	320×240
Our work*	21,659	85,607	256×256

*Operating frequency = 50 MHz. Three different versions of the developed algorithm were synthetized, setting the input image resolution as 640×480, 320×240 and 256×256, respectively.

In **Table** 6.3, speed processing for different image resolutions is shown. We synthesized different versions of our FPGA architecture (**Fig.** 6.4), and we adapted the circular buffers in order to work with all tested image resolutions (**Table** 6.2). Then, we carried out post-synthesis simulation in ModelSim Altera. In all cases, our FPGA architecture reached real-time processing. When compared with previous work (**Table** 6.3), our algorithm provided the highest speed processing, it outperforms several previous work [13, 37, 62, 87, 140], and for HD images, our algorithm reaches real-time processing: more than 60 fps for 1280×1024 image resolution.

Table 6.3. Processing speed compared with previous FPGA-based approaches. In all cases, our FPGA architecture outperforms the current state of the art.

Method	Resolution	Frames/s	Pixels/s
Martín et al. [87]	256×256	60	3,932,160
Díaz et al. [37]	320×240	30	2,304,000
Wei et al. [140]	640×480	64	19,550,800
Barranco et al. [13]	640×480	31	9,523,200
Honegger et al. [62]	376×240	127	11,460,480
Our work*	1280×1024	68	90,129,200
Our work*	640×480	297	91,238,400
Our work*	320×240	1,209	92,880,000
Our work*	256×256	1,417	92,876,430

*Four different versions of the developed algorithm were synthetized, setting the input image resolution as 1280×1024, 640×480, 320×240 and 256×256, respectively.

In **Fig.** 6.10, qualitative results for this work compared with previous work are shown. We used the "Garden" dataset since previous work [37, 87, 140] used this dataset as reference. When compared with previous work (**Fig.** 6.10), our algorithm provides high performance under real world scenarios, it outperforms several previous work [37, 87, 140], quantitatively closer to the ground truth (error near to 9%) compared with other FPGA-based approaches. In **Fig.** 6.11, quantitative and qualitative results for the KITTI dataset [50], are shown. In all cases our algorithm provides high performance, it reaches an acceptable error near to 10% with several test sequences.

6.8.2 The feature matching step: performance and limitations

In **Table** 6.4, accuracy comparisons are shown. In the case of the SURF/ORB [15, 113] algorithms, the image degradation between viewpoints introduces data inconsistences that generate erroneous matches. For the KLT algorithm, accurate tracking can be reached, as shown in **Table** 6.4, however, previous works have demonstrated that iterative operations inside the original KLT formulation [83] limits the processing speed [28]. For our algorithm, it allows high accuracy, superior to SURF/ORB algorithms and with the capability to be implemented inside GPU devices. This makes possible to reach real-time processing, higher processing sped than KLT-based algorithms.

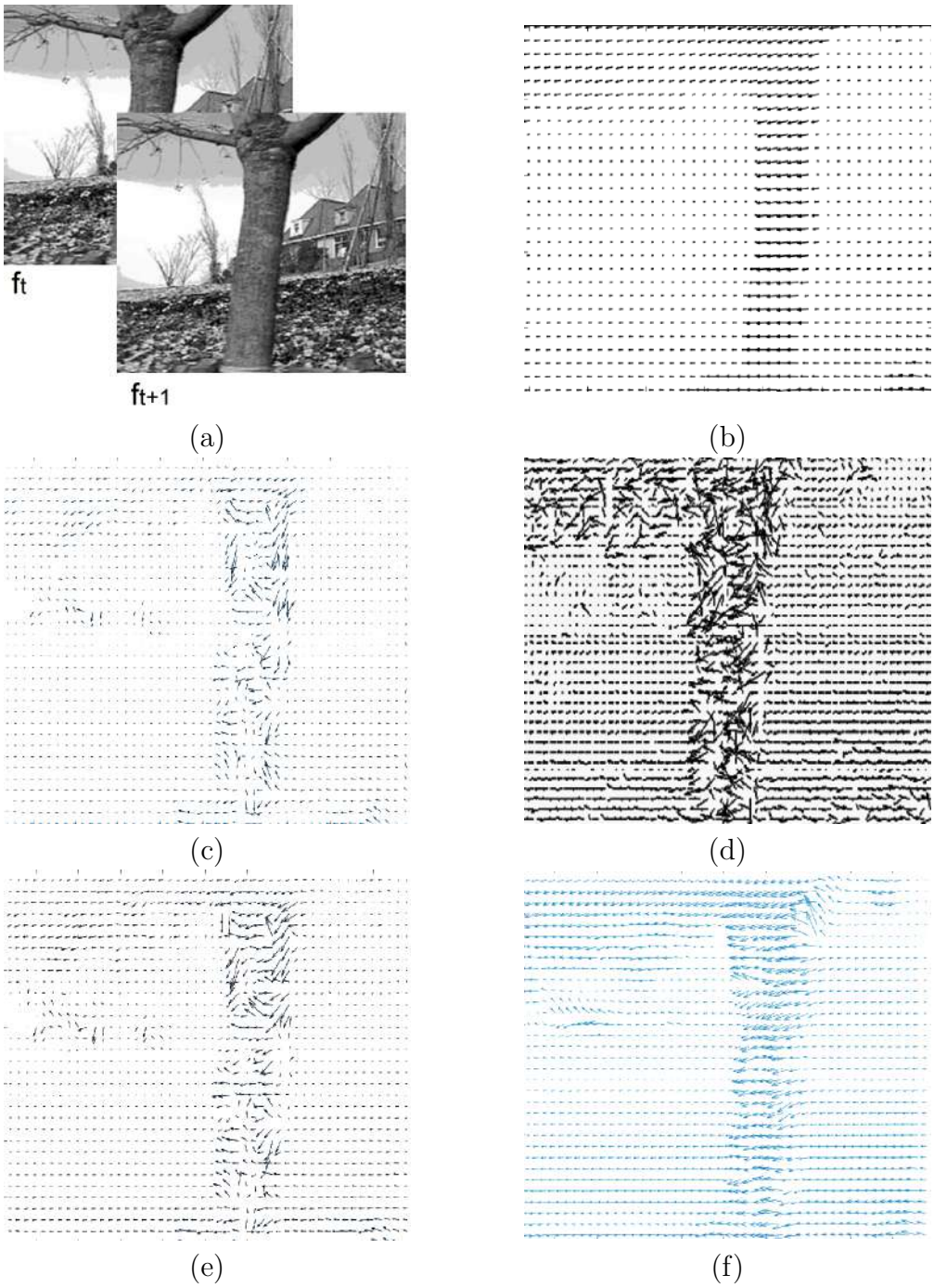


Figure 6.10. Accuracy comparisons for different FPGA-based pixel tracking algorithms. (a) Input data. (b) Ground truth. (c) Martín et al. [87]. (d) Wei et al. [140]. (e) Díaz et al. [37]. (f) This work (error = 9%). In all cases, our algorithms outperforms the current state of the art.

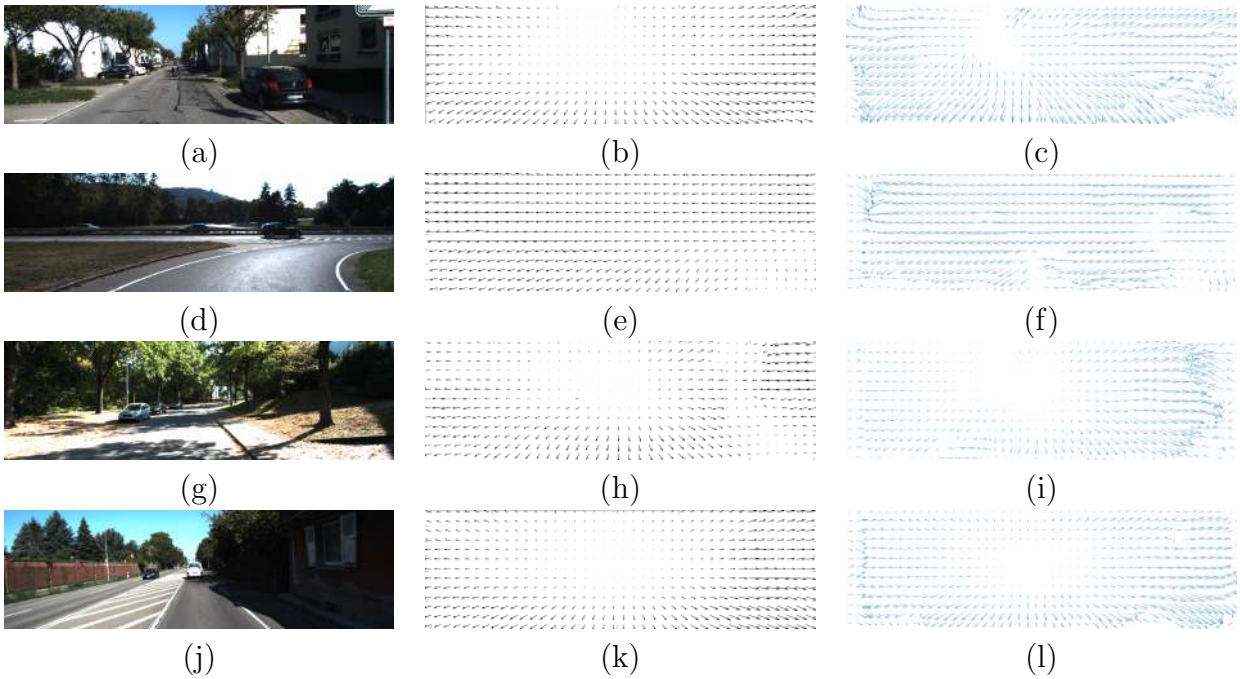


Figure 6.11. Pixel tracking results for the KITTI dataset. (a) Sequence 00, reference image. (b) Ground truth. (c) Tracking result (error = 11%). (d) Sequence 01, reference image. (e) Ground truth. (f) Tracking result (error = 12%). (g) Sequence 02, reference image. (h) Ground truth. (i) Tracking result (error = 11%). (j) Sequence 04, reference image. (k) Ground truth. (l) Tracking result (error = 12%).

Table 6.4. Accuracy comparisons for feature-matching algorithms used in SLAM formulations. Eerrors are measured in pixels, all sequences were obtained from [126]).

TUM dataset	**SURF** [**15**]	**ORB** [113]	**KLT** [83]	**proposed**
fr1/room	79.38	76.24	0.21	1.97
fr2/desk	81.12	73.63	0.45	1.76
fr1/plant	77.74	75.24	0.39	1.83
fr1/teddy	83.53	76.73	0.47	1.94
fr2/coke	80.78	75.28	0.32	1.73
fr2/dishes	78.25	74.19	0.01	1.67
fr3/cabinet	79.87	74.02	0.42	1.71
fr3/teddy	79.10	75.21	0.46	1.83
mean errors =	79.97	75.08	0.34	1.63

6.8.3 The pose estimation step: the proposed dataset

We introduce a new dataset that contains monocular video sequences and ground-truth data with the goal to establish a novel benchmark for the evaluation of visual odometry (VO) algorithms under complex camera movements. Our dataset consists of 130 monocular sequences provided with ground truth trajectories for all sequences. The data was recorded at full frame rate (60 Hz) and, the ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras (Vicon V8, 8MP, 2000Hz, [136]), for more details please see **Appendix A**. In **Table** 6.5 quantitative results for the proposed dataset are shown. For practical purposes, we present 33 possible movements (from a total of 65), so in the first column we indicate the sequence within the proposed dataset, then, the next 6 columns (Movement), indicate what type of movement is addressed in sequence. Finally, the last three columns show quantitative results for different look up tables. For each possible movement, our dataset provides two different sequences recorded at different spatial locations (one for training, another for test).

Given the training sequences shown in **Table** 6.4 $(1, 4, 7, \ldots 58)$ and given their corresponding ground truth, 37482 Q_j elements (see **Eq.** 4.19) could be included in the lookup table. However, there are several repeated or similar elements, then, when $\sigma_1 = 4, \sigma_2 = .5$ are applied, the lookup table is reduced to 7343 elements (19.59% of the full lookup table without thresholding). This reduced lookup table includes all the basic movements in the training sequence and therefore, any other sequences with similar movements, such as in the test sequences, have to be processed with high accuracy, as shown in **Table** 6.4, where average accuracy of 85% is reported. Furthermore, camera egomotion is computed at 77 fps. In other experiment, $\sigma_1 = 8, \sigma_2 = 1$, an average accuracy of 80% is obtained, however, in this case, the lookup table can be reduced to 5394 elements (14.59% of the full lookup table without thresholding), this makes possible to increase the processing speed to 81 fps. Finally, in another test, when $\sigma_1 = 15, \sigma_2 = 2$ were applied, a lookup table with 3749 elements (10.59% of the full lookup table) delivers an average accuracy around 71% combined with processing speed of 97 fps. For all cases, it was demonstrated that a relatively high accuracy for complex movements is possible. In particular, given $\sigma_1 = 4, \sigma_2 = .5$, $\sigma_1 = 8, \sigma_2 = 1$ accuracy around 80% and processing speed between 77-81 fps are achieved. In **Fig.** 6.12, we present qualitative results for the proposed dataset.

Table 6.5. Quantitative results for the pose estimation step under the proposed dataset. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). In all cases, a realtively high accuracy (close to 85%) is possible, using between 15 to 20% of the training data ($\sigma_1 = 4, \sigma_2 = .5$). With a small look-up table ($\sigma_1 = 15, \sigma_2 = 2$), which uses 10.59% of the training data, accuracy about 75% can be obtained.

	Movement						**Accuracy**		
	x	**y**	**z**	α	β	γ	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
0	-	-	-	-	-	-	81	85	73
1	-	-	-	-	-	×	81	80	71
3	-	-	-	-	×	×	81	81	69
6	-	-	-	×	×	-	80	82	72
7	-	-	-	×	×	×	80	85	70
9	-	-	×	-	-	×	85	82	75
12	-	-	×	×	-	-	88	80	73
13	-	-	×	×	-	×	83	82	74
15	-	-	×	×	×	×	83	83	74
18	-	×	-	-	×	-	88	81	74
19	-	×	-	-	×	×	88	79	73
21	-	×	-	×	-	×	86	83	73
24	-	×	×	-	-	-	88	82	73
25	-	×	×	-	-	×	86	81	71
27	-	×	×	-	×	×	83	79	72
30	-	×	×	×	×	-	85	81	75
31	-	×	×	×	×	×	82	80	69
33	×	-	-	-	-	×	82	80	72
36	×	-	-	×	-	-	86	84	72
37	×	-	-	×	-	×	84	81	75
39	×	-	-	×	×	×	83	79	73
42	×	-	×	-	×	-	85	81	75
43	×	-	×	-	×	×	86	84	73
45	×	-	×	×	-	×	85	80	74
48	×	×	-	-	-	-	84	85	72
49	×	×	-	-	-	×	85	81	72
51	×	×	-	-	×	×	84	81	70
54	×	×	-	×	×	-	80	80	73
55	×	×	-	×	×	×	86	84	75
57	×	×	×	-	-	×	88	79	72
60	×	×	×	×	-	-	83	85	72
61	×	×	×	×	-	×	88	80	74
63	×	×	×	×	×	×	83	81	69

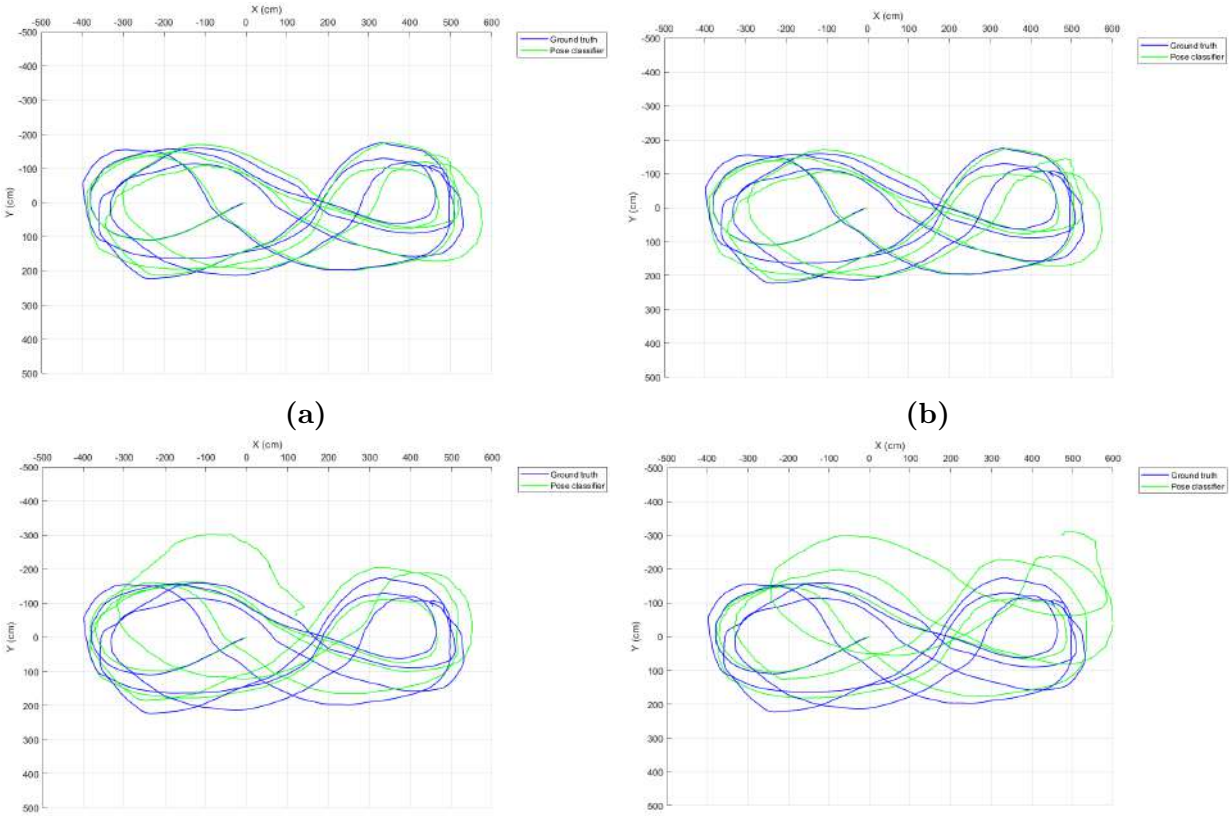


Figure 6.12. Performance for the pose estimation step under the proposed dataset. Sequence 48 which consists of a x, y, α camera movement and validates the performance under loop trajectories. σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). (a) $\sigma_1 = 4, \sigma_2 = .5$: Large look-up tables (using 20% of the training data) makes possible accurate ego-motion estimation (close to the ground truth) but processing speed decrease. For (c) $\sigma_1 = 25, \sigma_2 = 4$ and (d) $\sigma_1 = 30, \sigma_2 = 5$: Small look-up tables (using 5% of the training data) reach high processing speed but accuracy is low. In practice, look-up tables using 10-15% of the training data ($\sigma_1 = 8, \sigma_2 = 1$) deliver a good tradeoff between accuracy and processing speed, as shown in (b).

6.8.4 The pose estimation step: the KITTI dataset

In a first experiment, we carried out a cross validation for the training sequences (sequences 00-10 with public ground truth), that is, we construct the lookup table using all sequences in the training set, except the sequence being evaluated, (see **Table** 6.6). Given the training sequences and given their corresponding ground truth, 21732 Q_j elements are available. For $\sigma_1 = 4, \sigma_2 = .5$, a lookup table of 13791 elements is used. This configuration delivers accuracy around 91%. For $\sigma_1 = 8, \sigma_2 = 1$, average accuracy of 88% is achieved, in this case, the lookup table was reduced to 7371 elements. Finally, for $\sigma_1 = 15, \sigma_2 = 2$, a lookup table with 1749 elements was used, average accuracy around to 75% was achieved. For all cases, relatively high accuracy was obtained. In particular, for $\sigma_1 = 4, \sigma_2 = .5$, $\sigma_1 = 8, \sigma_2 = 1$, accuracy around 91% and processing speed equal to 200 fps are possible. In all cases high processing speed is (200 fps) is possible. In **Fig.** 6.13, we present qualitative results for the data presented in **Table** 6.6.

Table 6.6. Quantitative results for the KITTI dataset. σ_1, σ_2 are the threshold values used for the look up table reduccion. Large look-up tables ($\sigma_1 = 4, \sigma_2 = .5$) deliver error lower than 8% while small look-up tables ($\sigma_1 = 15, \sigma_2 = 2$) reach high processing speed. Look-up tables using 10-15% of the training data ($\sigma_1 = 8, \sigma_2 = 1$), deliver a good tradeoff between accuracy and processing speed (mean eror of 11%).

	Accuracy		
	$\sigma_1 = 4, \sigma_2 = .5$	$\sigma_1 = 8, \sigma_2 = 1$	$\sigma_1 = 15, \sigma_2 = 2$
00	90.35	88.92	76.30
01	92.60	88.15	75.90
02	90.48	87.12	76.09
03	89.44	87.47	75.59
04	92.12	87.70	76.49
05	90.56	88.65	75.37
06	89.96	87.03	76.38
07	90.61	87.08	75.36
08	89.38	87.33	75.74
09	89.52	88.30	76.25
10	92.77	88.47	76.56

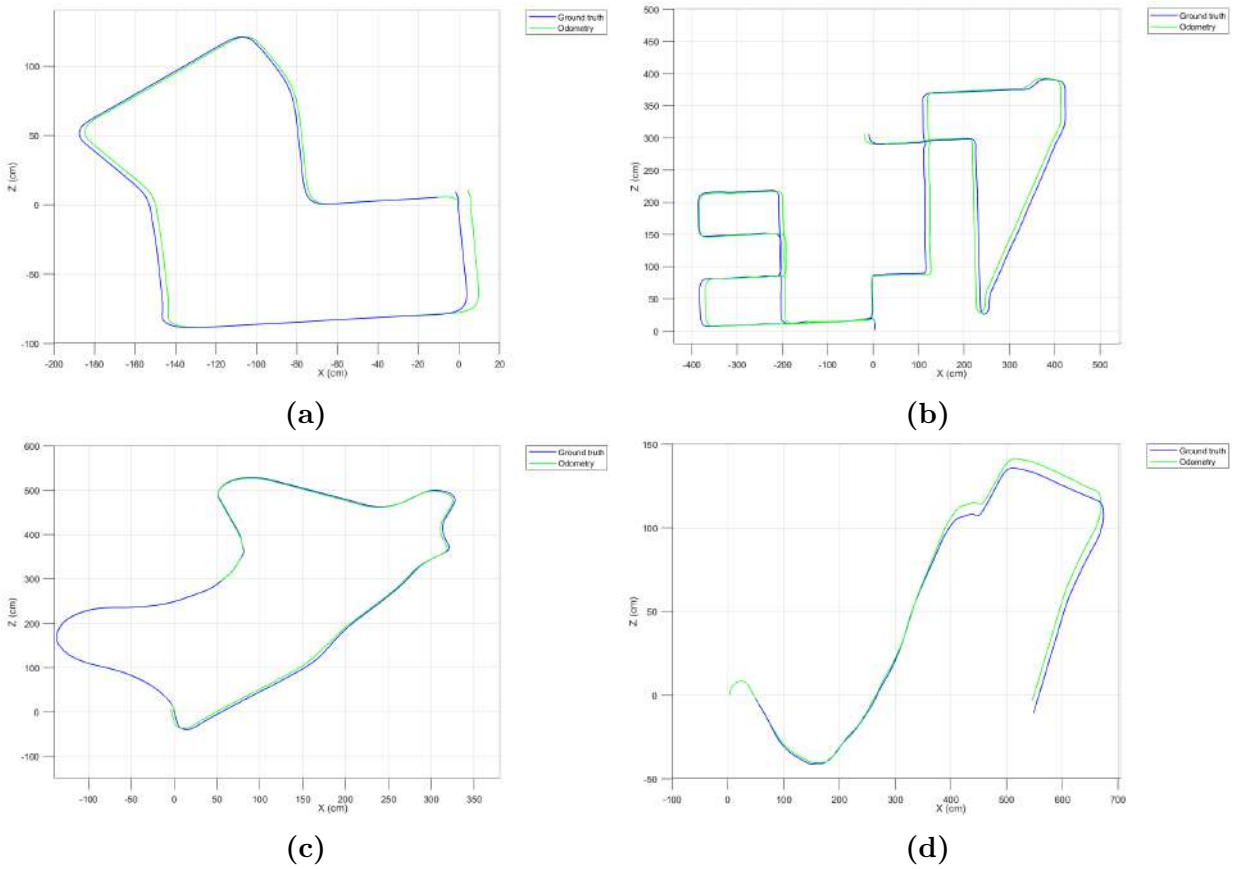


Figure 6.13. Performance for the KITTI dataset (training sequences). σ_1, σ_2 are the threshold values used for the look up table reduccion (**Section** 4.6). Setting $\sigma_1 = 4, \sigma_2 = .5$: For video sequences in which most of the time the environment is rigid, high accuracy (near to 92%) can be reached (a) (b) and (c). For video sequences with dynamic objects (d) the accuracy level decreases (accuracy of 89% can be reached).

6.8.5 The depth from motion step: performance and limitations

In **Fig.** 6.14, the quantitative and qualitative results (RMS error and depth maps, respectively) are shown for the KITTI dataset [50]. In all cases our algorithm provides rough depth maps compared with stereo-based or deep learning approaches [23, 47] but with real-time processing and with the capability to be implemented in embedded hardware, suitable for smart cameras. To our knowledge, previous FPGA-based approaches are limited; there are several GPU-based approaches but in these cases most of the effort was for accuracy improvements and real-time processing or embedded capabilities were not considered so, in several cases, details about the hardware requirements or the processing speed are not provided [144, 147]. In **Table** 6.7 quantitative comparisons between our algorithm and

the current state of the art are presented. For previous works, the RMS error, hardware specifications and processing speed were obtained from the published manuscripts while for our algorithm we computed the RMS error as indicated by the KITTI dataset, [134]. For accuracy comparisons, most of the previous works [85, 143, 144, 147, 148] outperform our algorithm (almost 15% more accurate than ours); however, our algorithm outperforms all of them in terms of processing speed (a processing speed up to 128 times faster than previous works) and with embedded capabilities (making it possible to develop a smart camera/sensor suitable for embedded applications).

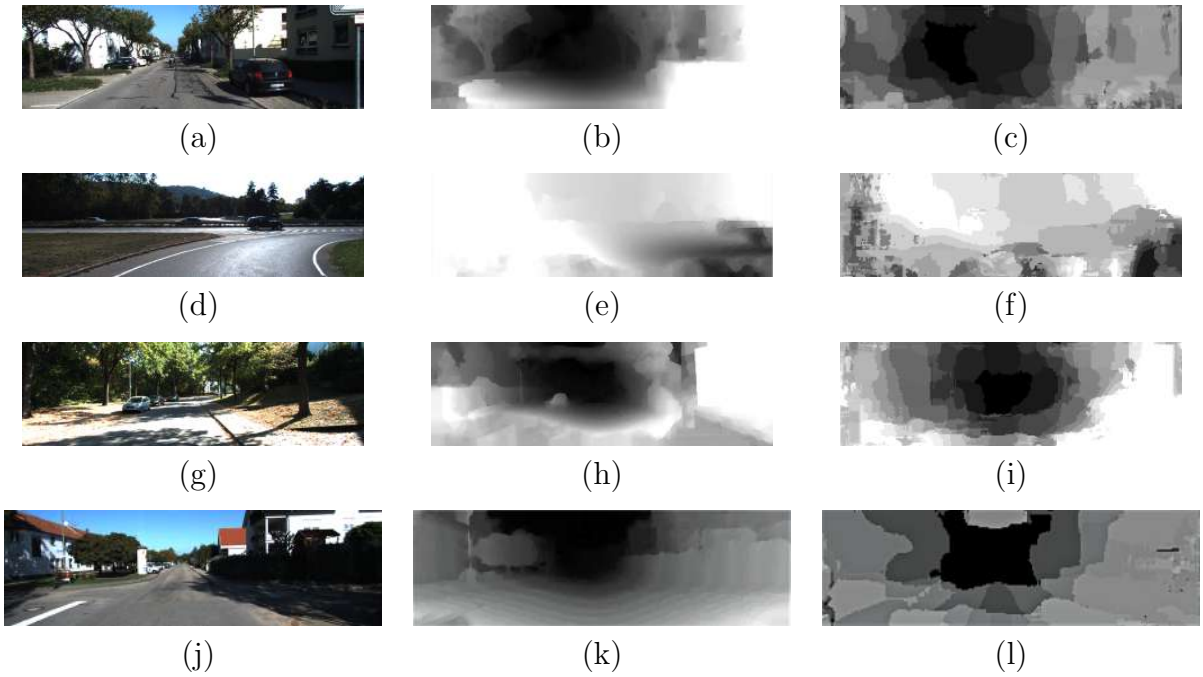


Figure 6.14. Depth from motion: results for the KITTI dataset. (a) Sequence 00, reference image. (b) Ground truth. (c) Depth estimation (error = 22%). (d) Sequence 01, reference image. (e) Ground truth. (f) Depth estimation (error = 22%). (g) Sequence 02, reference image. (h) Ground truth. (i) Depth estimation (error = 22%). (j) Sequence 05, reference image. (k) Ground truth. (l) Depth estimation (error = 23%). In all cases our algorithm provides rough depth maps with the capability to be implemented in hardware, suitable for smart cameras or embedded robotic applications.

Table 6.7. Depth estimation compared with the current state of the art. Training sequences of the KITTI dataset. Most of the previous works outperform our algorithm in terms of accuracy; however, our algorithm outperforms all of them in terms of processing speed. Further, our approach provides the unique algorithm with embedded capabilities.

Method	Error (RMS)	Speed	Image resolution	Hadware
[147]	6.8%	-	128×416	-
[143]	6.5%	5 fps	128×416	GTX 1080 (GPU)
[85]	6.2%	100 fps	128×416	Titan X (GPU)
[144]	6.2%	-	830×254	Titan X (GPU)
[148]	5.6%	1.25 fps	576×160	Tesla K80 (GPU)
Our work	21.5%	192 fps	1241×376	Cyclone IV (FPGA)

6.9 DreamCam Validation

DreamCam is a robust/flexible smart camera [20], see **Fig.** 6.15. In a final experiment, we have validated our FPGA architecture inside the DreamCam **Fig.** 6.16 where it was demonstrated the effectivity of our algorithmic approach.

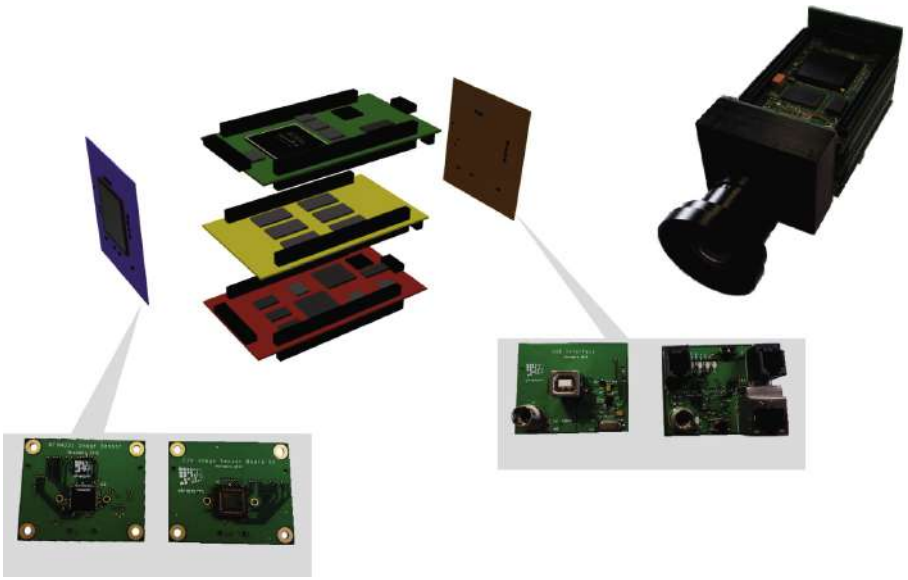


Figure 6.15. The DreamCam FPGA prototype.

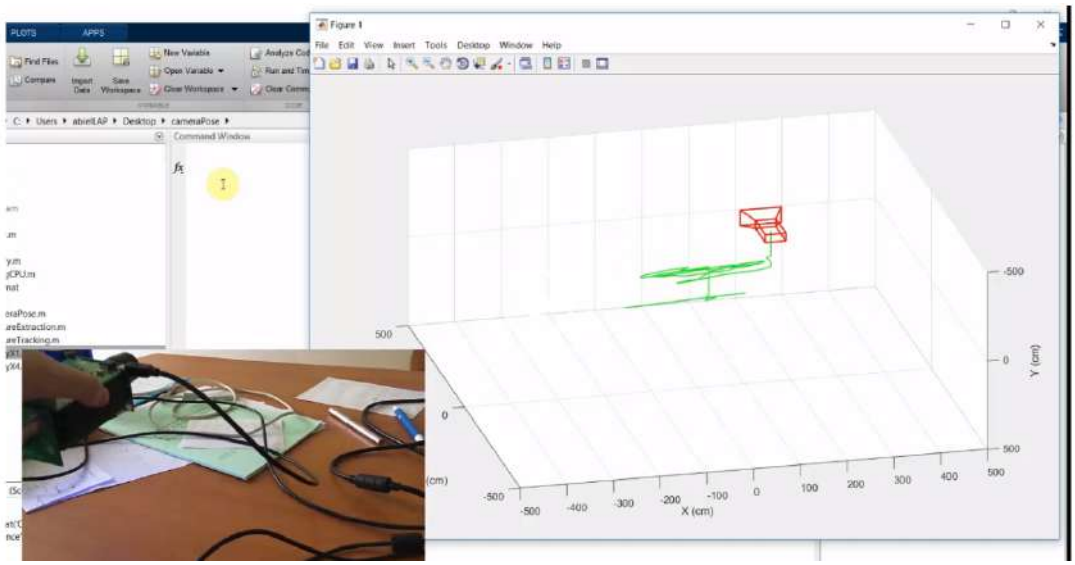


Figure 6.16. The DreamCam validation. High performance under real world indoor scenarios is possible. For a video demo please see [7].

6.10 Global performance: GPU vs FPGA

In previous sections, the performance and limitations for each step of our algorithmic formulation were presented. It was demonstrated that in most of the cases, our formulation improves the current state of the art (in terms of accuracy, embedded capabilities or processing speed). The aim of this section is to determine the global performance and scope of the full formulation presented in this work. For that, we consider the full formulation as a monocular-SLAM system in which the global performance can be estimated by measuring four different variables, localization accuracy, processing speed, hardware/power requirements and mapping density.

6.10.1 Localization accuracy

For localization accuracy, in **Table** 6.8 we show quantitative comparisons between simulation results and implemented results in FPGA and CUDA. For $\sigma_1 = 4, \sigma_2 = .5$, a lookup table of 13791 elements is used. This configuration delivers accuracy around 97% for the CPU simulation. It is the highest accuracy that our monocular-SLAM system can reach but the formulation uses double-precision floating-point format (which is not feasible and suitable under GPU/FPGA hardware architectures), and eliminates the real-time processing constrain from the system. The GPU implementation reaches 95% of accuracy that

is the closest to the simulation results. In this case the formulation uses single-precision floating-point format and this is the main reason of the accuracy detriment. For the FPGA implementation, the implementation uses integer format in order to fulfill the VHDL limitations. However, this involves an important detriment in the level of accuracy. As a result, the accuracy decreases to 91%

Table 6.8. Localization accuracy for the KITTI dataset. σ_1, σ_2 are the threshold values used for the look up table reduction: $\sigma_1 = 4, \sigma_2 = .5$ allow accuracy around 97% for the CPU simulation while for the GPU and FPGA implementations, accuracy around 95% and 91%, respectively, are achieved.

	Accuracy (compared to ground truth)		
	CPU	GPU	FPGA
00	97.47	95.77	90.35
01	97.48	95.93	92.60
02	97.16	95.12	90.48
03	97.58	95.56	89.44
04	97.92	95.46	92.12
05	97.27	95.01	90.56
06	97.21	95.33	89.96
07	97.09	95.16	90.61
08	97.13	95.79	89.38
09	97.35	95.31	89.52
10	97.83	95.52	92.77

6.10.2 Processing speed

Regarding processing speed, in **Table** 6.9 quantitative comparisons between CPU, GPU and FPGA are shown. For $\sigma_1 = 4, \sigma_2 = .5$, a lookup table of 13791 elements is used. For the simulation results, the processing speed is unfeasible and unsuitable for practical purposes. It requires several hours to process a single training sequence whose average lengths are 30 seconds. The GPU implementation reaches real-time processing. And in most of the cases, this speed should be enough for several real world applications. On the other hand, for the FPGA implementation, it delivers the highest processing speed that our monocular-SLAM system can reach.

Table 6.9. Processing speed for the KITTI dataset. σ_1, σ_2 are the threshold values used for the look up table reduction: $\sigma_1 = 4, \sigma_2 = .5$ allow real-time processing only for the GPU and FPG implementations.

	Processing speed (measured in fps)		
	CPU	GPU	FPGA
00	0.11 fps	87.34 fps	192 fps
01	0.12 fps	86.45 fps	192 fps
02	0.11 fps	83.15 fps	191 fps
03	0.11 fps	87.74 fps	191 fps
04	0.12 fps	86.22 fps	192 fps
05	0.12 fps	87.45 fps	192 fps
06	0.11 fps	85.54 fps	192 fps
07	0.11 fps	83.13 fps	191 fps
08	0.10 fps	86.49 fps	192 fps
09	0.11 fps	87.92 fps	191 fps
10	0.12 fps	85.11 fps	192 fps

6.10.3 Hardware/power requirements

For hardware/power requirements, in **Table** 6.10 we show quantitative comparisons between simulation requirements and implementation requirements in FPGA and CUDA. For the CPU simulation, we have used an intel i7-8700K processor whose power consumption while running our algorithm is 65 W, which is a reasonable power requirement for a monocular-SLAM system. The GPU implementation uses a GPU GTX 970M of NVIDIA and, it has the highest power consumption since 292 W are required. For the FPGA implementation, it has the lowest power requirements since only 6.3 W are necessary.

6.10.4 Mapping density.

For the mapping step, its density depends on the performance of the pixel tracking and depth from motion steps. In both cases, the single-precision floating-point format or the integer format involve a minimum accuracy detriment compared with the simulation results. Therefore, in both cases (GPU/FPGA), accurate dense mapping (as presented above in **Section** 4.10) is obtained.

Table 6.10. Hardware/power requirements under the KITTI dataset. σ_1, σ_2 are the threshold values used for the look up table reduction: using $\sigma_1 = 4, \sigma_2 = .5$ only the CPU and the FPGA implementations enable embedded capabilities.

	Power requeriments (measured in watts)		
	CPU (i7-8700K)	GPU (GTX 970M)	FPGA (Cyclone IV EP4CGX150CF23C8)
00	65 W	292 W	6.32 W
01	65 W	292 W	6.32 W
02	65 W	292 W	6.32 W
03	65 W	292 W	6.32 W
04	65 W	292 W	6.32 W
05	65 W	292 W	6.32 W
06	65 W	292 W	6.32 W
07	65 W	292 W	6.32 W
08	65 W	292 W	6.32 W
09	65 W	292 W	6.32 W
10	65 W	292 W	6.32 W

6.10.5 Discussion

For localization accuracy, the CPU simulation achieves the higher accuracy that our monocular-SLAM system can reach but due to it uses double-precision floating-point format, this formulation is unfeasible and suitable under GPU/FPGA hardware architectures. Further, this formulation eliminates the real-time processing constrain from the system, that is the one of the most important constrain in a real world application. The GPU implementation reaches similar accuracy than the simulation results and, with real-time processing. In most of the cases, the processing speed of the GPU implementation should be enough for real world applications such as, autonomous navigation for unmanned aerial vehicles, augmented reality, etc. Unfortunately, the power requirements are high. Therefore, embedded capabilities are low. As a result, for a real world application, the monocular-SLAM system should be implemented in a remote device (PC, laptop) while the system (aerial vehicle, glasses of augmented reality) only receives a feedback of the SLAM system and carries out the process action.

FPGA implementation reaches the lowest performance in terms of localization accuracy (near 91% of accuracy). Even though 91%, is a relatively high and acceptable accuracy compared with the GPU implementation. To our knowledge it is the first monocular-SLAM formulation implemented fully in hardware and this makes possible to outperform the GPU version in terms of processing speed (reaching a processing speed near to 200 fps). This could be useful in high speed applications, for example: mobile virtual and augmented reality systems in which processing speed higher than 100 fps is desirable. On the other hand, the FPGA implementation enables embedded capabilities (only 6.32 W are required). This makes possible to develop and FPGA-based smart camera for monocular-SLAM. As a result, for a real world application, the monocular-SLAM system could be implemented inside a sensor (smart camera) and then, this sensor could be connected directly to the system (aerial vehicle, glasses of virtual reality, etc.), eliminating the external device requirement.

6.11 The proposed approach vs visual-SLAM algorithms in the current literature

In a final analysis, we compare our algorithmic formulation (monocular) with other visual-based approaches (stereo, LiDAR, RGBD) in the current literature. Performance comparisons with previous visual-SLAM algorithms were made using the KITTI dataset [50]. For previous work, we obtained the performance data from the published manuscripts. For the proposed algorithm, we have submitted to the KITTI benchmark suite our results for the test sequences (11-21), for more details about our algorithmic performance (qualitative results, drift error graph), please see [8]. For the leader table we are ranked in the 82th place, however, most algorithms in the leader table are stereo-based approaches [10, 42, 44, 52, 64, 68, 89, 94, 107, 121, 122], or LiDAR-based approaches [16, 53]. In these cases, the disparity map or the rangefinder map make possible to compute accurate camera poses (localization) but in most cases, processing speed is low (near 15 fps), as shown in **Table** 6.11. For monocular-based approaches [30, 49], our algorithm outperforms most previous monocular approaches in both, accuracy and processing speed. This is because most previous work uses binary-based feature description and binary-based matching and, these are sensitive to image degradations (illumination changes, rotation, blur, etc.). In our case, the proposed pixel tracking/feature matching steps provide high robustness for image degradations and this improves the performance.

For previous hardware-based implementations [34, 91, 104, 139], our algorithm reach a good tradeoff between accuracy and processing speed. Compared to [34, 139] our algorithm outperforms those works in terms of accuracy, 4% more accurate than [34] and 7% more accurate than [139]. For processing speed, we achieve a speed up of 2 times (67 fps) compared with [34] which reaches 30 fps. [139] outperforms our algorithm in terms of processing speed (+71 fps), however, the GPU used in previous work is highly powerful (2888 CUDA cores) compared to the one used in this work (1664 CUDA cores). For [91], our algorithm reaches similar accuracy and processing speed than that work. Nevertheless, lower hardware resources are required in our case: 1280 CUDA cores in our case compared with the 1664 CUDA cores used in that work. Finally, [104] outperforms our algorithm in terms of accuracy (4% more accurate than our work), but with lower processing speed because the reported value (333.3fps), does not consider the image readout, feature extraction and feature tracking steps, which are the heaviest operations in the VO formulation. For our FPGA implementation, at our knowledge, it is the first FPGA architecture that addressees all the steps in the monocular-SLAM formulation, outperforming all previous works in terms of processing speed and reaching a relatively high accuracy compared with previous monocular-SLAM algorithms, around 91% of accuracy for the test sequences of the KITTI dataset.

Table 6.11. Quantitative results for the proposed algorithm compared with previous works.

Algorithm	Accuracy	Speed	Density	Hardware	Approach
Graeter et al (2018) [53]	98.78%	2 fps	Dense	CPU	LiDAR
Behley & Stachniss (2018) [16]	98.61%	10 fps	Dense	CPU	LiDAR
Mur-Artal et al (2017) [94]	98.85%	16.66 fps	Semi-dense	CPU	Stereo
Pire et al (2017) [107]	98.81%	10 fps	Semi-dense	CPU	Stereo
Engel et al (2015) [42]	98.80%	14.28 fps	Semi-Dense	CPU	Stereo
Meiqing et al (2017) [89]	98.74%	-	Semi-dense	CPU	Stereo

Huai et al (2015) [64]	98.24%	2 fps	Semi-dense	CPU	Stereo
Fanfani et al (2016) [44]	97.86%	2 fps	Semi-dense	CPU	Stereo
Shiyu et al (2012) [122]	97.46%	20 fps	Semi-dense	CPU	Stereo
Alcantarilla et al (2012) [10]	97.31%	1.78 fps	Semi-dense	CPU	Stereo
Gomez-Ojeda & Gonzalez-Jimenez (2016) [52]	96.74%	5 fps	Semi-dense	CPU	Stereo
Kaess et al (2012) [10]	95.83%	1.9 fps	Semi-dense	CPU	Stereo
Holzmann et al (2016) [61]	91.94%	20.30 fps	Sparse	CPU	Monocular
Geiger et al (2011) [49]	83.71%	16.39 fps	Sparse	CPU	Monocular
Ciarfuglia et al (2014) [30]	85.56%	9.09 fps	Sparse	CPU	Monocular
Costante et al (2016) [34]	91.04%	3.27 fps	Sparse	CPU	Monocular
Costante et al (2016) [34]	91.04%	20.83 fps	Sparse	GPU	Monocular
Mohanty et al(2016) [91]	94.50%	111.11 fps	Sparse	GPU	Monocular
Weber et al (2017) [139]	88.53%	158.73 fps	Sparse	GPU	Monocular
Pillai and Leonard (2017) [104]	**99.72**%	333.3 fps	Sparse	GPU	Monocular
Our work	95.07%	87.34 fps	Semi-dense	GPU	Monocular
Our work	91.01%	**206** fps	Semi-dense	FPGA	Monocular

6.12 Summary

In this chapter, we have presented the results of FPGA implementation for our monocular SLAM formulation. Our algorithm of tracking / matching functions based on FPGA offers a dense tracking (more feature points than previous algorithms) and no outliers. This avoids the use of RANSAC outliers filtering and allows a fully parallelization inside FPGA architectures. We have developed an FPGA-based architecture suitable for a smart camera which deliver high efficiency in terms of algorithmic parallelization [7]. So, unlike previous work, the camera ego-motion (localization) can be estimated without iterative behavior and without geometric constraints, suitable for embedded applications. Experimental results demonstrated that our algorithm reaches high accuracy (91.07%), compared with previous monocular VO algorithms such as CNN and depth learning-based algorithms (which reach 90% of accuracy) and with a processing speed up to 30 times faster than previous works.

We have developed a new depth from motion/linear triangulation algorithm whose hardware implementation deliver high efficiency in terms of algorithmic parallelization. Different to previous works, the depth information is estimated in real time within a compact FPGA device. Faster and with fewer hardware resources than previous works. It makes it possible to achieve a dense 3D reconstruction via GPU-based linear triangulation, improving by around 50 times the current state of the art.

Chapter 7

Conclusions and future work

In this chapter, a summary of the proposed algorithm is presented, as well as the contributions obtained from this doctoral research. Then, the conclusions drawn from this work are presented, and a discussion on the hypothesis of this work is carried out. Finally, the limitations of the proposed approach and the future work derived from this work are discussed.

7.1 Summary

Simultaneous Localization and Mapping (SLAM) is the problem of constructing a 3D map while simultaneously tracking the location of an agent within the map. In recent years, the work has focused on monocular-SLAM since it requires only requires a camera in motion and provides visual environmental information that can be exploited to create complex 3D maps while camera poses can be simultaneously estimated. Unfortunately, previous algorithms are based on optimization techniques implemented in sequential processors where sparse tracking has to be used in order to reach real-time processing. This makes possible to reach high accuracy for the camera pose estimation but limits the embedded capabilities and deliver sparse point clouds. To solve this problem, in this work we have proposed a new formulation of monocular-SLAM based on the hypothesis that it is possible to reach high efficiency for embedded applications, increasing the tracking density (and therefore 3D map density and overall positioning and mapping) by reformulating the feature-tracking/feature-matching process in order to reach high performance for embedded hardware architectures, such as FPGA or CUDA.

To obtain dense tracking, as established in our hypothesis, we have proposed a new pixel tracking algorithm that consists of an extension of the stereo matching problem. A pixel-parallel/window-parallel approach based on a Sum of Absolute Differences was proposed. Further, in order to improve the performance of the correlation, the curl of the intensity gradient as preprocessing step was introduced. To validate our hypothesis, two different hardware architectures (FPGA-based and CUDA-based) full compliant for real-time embedded applications were presented. The experimental results shown that it is possible to obtain accurate camera pose estimations. Compared to previous monocular systems, we are ranked as the 5th place in the KITTI benchmark suite, with higher processing speed (we are the faster algorithm in the benchmark) and more than $\times 10$ the point cloud density from previous approaches.

7.2 Discussion on hypothesis

The hypothesis of this work is that it is possible to achieve a high efficiency for integrated applications of monocular SLAM, increasing the point density (and therefore 3D map density and overall positioning and mapping) by reformulating the feature-tracking/feature-matching process. Furthermore, the feature matching process can be parallelized to reach real time performance in an FPGA or CUDA architecture. The experimental results demonstrate the effectiveness of the proposed pixel-tracking/feature matching algorithms in terms of its embedded monocular-SLAM capabilities. With that algorithms, a novel approach for the "ego-motion challenge" makes possible to estimate the camera ego-motion (localization) with no iterative loop and no geometrical constraints. Furthermore, using the proposed pixel-tracking algorithm, a novel depth from motion algorithm based on a flow/depth transformation delivers depth maps and 3D scene reconstructions with more than $\times 10$ pixels density compared with previous approaches. Finally, the experimental results demonstrated that our FPGA/GPU implementation deliver high efficiency in terms of algorithmic parallelization and this makes possible high efficiency under embedded monocular-SLAM applications.

7.3 Main Contributions

The main contributions of this thesis are:

1. A pixel tracking/feature matching algorithm that provides dense feature tracking/ feature matching. Previous works use high order metrics such as, the Jacobian or Laplacian of the patch being processed to obtain a robust feature tracking, however, this limits the processing speed. To address this problem, feature tracking has to be carried out only for a few pixels in the input image, generating sparse tracking. In our case, we carried out feature tracking for all pixels in the input image, for that, a local correlation function such as, SAD (Sum of Absolute Differences) was used, although local correlation functions have low robustness under image degradations. In order to improve the SAD performance, we have proposed the curl of the input image as a preprocessing step. Then, a pixel-parallel/window-parallel approach based on a SAD correlation function was proposed.

2. A parallel algorithm for the "ego-motion challenge". Previous algorithms are based on optimization techniques that require a high order of iterations to converge. In embedded systems where the clock speed is limited, this decreases the performance for real-time processing. To address this limitation, we have proposed a novel algorithm that consists of using look-up tables and a new feature matching algorithm. Unlike the previous, the movement of the camera is estimated with no iterative loop and no geometrical constraints which makes possible an efficient parallelization for embedded architectures such as, FPGA or CUDA.

3. An algorithm of depth from the movement based on a transformation of flow / depth. In recent years, the most popular solution is the use of active vision to estimate the depth information of the scene. However, this approach increases the size and cost of the system and is limited to indoor scenarios, in which the objects distribution and controlled illumination guarantees the correct propagation for the structured light. In order to reach high performance for embedded applications and high robustness for indoor/outdoor scenarios, in this work, we have introduced a flow/depth transformation inspired by the epipolar geometry. Then, depth in the scene is computed as the norm of the optical flow. This was Inspired by the Euclidean vector operations in which the Z component is proportional to the norm of the X, Y

components. So, for a single moving camera, we supposed that depth in the scene z has to be proportional to the norm of the x, y pixels velocities across time.

4. A Visual Odometry (VO) benchmark dataset. Several well know benchmark datasets such as the KITTI [50], TUM [126] etc. are available, however, most of them were focused on a particular real world application (autonomous vehicle navigation, indoor navigation, etc.). Then, camera movement is limited between two or three degrees of freedom. In order to test our algorithm under more complex movements, a new benchmark dataset was developed. Our benchmark dataset provides camera movements for each possible combination of movements given six degree of freedom (130 different video sequences with ground truth where all possible movement (65) were recorded at two different locations). For download: images, ground truth, documentations and scripts, please see [9].

5. A GPU hardware architecture for our monocular-SLAM formulation. Previous GPU-based approaches have parallelized only some parts of the original monocular-SLAM formulation, for example, the camera pose or the refinement have iterative operations that cannot be parallelized. In our case, our formulation was designed for a parallel implementation then, all parts of our monocular-SLAM formulation can be parallelized. As a result, our approach provides a processing speed up to 17 times faster than in previous works. Furthermore, we can provide high accuracy (95.07%), and making possible to reach dense 3D reconstruction, improving by around 15 times the current state of the art in terms of map density.

6. An FPGA hardware architecture for our monocular-SLAM formulation. To our knowledge our FPGA architecture is the first to solve the monocular-SLAM problem fully in hardware. It offers dense tracking (more feature points than previous CPU/GPU-based algorithms) and without outliers. We have validated our FPGA architecture in an FPGA-based smart camera where experimental results demonstrated that our approach reaches high accuracy (91.07%), compared with previous monocular systems such as CNN and depth learning-based algorithms (which reach 90% of accuracy); with a processing speed up to 30 times faster than previous works and improving around 50 times the current state of the art in terms of map density. Based on these characteristic, we believe that several embedded applications such as augmented reality, mobile robotics, autonomous flying, etc., can obtain advantages by applying and exploiting the embedded capabilities of our approach.

7.4 Future work

The results obtained in this thesis have shown a high efficiency for embedded monocular-SLAM applications. From these results, we propose as future work:

1. To improve the use of hardware resources for the FPGA architecture. As a future work, we will reformulate our FPGA architecture using a popcount-based correlation function (a codification in which a eight bit register is converted based on its number of bits equal to one) into the pixel-tracking step. We believe that this reformulation can reduce the use of logical elements near to 75%, therefore, achieving a higher resolution of follow-up.

2. In order to achieve a better scene understanding, as future work we will investigate the semantic segmentation within the depth from motion step. We believe that depth maps combined with semantic labels could be useful to construct semantic 3D reconstructions (semantic-monocular-SLAM) and, these maps can be exploited by several reald world applications such as, augmented reality, autonomous vehicle navigation, service robots, etc.

7.5 Publications

The following publications were generated as result of this doctoral research:

JCR Journals:

1. Aguilar-González, A., Arias-Estrada, M., & Berry, F. (2019). Monocular-SLAM: A survey. International Journal of Advanced Robotic Systems (IJARS). IF: 0.952 **Submitted**.

2. Aguilar-González, A., Arias-Estrada, M., Berry, F., & Osuna-Coutiño, J. A. de Jesús (2019). The Fastest Visual Ego-motion Algorithm in the West. Microprocessors and Microsystems, 67, 103-116. IF: 1.049 .

3. Aguilar-González, A., Arias-Estrada, M., & Berry, F. (2019). Depth from motion algorithm and hardware architecture for smart cameras. Sensors - Special Issue "Depth Sensors and 3D Vision". IF: 2.475

4. Aguilar-González, A., Arias-Estrada, M., & Berry, F. (2018). Robust feature extraction algorithm suitable for real-time embedded applications. Journal of Real-Time Image Processing, 14(3), 647-665. IF: 2.011

Conference proceedings:

1. Aguilar-González, A., & Arias-Estrada, M. (2016, September). Towards a smart camera for monocular SLAM. In Proceedings of the 10th International Conference on Distributed Smart Camera (pp. 128-135). ACM. Best Paper Award

2. Aguilar-González, A., & Arias-Estrada, M. (2016, October). Dense mapping for monocular-SLAM. In Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on (pp. 1-8). IEEE.

3. Aguilar-González, A., Arias-Estrada, M., & Berry, F. (2017, September). Dense Feature Matching Core for FPGA-based Smart Cameras. In Proceedings of the 11th International Conference on Distributed Smart Cameras (pp. 41-48). ACM.

4. Osuna-Coutiño, J. A. de Jesús, Aguilar-González, A., & Arias-Estrada, M. (2017, September). GPU-based Visual Odometry for Autonomous Vehicle Applications. In Proceedings of the 11th International Conference on Distributed Smart Cameras (pp. 210-211). ACM.

5. Aguilar-González, A., Arias-Estrada, M., & Berry, F. (2017, September). Camera Pose Estimation Suitable for Smart Cameras. In Proceedings of the 11th International Conference on Distributed Smart Cameras (pp. 202-204). ACM.

Appendices

Appendix A

INAOE/DREAM benchmark dataset

We provide a new dataset that contains monocular video sequences and ground-truth data with the goal to establish a novel benchmark for the evaluation of visual odometry algorithms under complex camera movements. Our dataset consists of 130 monocular sequences provided with ground truth trajectories for all the sequences. The data was recorded at full frame rate (60 Hz) and, the ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras (Vicon V8, 8MP, 2000Hz, [136]), a shown in **Fig.** A.1.

Figure A.1. INAOE/DREAM benchmark dataset setup. The data was recorded at full frame rate (60 Hz) and, the ground-truth trajectory was obtained from a high-accuracy motion-capture system with eight high-speed tracking cameras Vicon V8, 8MP, 2000Hz.

How can I use the INAOE/DREAM Benchmark dataset?

1. Download one or more benchmark sequences
2. Run your favorite visual odometry/visual SLAM algorithm
3. Save the estimated camera trajectory to a file
4. Evaluate your algorithm by comparing the estimated trajectory with the ground truth trajectory.

In the following table you can download all the benchmark sequences. Each sequence involve complex camera movements, where the dominant movements are as defined in the table. Each file contains both a folder with the image sequence and a .txt file with the corresponding ground truth. For the ground truth file each row correspond with the camera pose for the current frame (p1,1; p1,2; p1,3; x; p2,1; p2,2; p2,3; y; p3,1; p3,2; p3,3; z), where p(n,m) are the nine elements of the rotation matrix while (x,y,z) are the translation matrix. For the online version please see [9].

Table A.1. The INAOE/DREAM benchmark dataset.

	Movement						**Download**	
	x	**y**	**z**	α	β	γ	1	2
0	-	-	-	-	-	-	donwload	donwload
1	-	-	-	-	-	×	donwload	donwload
2	-	-	-	-	×	-	donwload	donwload
3	-	-	-	-	×	×	donwload	donwload
4	-	-	-	×	-	-	donwload	donwload
5	-	-	-	×	-	×	donwload	donwload
6	-	-	-	×	×	-	donwload	donwload
7	-	-	-	×	×	×	donwload	donwload
8	-	-	×	-	-	-	donwload	donwload
9	-	-	×	-	-	×	donwload	donwload
10	-	-	×	-	×	-	donwload	donwload
11	-	-	×	-	×	×	donwload	donwload
12	-	-	×	×	-	-	donwload	donwload
13	-	-	×	×	-	×	donwload	donwload

14	-	-	×	×	×	-	donwload	donwload
15	-	-	×	×	×	×	donwload	donwload
16	-	×	-	-	-	-	donwload	donwload
17	-	×	-	-	-	×	donwload	donwload
18	-	×	-	-	×	-	donwload	donwload
19	-	×	-	-	×	×	donwload	donwload
20	-	×	-	×	-	-	donwload	donwload
21	-	×	-	×	-	×	donwload	donwload
22	-	×	-	×	×	-	donwload	donwload
23	-	×	-	×	×	×	donwload	donwload
24	-	×	×	-	-	-	donwload	donwload
25	-	×	×	-	-	×	donwload	donwload
26	-	×	×	-	×	-	donwload	donwload
27	-	×	×	-	×	×	donwload	donwload
28	-	×	×	×	-	-	donwload	donwload
29	-	×	×	×	-	×	donwload	donwload
30	-	×	×	×	×	-	donwload	donwload
31	-	×	×	×	×	×	donwload	donwload
32	×	-	-	-	-	-	donwload	donwload
33	×	-	-	-	-	×	donwload	donwload
34	×	-	-	-	×	-	donwload	donwload
35	×	-	-	-	×	×	donwload	donwload
36	×	-	-	×	-	-	donwload	donwload
37	×	-	-	×	-	×	donwload	donwload
38	×	-	-	×	×	-	donwload	donwload
39	×	-	-	×	×	×	donwload	donwload
40	×	-	×	-	-	-	donwload	donwload
41	×	-	×	-	-	×	donwload	donwload
42	×	-	×	-	×	-	donwload	donwload
43	×	-	×	-	×	×	donwload	donwload
44	×	-	×	×	-	-	donwload	donwload
45	×	-	×	×	-	×	donwload	donwload
46	×	-	×	×	×	-	donwload	donwload

47	×	-	×	×	×	×	donwload	donwload
48	×	×	-	-	-	-	donwload	donwload
49	×	×	-	-	-	×	donwload	donwload
50	×	×	-	-	×	-	donwload	donwload
51	×	×	-	-	×	×	donwload	donwload
52	×	×	-	×	-	-	donwload	donwload
53	×	×	-	×	-	×	donwload	donwload
54	×	×	-	×	×	-	donwload	donwload
55	×	×	-	×	×	×	donwload	donwload
56	×	×	×	-	-	-	donwload	donwload
57	×	×	×	-	-	×	donwload	donwload
58	×	×	×	-	×	-	donwload	donwload
59	×	×	×	-	×	×	donwload	donwload
60	×	×	×	×	-	-	donwload	donwload
61	×	×	×	×	-	×	donwload	donwload
62	×	×	×	×	×	-	donwload	donwload
63	×	×	×	×	×	×	donwload	donwload

Bibliography

[1] A. Aguilar-González and M. Arias-Estrada, "Dense mapping for monocular-SLAM", *In Proceedings of the 2016 IEEE International Conference on Indoor Positioning and Indoor Navigation*, IPIN 2016, Alcala de Henares, Spain, Oct. 2016, pp. 1–8.

[2] A. Aguilar-González and M. Arias-Estrada, "Towards a smart camera for monocular-SLAM", *In Proceedings of the 10th International Conference on Distributed Smart Cameras*, ICDSC 2016, Paris, France, Sep. 2016, pp. 128–135.

[3] A. Aguilar-González, M. Arias-Estrada, M. Pérez-Patricio, and J.L. Camas-Anzueto, "An FPGA 2D-convolution unit based on the CAPH language", *Journal of Real-Time Image Processing*, 2015, pp. 1–15.

[4] A. Aguilar-González, M. Arias-Estrada, and F. Berry, "Camera pose estimation suitable for smart cameras", *In Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017, Stanford, CA, USA, Sep. 2017, pp. 202–204.

[5] A. Aguilar-González, M. Arias-Estrada, and F. Berry, "Dense feature matching core for FPGA-based smart cameras", *In Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017, Stanford, CA, USA, Sep. 2017, pp. 41–48.

[6] A. Aguilar-González, M. Arias-Estrada, and F. Berry, "Robust feature extraction algorithm suitable for real-time embedded applications", *Journal of Real-Time Image Processing*, vol. 14(3), pp. 647–665, 2018.

[7] A. Aguilar-González, "Visual Odometry Algorithm and Architecture for FPGA Acceleration", 2018. [Online]. Available: https://dream.ispr-ip.fr/monocular-slam-algorithm-architecture-fpga-acceleration-2/. [Accessed: 10-Feb-2019].

[8] A. Aguilar-González, "The Fastest Visual Ego-motion Algorithm in the West", 2018. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_odometry_detail.php?&result=fee1ecc5afe08bc002f093b48e9ba98a295a79ed, [Accessed: 10-Feb-2019].

[9] A. Aguilar-González. "INAOE/DREAM benchmark dataset", 2018. [Online]. Available: https://dream.ispr-ip.fr/ispr-benchmark-dataset/ [Accessed: 10-Feb-2019].

[10] P. F. Alcantarilla, J. J. Yebes, J. Almazán, and L. M. Bergasa, "On combining visual SLAM and dense scene flow to increase the robustness of localization and mapping in dynamic environments", *In Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, ICRA 2012, Saint Paul, MN, USA, Jun. 2012, pp. 1–6.

[11] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II", *IEEE Robotics & Automation Magazine*, vol. 13(3), pp.108–117, 2006.

[12] S. Y. Bao, M. Bagra, Y. W. Chao, and S. Savarese, "Semantic structure from motion with points, regions, and objects", *In Proceedings of the 2012 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2012, Providence, RI, USA, Jun. 2012, pp. 2703–2710.

[13] F. Barranco, M. Tomasi, J. Diaz, M. Vanegas, and E. Ros, "Parallel architecture for hierarchical optical flow estimation based on FPGA", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20(6), pp.1058–1067, 2012.

[14] J. L. Barron and N. A. Thacker, "Tutorial: Computing 2D and 3D optical flow", *Technical report, Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, Jan. 2007.

[15] H. Bay, A. Ess, T. Tuytelaars, and L. Van-Gool, "Speeded-Up Robust Features (SUFR)", *Computer vision and image understanding*, , vol. 110(3), pp.346–359, 2008.

[16] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3D laser range data in urban environments", *In Proceedings of Robotics: Science and Systems (RSS)*, Pittsburgh, PA, USA, Jun. 2018, pp. 1–10.

[17] S. Benhimane and E. Malis, "Homography-based 2D visual tracking and servoing", *The International Journal of Robotics Research*, vol. 26(7), pp.661–676, 2007.

[18] P. Bergmann, R. Wang, and D. Cremers, "Online photometric calibration of auto exposure video for realtime visual odometry and SLAM", *IEEE Robotics and Automation Letters*, vol. 3(2), pp.627–634, 2018.

[19] P. J. Besl and N. D. McKay, "Method for registration of 3D shapes", *Sensor Fusion IV: Control Paradigms and Data Structures*, vol. 1611, pp.586–607, 1992.

[20] M. Birem and F. Berry, "Dreamcam: A modular FPGA-based smart camera architecture", *Journal of Systems Architecture*, vol. 60(6), pp.519–527, 2014.

[21] S. R. Bista, P. R. Giordano, and F. Chaumette, "Appearance-based indoor navigation by using line segments", *IEEE Robotics and Automation Letters*, vol. 1(1), pp.423–430, 2016.

[22] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation", *In Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, ICRA 2012, Saint Paul, MN, USA, Jun. 2012, pp. 1697–1702.

[23] B. Li, Y. Dai and M. He, "Monocular depth estimation with hierarchical fusion of dilated CNNs and soft-weighted-sum inference", *Pattern Recognition*, vol. 83, pp. 28-339, 2018.

[24] G. Bourmaud and R. Megret, "Robust large scale monocular visual SLAM", *In Proceedings of the 2015 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2015, Boston, MA, USA, Oct. 2012, pp. 1638–1647.

[25] R. G. Brown, P. Y. C. Hwang, *"Introduction to random signals and applied Kalman filtering",* Wiley New York, 1992.

[26] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features", *In Proceedings of the 2010 European conference on computer vision*, ECCV 2010, Crete, Greece, Sep. 2010, pp. 778–792.

[27] S. Caux, E. Hendrickx, F. Berry, M. Pelcat, and J. Sérot, "Demo GPStudio: a toolchain for FPGA-based smart cameras", *In Proceedings of the 10th International Conference on Distributed Smart Camera*, ICDSC 2016, Paris, France, Sep. 2016, pp. 778–792.

[28] D. Chekhlov, A. P. Gee, A. Calway, and W. Mayol-Cuevas. "Ninja on a plane: Automatic discovery of physical planes for augmented reality using visual SLAM", *In Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2007, Nara, Japan, Nov. 2007, pp. 1–4.

[29] Y. Cheng, "Mean shift, mode seeking, and clustering", *IEEE transactions on pattern analysis and machine intelligence*, vol. 17(8), pp. 790–799, 1995.

[30] T. A. Ciarfuglia, G. Costante, P. Valigi, and E. Ricci, "Evaluation of non-geometric methods for visual odometry", *Robotics and Autonomous Systems*, vol. 62(12), pp. 1717–1730, 2014.

[31] J. Civera, A. J. Davison, and J. M. Martinez-Montiel, "Inverse depth parametrization for monocular SLAM", *IEEE transactions on robotics*, vol. 24(5), pp. 932–945, 2008.

[32] A. Concha and J. Civera, "Using superpixels in monocular-SLAM", *In Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, ICRA 2014, Hong Kong, China, May 2014, pp. 365–372.

[33] A. Concha-Belenguer and J. Civera-Sancho, "DPPTAM: Dense Piecewise Planar Tracking And Mapping from a monocular sequence", *In Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems*, IROS 2015, Hamburg, Germany, Sep. 2015, pp. 1–8.

[34] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring representation learning with cnns for frame-to-frame ego-motion estimation", *IEEE robotics and automation letters*, vol. 1(1), pp. 18–25, 2016.

[35] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera SLAM", *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 6, pp. 1052–1067, 2007.

[36] O. Demetz, D. Hafner, and J. Weickert, "The complete rank transform: A tool for accurate and morphologically invariant matching of structure", *In Proceedings of the 2013 British Machine Vision Conference*, BMVC 2013, Bristol , U.K., Sep. 2013, pp. 1–12.

[37] J. Díaz, E. Ros, F. Pelayo, E. M. Ortigosa, and S. Mota, "FPGA-based real-time optical-flow system", *IEEE transactions on circuits and systems for video technology*, vol. 16(2), pp. 274–279, 2006.

[38] Z. Dong, G. Zhang, J. Jia, and H. Bao, "Efficient keyframe-based real-time camera tracking", *Computer Vision and Image Understanding*, vol. 118, pp. 97–110, 2014.

[39] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I", *IEEE robotics & automation magazine*, vol. 13, pp. 99–110, 2006.

[40] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera", *In Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV 2013, Sydney, NSW, Australia, Dec. 2013, pp. 1449–1456.

[41] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct monocular-SLAM", *In Proceedings of the 2014 European Conference on Computer Vision*, ECCV 2014, Zurich, Switzerland, Sep. 2013, pp. 834–849.

[42] J. Engel, Jörg Stückler, and D. Cremers, "Large-scale direct SLAM with stereo cameras", *In Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems*, IROS 2015, Hamburg, Germany, Sep. 2015, pp. 1935–1942.

[43] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry", *IEEE transactions on pattern analysis and machine intelligence*, vol. 40(3), pp. 611–625, 2018.

[44] M. Fanfani, F. Bellavia, and C. Colombo. "Accurate keyframe selection and keypoint tracking for robust visual odometry", *Machine Vision and Applications*, vol. 27(6), pp. 833–844, 2016.

[45] O. D. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2(3), pp. 485–508, 1988.

[46] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast Semi-direct monocular Visual Odometry", *In Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, ICRA 2014, Hong Kong, China, May 2014, pp. 15–22.

[47] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep Ordinal Regression Network for Monocular Depth Estimation", *In Proceedings of the 2018 IEEE In-*

ternational Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, United States, 2018, pp. 1–8.

[48] D. Gálvez-López, M. Salas, J. D. Tardós, and J. M. Montiel, "Real-time monocular object SLAM", *Robotics and Autonomous Systems*, vol. 75, pp. 435–449, 2016.

[49] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3D reconstruction in real-time", *In Proceedings of the 2011 IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, 2011, pp. 963–968.

[50] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset", *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, 2013.

[51] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford and G. Wyeth, "Openfabmap: An open source toolbox for appearance-based loop closure detection", *In Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, ICRA 2012, Saint Paul, MN, USA 2012, pp. 4730–4735.

[52] R. Gomez-Ojeda and J. Gonzalez-Jimenez, "Robust Stereo Visual Odometry through a Probabilistic Combination of Points and Line Segments", *In Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, ICRA 2016, Stockholm, Sweden, May 2016, pp. 1130–1137.

[53] J. Graeter, A. Wilczynski, and M. Lauer, "LIMO: Lidar-Monocular Visual Odometry", *In Proceedings of the 2018 IEEE International Conference on Intelligent Robots and Systems*, IROS 2018, Madrid, Spain, Oct. 2018, pp. 1–8.

[54] A. Graves, S. Lim and T. Fagan, "Visual odometry using convolutional neural networks", *The Kennesaw Journal of Undergraduate Research*, vol. 5(3), pp. 1–10, 2017.

[55] W. N. Greene, K. Ok, P. Lommel, and N. Roy, "Multi-level mapping: Real-time dense monocular-SLAM", *In Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, ICRA 2016, Stockholm, Sweden, May 2016, pp. 833–840.

[56] C. Harris and M. Stephens, "A combined corner and edge detector", *In Proceedings of the 4th Alvey vision conference*, Manchester, U.K., Aug. 1988, pp. 23.1–23.6.

[57] R. Hartley and A. Zisserman, "Multiple view geometry in computer vision", Cambridge university press, 2003.

[58] R. I. Hartley and P. Sturm. "Triangulation", *Computer vision and image understanding*, vol. 68, pp. 146–157, 1997.

[59] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, "Mesheye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance", *In Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, ICIPSN 2007, Cambridge, Massachusetts, USA, Apr. 2007, pp. 360–369.

[60] C. D. Herrera, K. Kim, J. Kannala, K. Pulli, and J. Heikkilä, "DT-SLAM: Deferred Triangulation for robust SLAM", *In Proceedings of the 2nd International Conference on 3D Vision*, 3DV 2014, Tokyo, Japan, Dec. 2014, pp. 609–616.

[61] T. Holzmann, F. Fraundorfer, and H. Bischof, "Direct stereo visual odometry based on lines", *In Proceedings of the 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, VISAPP 2016, Rome, Italy, Feb. 2016, pp. 1–11.

[62] D. Honegger, P. Greisen, L. Meier, P. Tanskanen, and M. Pollefeys, "Real-time velocity estimation based on optical flow and disparity matching", *In Proceedings of the 2012 IEEE International Conference on Intelligent Robots and Systems*, IROS 2012, Vilamoura, Algarve, Portugal, Oct. 2012, pp. 5177–5182.

[63] B. K. P. Horn and B. G. Schunck, "Determining optical flow", *Artificial intelligence*, vol. 17, pp. 185–203, 1981.

[64] J. Huai, C. Toth, and D. Grejner-Brzezinska, "Stereo-inertial odometry using nonlinear optimization", *In Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation*, ION GNSS+ 2015, Tampa, Florida, USA, Sep. 2015, pp. 1–8.

[65] T. W. Hui, X. Tang, and C. C. Loy, "Liteflownet: A lightweight convolutional neural network for optical flow estimation", *In Proceedings of the 2018 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2018, Salt Lake City, United States, 2018, pp. 1–8.

[66] A. Jaegle, S. Phillips, and K. Daniilidis, "Fast, robust, continuous monocular egomotion computation", *In Proceedings of the 2016 IEEE International Conference on Robotics and Automation*, ICRA 2016, Stockholm, Sweden, May 2016, pp. 773–780.

[67] K. Jo, M. Gupta, and S. K. Nayar, "Spedo: 6 DOF ego-motion sensor using speckle defocus imaging", *In Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV 2015, Santiago, Chile, Dec. 2015, pp. 4319–4327.

[68] M. Kaess, K. Ni, and F. Dellaert, "Flow separation for fast and robust stereo odometry", Kobe press, Japan, 2009.

[69] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces", *In Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2007, Nara, Japan, Nov. 2007, pp. 225–234.

[70] G. Klein and D. Murray, "Improving the agility of keyframe-based SLAM", *In Proceedings of the 2008 European Conference on Computer Vision*, ECCV 2008, Florence, Italy, Oct. 2018, pp. 802–815.

[71] K. R. Konda and R. Memisevic, "Learning visual odometry with a convolutional network", *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.*, VISAPP 2015, Berlin, Germany, Mar. 2015, pp. 486–490.

[72] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization", *In Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, ICRA 2011, Shanghai, China, May 2011, pp. 3607–3613.

[73] A. Kundu, Y.Li, F. Dellaert, F. Li, and J. M. Rehg, "Joint semantic segmentation and 3D reconstruction from monocular video", *In Proceedings of the 2014 European Conference on Computer Vision*, ECCV 2014, Zurich, Switzerland, Sep. 2014, pp. 703–718.

[74] S. H. Lee and J. Civera, "Loosely-coupled semi-direct monocular SLAM", *In Proceedings of the 2018 European Conference on Computer Vision*, ECCV 2018, Munich, Germany, Sep. 2018, pp. 1–8.

[75] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem", *International Journal of Computer Vision*, vol. 81(155), pp. 185–203, 2009.

[76] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning", *In Proceedings of the 2018 IEEE International Conference on Robotics and Automation*, ICRA 2018, Brisbane, Australia, May 2018, pp. 7286–7291.

[77] Y. Li and W. Chu, "A new non-restoring square root algorithm and its VLSI implementations", *In Proceedings of the 1996 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, ICCD'96, Austin, TX, USA, 1996, pp. 538–544.

[78] H. Lim, J. Lim, and H. J. Kim, "Real-time 6-DOF monocular visual SLAM in a large-scale environment", *In Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, ICRA 2014, Hong Kong, China, May 2014, pp. 1532–1539.

[79] Jo. Lim, J. M, Frahm, and M, Pollefeys, "Online environment mapping", *In Proceedings of the 2011 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2011, Colorado Springs, CO, USA, Jun. 2011, pp. 1–6.

[80] H. Liu, G. Zhang, and H. Bao, "Robust keyframe-based monocular SLAM for augmented reality", *In Proceedings of the 15th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2016, Merida, Mexico, Sep. 2016, pp. 1–10.

[81] H. C. Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections", *Nature*, vol. 293, pp. 133–135, 1981.

[82] D. G. Lowe, "Object recognition from local scale-invariant features", *In Proceedings of the 1999 IEEE International Conference on Computer Vision*, ICCV 1999, Corfu, Greece, Sep. 1999, pp. 1150–1157.

[83] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision", *In Proceedings of the 7th International Joint Conference on Artificial Intelligence*, IJCAI'81, Vancouver, BC, Canada, Aug. 1981, pp. 674–679 .

[84] W. Maddern and P. Newman, "Real-time probabilistic fusion of sparse 3D LiDAR and dense stereo", *In Proceedings of the 2016 IEEE International Conference on Intelligent Robots and Systems*, IROS 2016, Daejeon, Korea, Oct. 2016, pp. 2181–2188.

[85] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints", *In Proceedings of the 2018 International Conference on Computer Vision and Pattern Recognition*, ICPR 2018, Beijing, China, Aug. 2018, pp. 5667–5675.

[86] R. A. Maronna, D. Martin, and R. S. Yohai, "Robust Statistics: Theory and Methods", *Wiley series in probability and statistics*, Wiley New York, 2006.

[87] J. L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro, and U. Bidarte, "Hardware implementation of optical flow constraint equation using FPGAs", *Computer Vision and Image Understanding*, vol. 98(3), pp. 462–490, 2005.

[88] J. Martínez-Carranza and A. Calway, "Unifying planar and point mapping in monocular-SLAM", *In Proceedings of the 2010 British Machine Vision Conference*, BMVC 2010, Aberystwyth, U.K., Sep. 2010, pp. 1–11.

[89] W. Meiqing, L. Siew-Kei, and S. Thambipillai, "A framework for fast and robust visual odometry", *IEEE Transaction on Intelligent Transportation Systems*, vol. 18(12), pp. 3433 - 3448, 2017.

[90] S. Meister, J. Hur, and S. Roth, "Unflow: Unsupervised learning of optical flow with a bidirectional census loss", *In Proceedings of the 2016 British Machine Vision Conference*, BMVC 2016, York, U.K., Sep. 2016, pp. 1–11.

[91] V. Mohanty, S. Agrawal, S. Datta, A. Ghosh, V. .t Sharma, and D. Chakravarty, "Deepvo: a deep learning approach for monocular visual odometry", *In Proceedings of the 2017 British Machine Vision Conference*, BMVC 2017, London, U.K., Sep. 2017, pp. 1–11.

[92] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3D reconstruction", *In Proceedings of the 2006 International Conference on Computer Vision and Pattern Recognition*, ICPR 2006, Hong Kong, China, Aug. 2006, pp. 363–370.

[93] A. Mulloni, M. Ramachandran, G. Reitmayr, D. Wagner, R. Grasset, and S. Diaz, "User friendly SLAM initialization", *In Proceedings of the 12th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2013, Adelaide, Australia, Sep. 2013, pp. 153–162.

[94] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras", *IEEE Transactions on Robotics*, vol. 33(5), pp. 1255–1262, 2017.

[95] R. Mur-Artal, J. M. Martinez-Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular-SLAM system", *IEEE Transactions on Robotics*, vol. 31(5), pp. 1147–1163, 2015.

[96] R. A Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera", *In Proceedings of the 2010 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2010, San Francisco, CA, USA, Jun. 2010, pp. 1498–1505.

[97] R. A Newcombe, S. J Lovegrove, and A. J. Davison, "DTAM: Dense Tracking and Mapping in real-time", *In Proceedings of the 2011 IEEE International Conference on Computer Vision*, ICCV 2011, Barcelona, Spain, Nov. 2011, pp. 2320–2327.

[98] D. Nistér. "An efficient solution to the five-point relative pose problem", *IEEE transactions on pattern analysis and machine intelligence*, vol. 26(6), pp. 756–770, 2004.

[99] C. F. Olson, L. H. Matthies, J. R. Wright, R. Li, and K. Di, "Visual terrain mapping for mars exploration", *Computer Vision and Image Understanding*, vol. 105, pp. 73–85, 2007.

[100] G. Pascoe, W. Maddern, M. Tanner, P. Piniés, and P. Newman, "NID-SLAM: Robust monocular SLAM using Normalised Information Distance", *In Proceedings of the 2017 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2017, Honolulu, Hawaii, USA, Jul. 2017, pp. 1–8.

[101] M. Pérez-Patricio, A. Aguilar-González, M. Arias-Estrada, H. R. Hernandez-de Leon, J. L. Camas-Anzueto, and J.A. de Jesús Osuna-Coutiño, "An FPGA stereo matching unit based on fuzzy logic", *Microprocessors and Microsystems*, vol. 42, pp. 87–99, 2016.

[102] M. Persson, T. Piccini, M. Felsberg, and R. Mester, "Robust stereo visual odometry from monocular techniques", *In Proceedings of the 2015 IEEE Intelligent Vehicles Symposium*, Seoul, Korea, Jun. 2015, pp. 686–691.

[103] S. Pillai and J. Leonard, "Monocular slam supported object recognition", *In Proceedings of the 2016 British Machine Vision Conference*, BMVC 2016, York, U.K., Sep. 2016, pp. 1–10.

[104] S. Pillai and J. J. Leonard. "Towards visual ego-motion learning in robots", *In Proceedings of the 2017 IEEE International Conference on Intelligent Robots and Systems*, IROS 2017, Vancouver, BC, Canada, Sep. 2017, pp. 1–6.

[105] C. Pirchheim and G. Reitmayr, "Homography-based planar mapping and tracking for mobile phones", *In Proceedings of the 10th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2011, Basel, Switzerland, Oct. 2011, pp. 27–36.

[106] C. Pirchheim, D. Schmalstieg, and G. Reitmayr, "Handling pure camera rotation in keyframe-based SLAM", *In Proceedings of the 12th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2013, Adelaide, Australia, Oct. 2013, pp. 229–238.

[107] T. Pire, T. Fischer, G. Castro, P. De Cristóforis, J. Civera, and J. Jacobo-Berlles. "S-PTAM: Stereo Parallel Tracking and Mapping", *Robotics and Autonomous Systems (RAS)*, vol. 93, pp. 27–42, 2017.

[108] K. Pirker, M. Rüther, and H. Bischof, "Cd-SLAM: Continuous localization and mapping in a dynamic world", *In Proceedings of the 2015 IEEE International Conference on Intelligent Robots and Systems*, IROS 2011, San Francisco, CA, USA, Sep. 2011, pp. 3990–3997.

[109] A. Plyer, G. Le-Besnerais, and F. Champagnat, "Massively parallel lucas kanade optical flow for real-time video processing applications", *Journal of Real-Time Image Processing*, vol. 11(4), pp. 713–730, 2014.

[110] A. Pretto, E. Menegatti, and E. Pagello, "Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation", *In Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, ICRA 2011, Shanghai, China, May 2011, pp. 3289–3296.

[111] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, "PL-SLAM: Real-time monocular visual SLAM with points and lines", *In Proceedings of the 2017 IEEE International Conference on Robotics and Automation*, ICRA 2017, Singapore, Singapore, May 2017, pp. 4503–4508.

[112] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking", *In Proceedings of the 2005 IEEE International Conference on Computer Vision*, ICCV'05, Beijing, China, Oct. 2005, pp. 1508–1515.

[113] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF", *In Proceedings of the 2011 IEEE International Conference on Computer Vision*, ICCV 2011, Barcelona, Spain, Nov. 2011, pp. 2564–2571.

[114] S. Schubert, P. Neubert, and P. Protzel, "Towards camera based navigation in 3D maps by synthesizing depth images", *In Proceedings of the Anual Conference Towards Autonomous Robotic Systems*, TAROS 2017, Guildford, UK, Jul. 2017, pp. 601–616.

[115] T. Senst, J. Geistert, I. Keller, and T. Sikora, "Robust local optical flow estimation using bilinear equations for sparse motion estimation", *In Proceedings of the 20th IEEE International Conference on Image Processing*, ICIP 2013, Melbourne, Australia, Sep. 2013, pp. 1–8.

[116] J. Shi and C. Tomasi, "Good features to track", Technical report, Cornell University, 1993.

[117] G. Silveira, E. Malis, and P. Rives, "An efficient direct approach to visual SLAM", *IEEE transactions on robotics*, vol. 24(5), pp. 969–979, 2008.

[118] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer, "Discriminative learning of deep convolutional feature point descriptors", *In Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV 2015, Santiago, Chile, Sep. 2015, pp. 118–126.

[119] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics", *In Proceedings of the 1987 IEEE International Conference on Autonomous robot vehicles*, Raleigh, NC, USA, Mar. 1987, pp. 167–193.

[120] S. M Smith and J. M. Brady, "Susan—a new approach to low level image processing", *International Journal of Computer Vision*, vol. 23(1), pp. 45–78, 1997.

[121] S. Song and M. Chandraker, "Robust scale estimation in real-time monocular SfM for autonomous driving", *In Proceedings of the 2014 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2014, Columbus, OH, USA, Oct. 2014, pp. 24-27.

[122] S. Song, M. Chandraker, and C. C. Guest, "Parallel real-time monocular visual odometry", *In Proceedings of the 2013 IEEE International Conference on Robotics and Automation*, ICRA 2013, Karlsruhe, Germany, May 2013, pp. 1–6.

[123] C. Stachniss, "Robotic mapping and exploration", Springer-Verlag, 2009.

[124] J. Stowers, M. Hayes, and A. Bainbridge-Smith, "Altitude control of a quadrotor helicopter using depth map from microsoft kinect sensor", *In Proceedings of the 2011 IEEE International Conference on Mechatronics*, ICM 2011, Istanbul, Turkey, Apr. 2011, pp. 358–362.

[125] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular-SLAM", *Robotics: Science and Systems*, vol. 1(1), pp. 1–8, 2010.

[126] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems", *In Proceedings of the 2012 IEEE International Conference on Intelligent Robots and Systems*, IROS 2012, Vilamoura, Algarve, Portugal, Oct. 2012, pp. 573–580.

[127] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao, "Robust monocular-SLAM in dynamic environments", *In Proceedings of the 12th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR 2013, Adelaide, Australia, Sep. 2013, pp. 209–218.

[128] J. J. Tarrio and S. Pedre, "Realtime edge-based visual odometry for a monocular camera", *In Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV 2015, Santiago, Chile, Dec. 2015, pp. 702–710.

[129] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular-SLAM with learned depth prediction", *In Proceedings of the 2017 IEEE*

International Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, Hawaii, USA, Jul. 2017, pp. 1–8.

[130] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines", *In Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, ICRA 2003, Taipei, Taiwan, Sep. 2003, pp. 4270–4275.

[131] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, and G. Hoffmann "Stanley: The robot that won the darpa grand challenge", *Journal of field Robotics*, vol. 23(9), pp. 661–692, 2006.

[132] C. Tomasi and T. Kanade, "Detection and tracking of point features", Technical Report CMU-CS-91-132, 1991.

[133] P. H. S. Torr and A. Zisserman, "Mlesac: A new robust estimator with application to estimating image geometry", *Computer vision and image understanding*, vol. 78, pp. 138–156, 2000.

[134] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger. "Depth Prediction Evaluation", 2017. [Online]. Available: http://www.cvlibs.net/datasets/kitti/eval_depth.php?benchmark=depth_prediction, [Accessed: 10-Feb-2019].

[135] Y. Verdie, K. Yi, P. Fua, and V. Lepetit, "Tilde: A temporally invariant learned detector", *In Proceedings of the 2015 IEEE International Conference on Computer Vision*, ICCV 2015, Santiago, Chile, Dec. 2015, pp. 5279–5288.

[136] VICON. "Vantage: Cutting edge, flagship camera with intelligent feedback and resolution", 2018. [Online]. Available: https://www.vicon.com/products/camera-systems/vantage, [Accessed: 10-Feb-2019].

[137] G. Vogiatzis and C. Hernández, "Video-based, real-time multi-view stereo", *Image and Vision Computing*, vol. 29, pp. 434–441, 2011.

[138] Y. Wang, J. Fan, C. Qian, and L. Guo, "Ego-motion estimation using sparse SUFR flow in monocular vision systems", *International Journal of Advanced Robotic Systems*, vol. 13(6), pp. 1–9, 2016.

[139] M. Weber, C. Rist, and J M. Zöllner, "Learning temporal features with CNNs for monocular visual ego motion estimation", *In Proceedings of the 2017 IEEE International Conference on Intelligent Transportation Systems*, ITSC 2017, Yokojama, Japan, Oct. 2017, pp. 1–6.

[140] Z. Wei, D. J. Lee, and B. E. Nelson, "FPGA-based real-time optical flow algorithm design and implementation", *Journal of Multimedia*, vol. 2(5), pp. 38–45, 2007.

[141] L. Yang, F. Tan, A. Li, Z. Cui, Y. Furukawa, and P. Tan, "Polarimetric dense monocular-SLAM", *In Proceedings of the 2018 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2018, Salt Lake City, United States, 2018, pp. 3857–3866.

[142] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up SLAM: Semantic monocular plane SLAM for low-texture environments", *In Proceedings of the 2016 IEEE International Conference on Intelligent Robots and Systems*, IROS 2016, Daejeon, Korea, Oct. 2016, pp. 1222–1229.

[143] Z. Yang, P. Wang, W. Xu, L. Zhao, and R. Nevatia, "Unsupervised learning of geometry with edge-aware depth-normal consistency", *In Proceedings of the The Thirty-Second AAAI Conference on Artificial Intelligence*, AAAI-18, New Orleans, LA, USA. Feb. 2017, pp. 1222–1229.

[144] Z. Yang, P. Wang, Y. Wang, W. Xu, and R. Nevatia, "Lego: Learning edge with geometry all at once by watching videos", *In Proceedings of the 2018 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2018, Salt Lake City, United States, 2018, pp. 225–234.

[145] J. Zhang and S. Singh, "Visual-LiDAR odometry and mapping: Low-drift, robust, and fast", *In Proceedings of the 2015 IEEE International Conference on Robotics and Automation*, ICRA 2015, Seattle, WA, USA, May 2015, pp. 365–372.

[146] L. Zhang and R. Koch, "Hand-held monocular SLAM based on line segments", *In Proceedings of the 2011 Irish Machine Vision and Image Processing Conference*, IMVIP 2011, Dublin, U.K., Sep. 2011, pp. 7–14.

[147] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe,. "Unsupervised learning of depth and ego-motion from video", *In Proceedings of the 2017 IEEE International Confer-*

ence on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, Hawaii, USA, Jul. 2017, pp. 7–12.

[148] Y. Zou, Z. Luo, and J. B. Huang, Df-net: Unsupervised joint learning of depth and flow using cross-task consistency. *In Proceedings of the 2018 European Conference on Computer Vision*, ECCV 2018, Munich, Germany, Sep. 2018, pp. 38–55.

[149] S. Zweig and L. Wolf, "Interponet, a brain inspired neural network for optical flow dense interpolation", *In Proceedings of the 2017 IEEE International Conference on Computer Vision and Pattern Recognition*, CVPR 2017, Honolulu, Hawaii, USA, Jul. 2017, pp. 79–86. .