Robust Real-Time Visual Odometry for Autonomous Ground Vehicles

by

Mohamed D. Aladem

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering
(Computer Engineering)
in the University of Michigan-Dearborn
2017

Master's Thesis Committee:

Assistant Professor Samir Rawashdeh, Chair
Assistant Professor Stanley Baek
Assistant Professor Yu Zheng

To my beloved parents

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ATE Absolute Translation Error

CPU Central Processing Unit

DoF Degrees of Freedom

GPS Global Positioning System

GPU Graphics Processing Unit

IMU Inertial Measurement Unit

RANSAC Random Sample Consensus

RMSE Root Mean Square Error

RPE Relative Pose Error

PROSAC Progressive Sample Consensus

SIMD Single Instruction Multiple Data

VO Visual Odometry

V-SLAM Visual Simultaneous Localization And Mapping

XOR Exclusive Or

Abstract

Estimating the motion of an agent, such as a self-driving vehicle or mobile robot, is an essential requirement for many modern autonomy applications. Real-time and accurate position estimates are essential for navigation, perception and control, especially in previously unknown environments. Using cameras and *Visual Odometry* (VO) provides an effective way to achieve such motion estimation. Visual odometry is an active area of research in computer vision and mobile robotics communities, as the problem is still a challenging one. In this thesis, a robust real-time feature-based visual odometry algorithm will be presented. The algorithm utilizes a stereo camera which enables estimation in true scale and easy startup of the system.

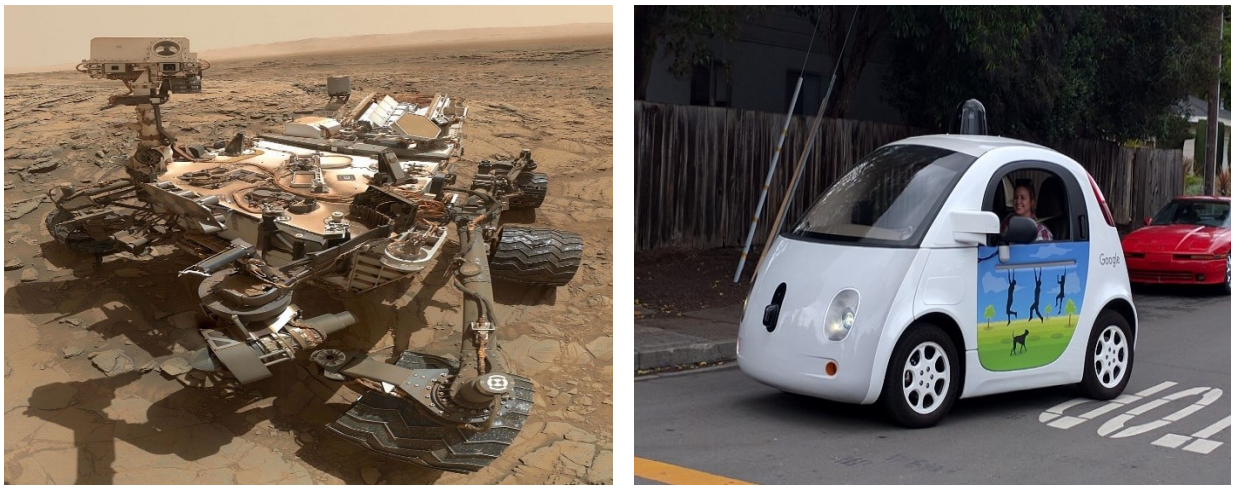
A distinguishing aspect of the developed algorithm is its utilization of a local map consisting of sparse 3D points for tracking and motion estimation. This results in the full history of each feature being utilized for motion estimation. Hence, drift in the ego-motion estimates are greatly reduced, enabling long-term operation over prolonged distances. Furthermore, the algorithm employs Progressive Sample Consensus (PROSAC) in order to increase robustness against outliers. Extensive evaluations on the challenging KITTI and New College datasets are presented. KITTI dataset was collected by a vehicle driving in the city of Karlsruhe in Germany, and represents one of the most commonly used datasets in evaluating self-driving algorithms. The New College dataset was collected by a mobile robot traversing within New College grounds in Oxford. Moreover, experiments on custom data are performed and results are presented.

Chapter 1

Introduction

In recent years, robots are being integrated more and more into our lives. We have been relying on robots to perform diverse tasks and applications. Autonomous mobile robots are used in industrial domain for transporting material around in warehouses. In military, autonomous mobile robots (be it ground or aerial) are very important tools. In unfortunate catastrophes, mobile robots can be used to rescue survivors, and to help in the cleanup operations in case of a nuclear plant meltdown for example. As for domestic applications, autonomous robotic vacuum cleaners have gained tremendous popularity and are found in many households; or they are simply used as high-tech toys.

Two interesting applications of mobile robotics are shown in Figure 1-1. Mars rovers are at the forefront of human quest to inhabit the red planet. They continuously help us in exploring the



(a) (b)

Figure 1-1: Two examples of autonomous mobile robotics. (a) NASA's Curiosity Mars rover. (b) Google's self-driving car[1].

[1] By Grendelkhan - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=47467048

planet and collecting scientific data. Furthermore, self-driving cars are becoming a reality, and research done in mobile robotics and related fields is enabling the autonomous cars of the future. A key aspect is that in most applications, the robot will be moving autonomously and interacting with the environment.

Being able to estimate its position in real-time is a fundamental task of mobile robots. This position estimation enables other important tasks such as planning, mapping and control. Low-latency, local position estimates are important for stable high-speed control; while low-rate, global position estimates are essential to relate prior map data [1]. It is clear that accurate estimation of motion is an essential requirement enabling autonomy for mobile robots and modern automotive applications.

Estimating the mobile robot's position robustly and accurately for long-term operation is a difficult problem, and it is an active area of research. Dead-reckoning approach based on a combination of Inertial Measurement Units (IMUs) and wheel odometers might be acceptable for short-term tasks. However, continuous integration of noisy measurements from such sensors results in accumulation of error and permanent drift in estimated position, which makes them unsuitable for long-term sustained operation. Moreover, wheel odometry suffers from wheel slippage on uneven terrains or slippery surfaces, such as those facing Mars rovers [2].

The Global Positioning System (GPS) is widely available and is capable of providing geolocation to the receiver. This absolute location measurement can be used to observe the drift in dead-reckoned position estimate and hence correct it properly. However, GPS can only provide said location information when it has unobstructed line-of-sight to at least four GPS satellites. This means GPS sensors lose satellite lock in tunnels and urban canyons. Other commonly used sensors include: Light Detection and Ranging (LIDAR) which fail in rain, Ultra-sonic sensors which have a short range, and Radar which has low angular resolution and may not detect less massive objects well, such as pedestrians or animals. Each sensor has its advantages and its share of downsides, which makes it necessary to employ a suite of sensors when designing navigation systems to overcome the limitations of each sensor alone.

Recent advances in image processing hardware and software are enabling the deployment of cameras as real-time embedded systems in autonomous mobile robots and for automotive applications, being a central component enabling self-driving capability. Cameras are attractive

sensor to perform motion estimation because of their affordability and the richness of information that can be gathered. Furthermore, since cameras are passive sensors, they do not interfere with each other when multiple are deployed.

Visual Odometry (VO) is the process of estimating the position and orientation of an agent (e.g., a vehicle) using only the input of a single or multiple cameras attached to it [3]. Vision-based motion estimation is capable of providing accurate estimates in conditions where dead-reckoning or GPS fail. For example, visual odometry was used successfully in NASA's Mars Exploration Rovers [2]. The problem of estimating vehicle motion from visual input was first approached by Moravec [4] in the early 1980s. Moravec established the first motion-estimation pipeline whose main functional blocks are still used today. However, the term visual odometry was first coined by Nister et al. in their landmark paper [5]. Their paper was first to demonstrate a real-time long-run implementation with a robust outlier rejection scheme. Many different methods have been developed over the years.

Visual odometry approaches are classified as either *monocular* VO when one camera is used, or *stereo* VO when more than one is used. Stereoscopic camera systems are necessary to obtain depth information. In order to get more intuition into the process, the image of an object is formed by light rays bouncing off the object and falling onto the camera sensor plane. If we backtracked a ray from the camera sensor, we are certain that the object lies somewhere along the ray but we cannot tell at what point exactly. On the other hand, if we backtracked two such rays from two different cameras imaging the same object, we can find the object at the intersection of the two rays. Figure 1-2 illustrates the difference between monocular and stereoscopic vision.

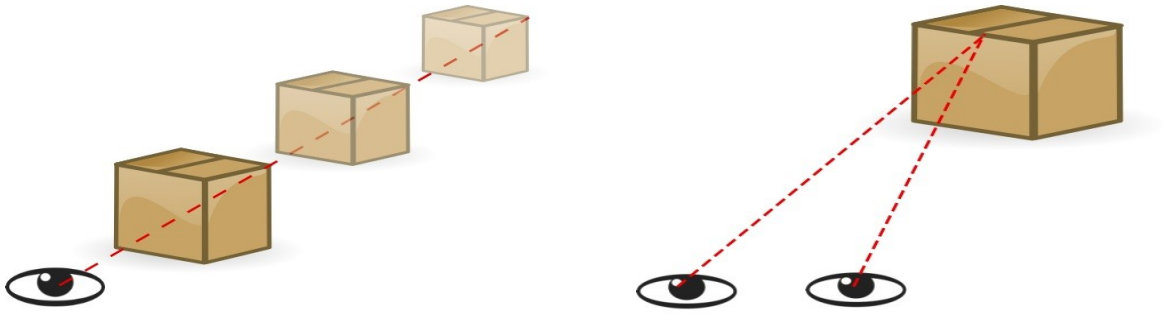


Figure 1-2: [left] Monocular vision where observing an object from one point is not enough to identify its exact position in space. [right] Stereoscopic vision where tracking rays intersect at the object in space.

By relying on only a single camera for its operation, monocular VO is inherently a more difficult problem. As was illustrated, one of the major issues in monocular VO is the scale ambiguity, which means motion trajectory can only be estimated with an ambiguous scale factor. On the other hand, with a known baseline distance between cameras, stereo VO can estimate the exact motion trajectory, i.e., it is able to recover global metric scale, and thus do not suffer from the scale ambiguity problem [6]. Another issue with monocular VO is that they require careful and usually involve tricky initialization procedures, whereas in stereo VO it is easier to achieve a *power-and-go* system. The main downside of stereo VO is that they degenerate into monocular VO for scenes with very distant objects relative to the stereo camera baseline.

As was previously mentioned, each sensor has its downsides and cameras are no exception. Cameras usually require a large amount of processing power and suffer from over exposure when the sun is shining head-on or is reflected into the camera lens. Therefore, visual odometry positioning accuracy can be improved by performing sensor-fusion with other positioning or motion sensors such as GPS [6], or Inertial Measurement Units (IMUs) [7].

1.1 Contributions

Visual odometry and vision-based motion estimation methods are indeed attractive solutions. Although the basic steps for visual motion estimation might seem intuitive and simple, implementing such a system is difficult. This is because of the large space of trade-offs and different potential heuristics or techniques employed. Such trade-offs are present in each stage of the visual motion-estimation pipeline: the type of feature to be used, how those features are described and the strategy to find correspondences among them, the method used to compute the camera motion, how outliers are being handled in the process, among other considerations. Other factors include the operational environment, frame rates and memory consumption. All such factors impact the performance of the system.

In this thesis, a feature-based stereo visual odometry algorithm is presented. The algorithm is targeted for long-term operation over extended distances of ground vehicles. The main features of the developed algorithm are:

1- It employs a point-based local map for its operation.

2- It employs an outlier rejection scheme based on PROgressive SAmple Consensus (PROSAC) [8].
3- A stereo camera is used thus achieving the benefits of a stereo visual odometry system.

The first point, using a local map for operation, is a distinguishing aspect of the developed algorithm. Traditionally, VO algorithms relied on tracking features between consecutive frames, i.e., from frame-to-frame. However, using a point-based local map means the full history of each feature is used for motion estimation. This greatly helps in reducing drift in the estimated path and makes it more suitable for long-distance operation. Only recently the full-feature history started being used in VO [9].

The algorithm's accuracy and performance will be extensively evaluated using the challenging KITTI vision benchmark dataset [10] which is widely used in evaluating autonomous driving algorithms, and the New College dataset [11] which was collected by a mobile robot traversing within New College grounds in Oxford. Custom experiments on manually collected data will also be performed and demonstrated.

Chapter 2

Background

The end goal of visual odometry is estimating the vehicle motion trajectory while it is exploring an unknown environment. In this chapter, some essential background concepts required for solving the visual motion estimation problem are presented.

2.1 Image formation

An essential first step is understanding the geometry of the camera and how images are formed. Among the most well know references on visual geometry is the work of Hartley and Zisserman [6] where more complete mathematical treatment of the subject at hand can be found.

The *pinhole camera model* is the dominating model of image formation in computer vision and is the most basic imaging device [6]. In the pinhole camera, rays of light pass through a small hole, or *aperture*, in one side of a dark chamber and strike an imaging plane where the image is formed or recorded. The aperture is assumed infinitesimally small for the pinhole camera model. Figure 2-1 illustrated the pinhole camera model. A small aperture is necessary to form a sharp image. Note how the recorded image appears to be flipped relative the real scene as a result of the light rays passing through the camera aperture.

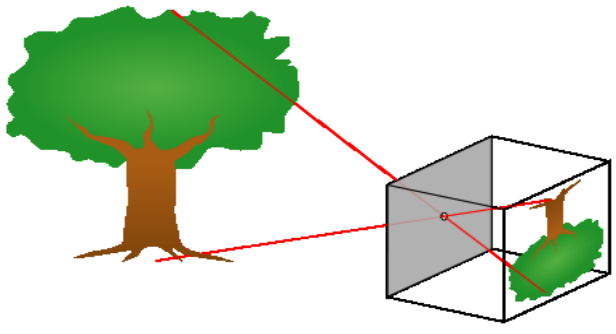


Figure 2-1: Illustration of the ideal pinhole camera model. As rays of light reflected from the real object pass through the camera aperture, the image is therefore captured flipped.

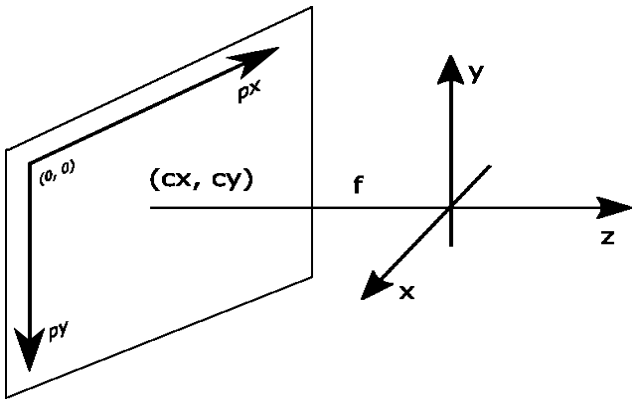


Figure 2-2: A simplified illustration of camera intrinsics showing the focal length (f) and the principal point (cx, xy).

Geometrically speaking, the pinhole is also called the *center of projection* where the real object is projected onto the imaging device plane through it. It is common to represent the imaging sensor plane *in front of* the projection point; that is, swap the pinhole and the image plane. This way, the object appears right up and not flipped. The distance between the center of projection and the imaging sensor is the *focal length*, commonly abbreviated as f. Refer to Figure 2-2 for a simplified illustration of the geometry involved.

The projection of objects onto the imaging sensor is a linear mapping between 3D points and image points in homogeneous co-ordinates. The process can be described in terms of: 3D point $X = [x, y, z]^T$, camera parameters, the focal lengths (f_x, f_y) and the principal point (c_x, c_y). Projecting through the pinhole results in image point (p_x, p_y) in pixels. The governing projection equations are:

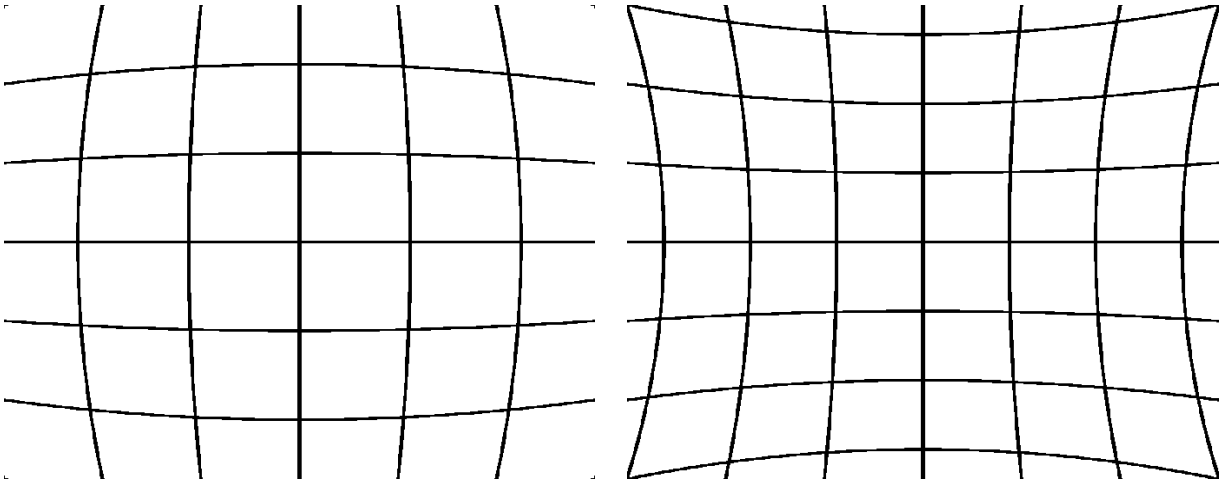
$$p_x = f_x \frac{x}{z} + c_x \tag{2.1}$$

$$p_y = f_y \frac{y}{z} + c_y \tag{2.2}$$

The geometry of the imaging process is usually encoded in the *camera intrinsics matrix* $\boldsymbol{K}$,

$$\boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Real-life imaging devices differ from the ideal pinhole camera in that they employ a lens at the pinhole. Such lenses are placed for different goals, they are used to allow sufficient amount of light in, zooming in and out by changing the focal length, changing the sharpness of the image by controlling the aperture, or to introduce specific visual effects [13]. The use of lenses however can result in *distortion* in the recorded image, where the image deviates from and cannot be described by the pinhole camera model alone. This image distortion can take a variety of shapes depending on the lens material and manufacturing quality. The most commonly encountered form of distortion is *radial distortion* which can be classified as either *barrel* or *pincushion* distortion; as shown in Figure 2-3.



(a) (b)

Figure 2-3: Illustration of the two common radial distortions: (a) Barrel distortion, and (b) Pincushion distortion.

It is important to account for the effect of such distortions in order to get accurate and meaningful results for computer vision tasks. Radial distortion can be described as a function of the distance to the center of distortion [13]:

$$\begin{pmatrix} u_d \\ v_d \end{pmatrix} = L\left(\sqrt{u_l^2 + v_l^2}\right)\begin{pmatrix} u_l \\ v_l \end{pmatrix} \tag{2.4}$$

where (u_d, v_d) are the distorted image pixels, and (u_l, v_l) are the linearized image pixels. $L(r)$ is the distortion function and depends on the distance r to the center of distortion. It is usually approximated by a Taylor series of order n:

$$L(r) \approx 1 + k_1 r + k_2 r^2 + k_3 r^3 + \cdots + k_n r^n \tag{2.5}$$

The intrinsics matrix **K** along with the parameters of the distortion polynomial are collectively called the *intrinsic parameters* of the camera. The rigid transformation (rotation and translation) from the world 3D coordinates to the camera 3D coordinates is referred to as the *extrinsic parameters* of the camera. These intrinsic and extrinsic camera parameters can be simultaneously estimated in a process commonly known as *camera calibration* [14].

2.2 Stereo imaging

After introducing the ideal pinhole camera model and basic image formation process, we are in position to proceed to address *stereo imaging*. It was mentioned in the first chapter how a stereoscopic vision approach is necessary to properly pinpoint objects in 3D space, that is, estimate the object *depth* (refer to Figure 1-2). This stereo imaging capability is built into our eyes and help us perceive depth. Computer *stereo vision* is an emulation of this depth perception process. It relies on *stereo cameras* to achieve this, where a stereo camera consists of two identical pinhole cameras with a fixed distance between them called the *baseline*. Figure 2-4 shows an example of a stereo camera.

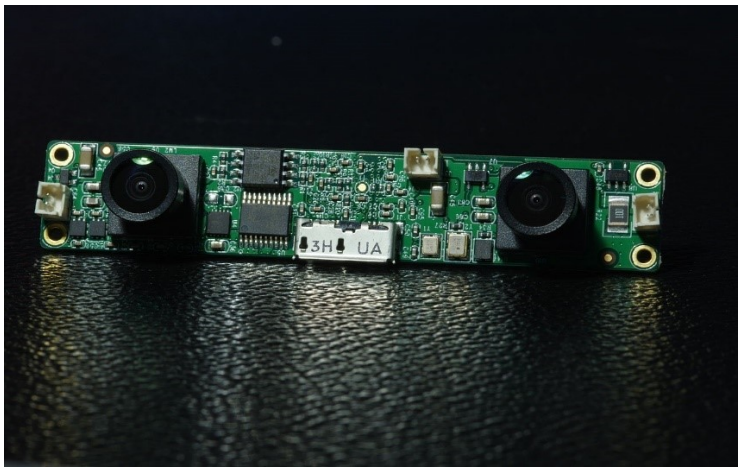
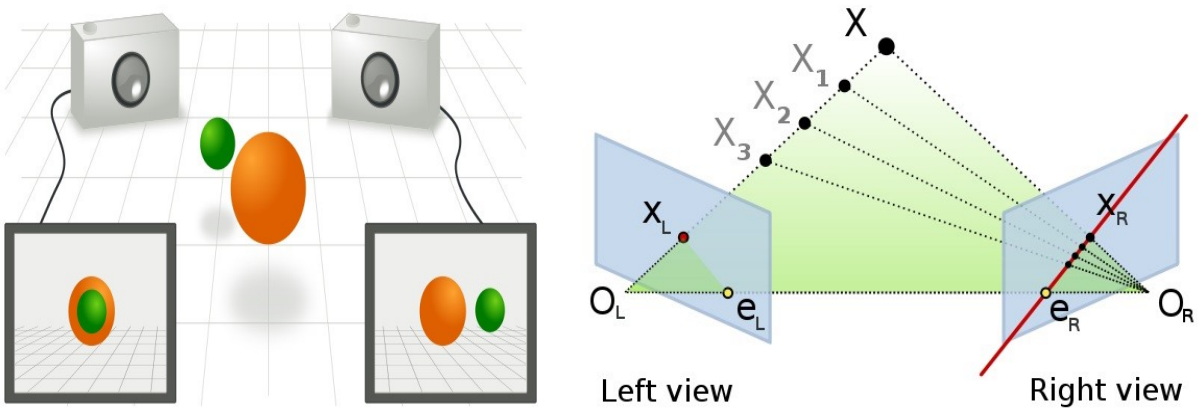


Figure 2-4: A stereo camera example. Note how it consists of two identical pinhole cameras separated by a fixed distance, or baseline.[2]

Epipolar geometry is the geometry of stereo vision. When a scene is imaged by two cameras from different perspectives, epipolar geometry can be used to describe the relations between the result views. Figure 2-5 illustrates this concept and the epipolar geometry itself.

[2] By Timothée Cognard - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=33843774



(a) (b)

Figure 2-5: Epipolar geometry illustrations. (a) Epipolar geometry describe the relations between two images of the scene from two different point of views [3]. (b) The epipolar geometry [4].

In Figure 2-5 (b), O_L and O_R represent the two centers of projection of the left and right cameras respectively of the stereo camera. The two cameras are looking at point of interest **X**. Points $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ represent the projections of point **X** onto the image planes of the left and right cameras. Each camera's optical center (O_L and O_R) projects into a different point in the other camera's image plane. These two image points, $\boldsymbol{e}_L$ and $\boldsymbol{e}_R$, are called *epipoles* or *epipolar points*. The plane formed by **X**, O_L and O_R is called the *epipolar plane*. It intersects the two image planes at the *epipolar lines*. That is, the line ($\boldsymbol{e}_R$, $\boldsymbol{x}_R$) in the right camera is an epipolar line. This line is also the projection of the 3D line (**X**, O_L) onto the right camera's image plane. Similar symmetric argument can be made for the left camera.

The important observation is, assuming the relative position of the two cameras is known, if the projected point $\boldsymbol{x}_L$ is known, then the epipolar line ($\boldsymbol{e}_R$, $\boldsymbol{x}_R$) is known, and the point **X** projection on the right image plane ($\boldsymbol{x}_R$) *must* lie on this specific epipolar line. This is the *epipolar constraint*. Observe how all points that lies on the 3D line **X**, O_L (e.g., $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, …) will verify this constraint. It means that it is possible to test if two points correspond to the same 3D point. Similarly, if the two projection points $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$ are known and they correspond to the same point **X**, then the projection lines must intersect at **X**. This process of calculating 3D point **X** from the two image points is called *triangulation*, an important concept that forms the basis of

[3] By Arne Nordmann (norro) - Own illustration, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3550774
[4] By Arne Nordmann (norro) - Own work (Own drawing), CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1702052

3D reconstruction. By utilizing the epipolar constraint, the *stereo correspondence* problem – finding which parts of an image corresponds to in the other one from the stereo camera – is simplified as any point in one view must lie on the epipolar line in the other one.

From Figure 2-5 (b), notice how if the two image planes of the left and right cameras were coplanar, then the epipolar lines would be horizontal (assuming it is a horizontal stereo camera). In this case, finding stereo correspondences will be further simplified since the search for any point from one of the image pair would be restricted to the same horizontal pixel row in the other image. As it is difficult in practice to build a perfect stereo camera with coplanar image planes, a transformation process called *stereo rectification* is usually performed on the recorded stereo images to achieve the same desired effect. The effect of stereo rectification can be seen in Figure 2-6 where looking at the marked lamp post, for example, you can see that in the unrectified case there is an offset between the images from the left and right cameras, whereas it lies on the same pixel row in the rectified one.

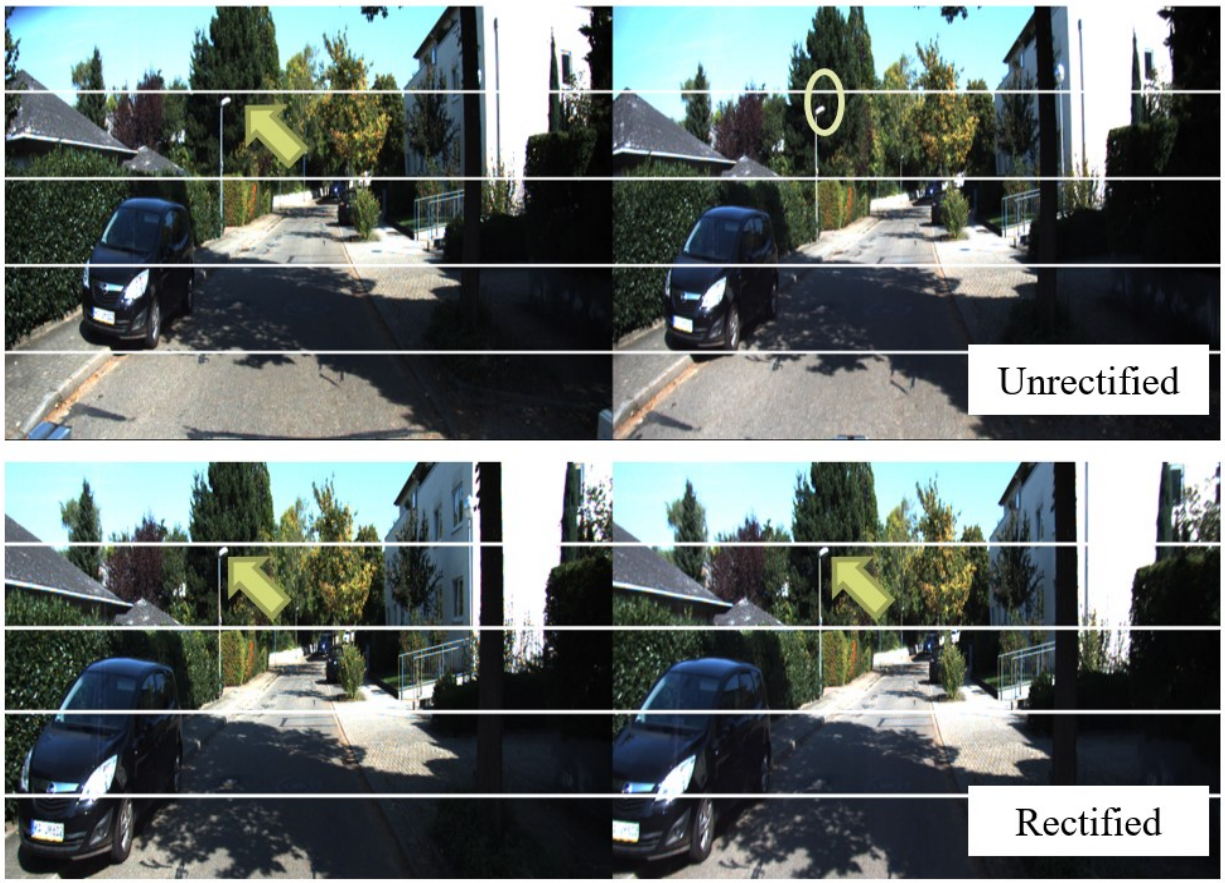


Figure 2-6: Illustration of the effect of stereo rectification on a sample image from the KITTI data set.

2.3 Coordinate frames and transformations

It is important to clearly define the coordinate frames used to ensure correctness and consistency. In this thesis, we define two coordinate frames: *world coordinate frame* (**F**w) and *camera coordinate frame* (**F**c). The camera coordinate frame is the one attached to the stereo camera, it is assumed to be originated at the optical center of the left camera of the stereo rig. The world coordinate frame is a universal frame where all elements of interest are referenced with respect to it. Setting up the world frame will be discussed more in the upcoming chapter. Both coordinates frames are assumed to be *right-handed* and follow *north-east-down* (NED) convention. That is, z-axis is pointing north, x-axis is pointing east, and y-axis is pointing down. These coordinate frames are illustrated in Figure 2-7.

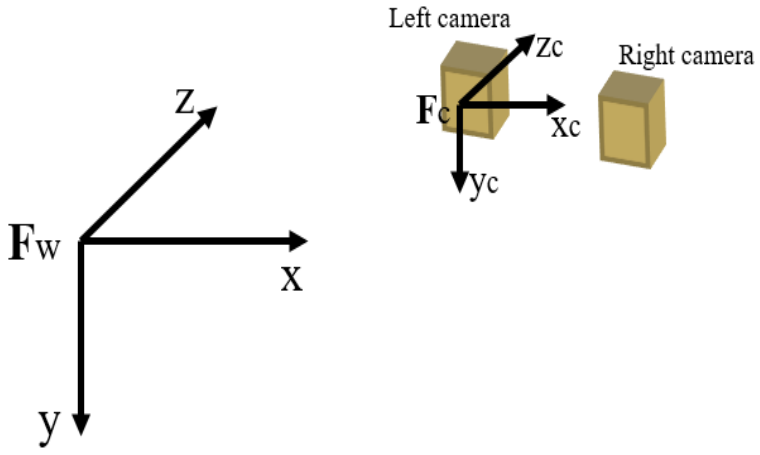


Figure 2-7: Illustrated of the two coordinate frames used, Fw is the world frame and Fc is the camera frame.

All transformations between coordinate frames are assumed to be *rigid* transformation and in all six degrees of freedom (6-DoF), i.e., it consists of only rotation and translation.

$$C = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{2.6}$$

where $C \in SE(3)$, the Lie group of rigid transformations in 3D space. $R \in SO(3)$, the group of 3D rotations. And $t \in \mathbb{R}^3$, is the translation 3D vector.

Camera *pose* is defined to be the *position* and *orientation* of the camera frame with respect to the world frame, or put another way, it is the *rotation* and *translation* that transforms the world frame to coincide with the camera frame.

There exist different possibilities for the parametrization of the 3D rotation:

Euler angles: In this parametrization, a rotation matrix is formed as the product of three rotations around three cardinal axes, e.g., x, y and z. Although it might be more intuitive to think about rotations in terms of Euler angles, they are not generally used much in practice as they suffer from some serious shortcomings. For instance, the resultant rotation depends on the order in which transforms were applied. Furthermore, they suffer from a problem called *gimbal lock*, were the 3D rotation loses one degree of freedom.

Axis/angle: In this representation, the rotation is parametrized by a unit vector representing an axis, and angle representing the rotation about that axis. This representation of rotation is minimal where the 3D rotation can be represented by three number only, and it is an excellent choice when dealing with small rotations (e.g., corrections to rotations) [15]. However, this representation is not unique as we can always add 360 degrees to the angle and get the same rotation matrix. Negating both the axis and the angle will result in the same rotation.

Unit quaternions: This representation is closely related to the axis/angle one. A unit quaternion is a unit length 4-vector written as $\boldsymbol{q} = (x, y, z, w)$. Unit quaternions live on the unit sphere $\|\boldsymbol{q}\| = 1$ and opposite sign quaternions (**q** and -**q**) represent the same rotation. Other than this dual covering ambiguity, the unit quaternion representation of rotation is unique. Furthermore, the representation is *continuous*, i.e., as rotation matrices vary continuously, a continuous unit quaternion representation can be found. Unit quaternions are an attractive representation and one of the most commonly used.

2.4 Image features

It was shown when discussing image formation process and the pinhole camera model that the imaging sensor captures a large array of pixels, or intensity samples. The total number of pixels typically reach hundreds of thousands. In order to cope with such a large amount of data, *image features* could be extracted and used for the computer vision tasks at hand, hence reducing the image array to a smaller discrete set of image features of interest. The detected feature type

could be corners (points), lines, or uniform regions (blobs). An example of these different types of features is show in Figure 2-8.

Figure 2-8: Common feature types: [TOP] Points or corners, [MIDDLE] Lines, [BOTTOM] Blob or region features.

After detecting features (e.g., corners), the next question is how would you know how similar two features are? Assuming corner features, one solution would be to extract image patches around the corners and compare their similarity by computing the pixel to pixel similarity using the Euclidian distance [16], but such measure is very sensitive to noise, rotation, translation and illumination changes. The answer to this dilemma is *feature descriptor*. A feature descriptor extractor is some function that is applied to the image patch around the feature in order to describe it in a way that makes it *invariant* to different image changes of interest applicable to our application.

Different invariant properties commonly described include [1]: Lighting, Translation (invariant to image features positions), Scale (invariant to size changes of image features), Rotation (invariant to in-plane rotation of the image), and Affine (invariant to combination of translation, rotation and scaling). Note that affine invariance in practice is used to increase robustness to out-of-plane rotations. The descriptor distance measure is designed with the design of the descriptor. This distance measure can be then used to compute how similar two descriptors describing two image patches are.

Examples of corners or points detectors are Harris [17] and FAST [18]. Those are detectors only with no descriptors. Examples of descriptors only include BRIEF [19] and FREAK [20]. An example of a corner detector with a descriptor combined is ORB [21]. An example of a detector for line segments is LSD line segment detector by Rafael et al. [22]. An example of a blob or region detector is MSER [23], while for a blob or region detector with a descriptor is SIFT [24] and SURF [25].

Point or corner features usually are the fastest to detect and require the least amount of computation, that is why they are more common in real-time high frame rate applications such as visual odometry. On the other hand, blob features enjoy the greatest level of invariance but they are slower to compute. Blob features are usually used for visual search, object recognition, place recognition and similar applications. Other than the required computation and the memory footprint of features, one important performance metric is *repeatability*. Repeatability can be defined as the degree to which features detected in a reference image are detected in other images of the same content [1]. In this thesis, the term *feature* will refer to detected points or corner-like geometric primitives in the image. Features will be associated with *descriptors*. The area in the image surrounding a feature is a pixel *patch*.

2.5 A word about V-SLAM

Related to the visual odometry problem at hand is the *Simultaneous Localization And Mapping* (SLAM) problem. *Visual*-SLAM, or V-SLAM, are SLAM algorithms where visual (camera) is used as the primary sensor. The SLAM problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map [26].

V-SLAM was solved initially using filtering where every frame is processed by a filter to simultaneously estimate the map feature locations and the camera pose. An example is MonoSLAM by Davison et al. [27]. The other dominant methodology to solve the V-SLAM problem is *keyframe* methods where they perform the global bundle adjustment to a selected keyframes. A representative keyframe system is PTAM by Klein and Murray [28]. PTAM uses a monocular camera and is targeted for augmented-reality applications in real-time. PTAM proposed computing the camera pose and performing the mapping operation of the environment in parallel, thus reducing what used to be an expensive batch operation into a real-time one. A rigorous analysis and comparison of both visual SLAM methods can be found in Ref. [29].

SLAM algorithms goal is not just to localize or estimate motion trajectory, but to build a global map of the environment. The algorithm will be continuously expanding this map while exploring new areas, and optimizing explored portions of the map based on new measurements. One technique commonly implemented for optimizing this global map is by detecting loop-closures, i.e., detecting when the camera observed a previously visited location. This helps in greatly reducing drift in the ego-motion estimation. Although SLAM methods are more capable than visual odometry, they are more complex and computationally demanding. Furthermore, for outdoors long-term operation over long-distances, mapping cost might become prohibitive, in terms of computation and memory requirements; added the fact that one cannot rely on the vehicle to return to a previously visited location thus performing loop-closing. As with any engineering problem, a tradeoff has to be made depending on the application between visual odometry and V-SLAM.

2.6 Feature-based versus direct methods

Approaches to visual odometry and V-SLAM can be classified as either *feature-based* or *dense/direct* methods. Feature-based methods try to extract distinctive interest points (features) and then discard the image and just use the image patches around the features for the tasks at hand (e.g., tracking, mapping, or loop-closure). On the other hand, direct methods operate on the whole image directly. That is, camera is localized by optimizing directly over image pixel intensities. By operating on the whole camera directly, direct methods are able to generate dense or semi-dense reconstructions of the environment. These reconstructions can be useful for other tasks than just localization.

Direct methods are generally more robust to blur and low-textured environments. However, direct methods are computationally demanding and became feasible to perform in real-time only with the wide spread use of Graphical Processing Units (GPUs). A key representative of feature-based visual SLAM systems is ORB-SLAM [30], whereas of the direct visual SLAM systems is LSD-SLAM [31].

2.7 A word about RANSAC

The problems that computer vision is trying to solve are generally *inverse* problems. This means that we have obtained some measurements or observations and our goal is to find the model and factors that caused them. The problem here is that the measurements will usually be contaminated with *outliers* – erroneous measurements. Computer vision algorithms need to be robust against such outliers.

One of the most notable algorithms to achieve such robustness is *RANsom SAmple Consensus* (RANSAC). RANSAC was initially developed Fischler and Bolles [32] in 1981. The algorithm is widely used due to its simplicity and effectiveness. After its initial publication, many different variations and enhancements have been developed over the years. The basic steps can be summarized as follows:

1. Select randomly a subset of the data.
2. Fit a model to the selected subset.
3. For all the other data, test if they agree with the model according to a loss-function. Data points that do agree with the model will constitute the consensus set.
4. If the consensus set is larger than a specified threshold, the model is declared good enough.

A good example to show the power of RANSAC is in a simple 2D line fitting problem shown in Figure 2-9Figure 2-9: An illustration showing a line fitting problem for data contaminated with outliers and RANSAC is able to fit a line to the inlier set. In this problem, the data is contaminated with outliers and a least-squares based solution will try to fit all the data points optimally including the outliers which will generally result in a bad line fit. On the other hand, RANSAC will be able to produce a line fit with a higher probability of fitting the inliers.

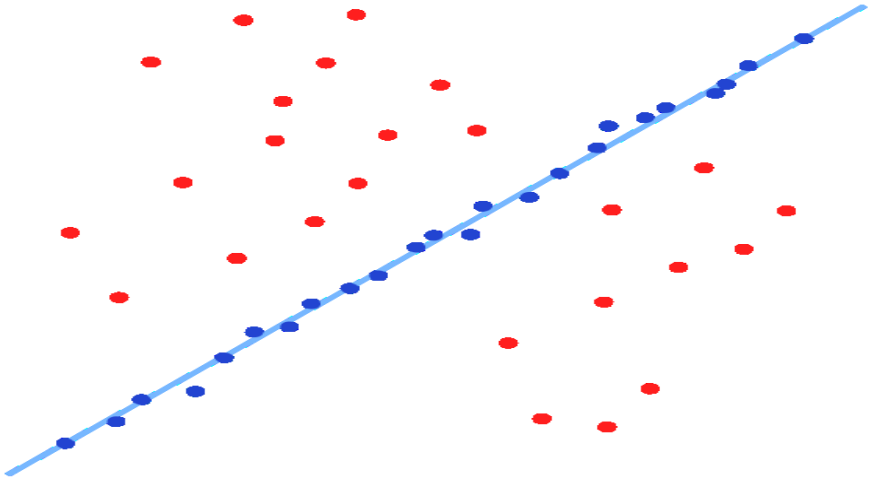


Figure 2-9: An illustration showing a line fitting problem for data contaminated with outliers and RANSAC is able to fit a line to the inlier set[5].

[5] By Msm - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2071406

Chapter 3

Visual Odometry Algorithm

In this chapter, the developed visual odometry algorithm is illustrated. First, the algorithm is introduced from a higher level perspective, then the details of each constituent component is illustrated in depth.

3.1 Algorithm overview

In this section, a generic high-level overview of the algorithm is presented. As mentioned previously, the introduced algorithm is feature-based using corner or point features, and it uses a stereo camera for its operation. One distinguishing aspect of the algorithm is that it employs a point-based local map. This local map consists of a sparse set of 3D points. It is important to realize that this local map is *transient* and used internally by the algorithm for its operation, and it is *not* an attempt to generate a global map of the environment.

The basic steps of the main loop of the algorithm are shown in Figure 3-1. The steps are summarized below:

1. The algorithm starts by retrieving the images from the stereo camera. The following preprocessing operations are needed:
    a. The images are converted into grayscale if they are in color. This is necessary for the feature extractor to work.
    b. The retrieved images are stereo rectified. Stereo rectification will morph epipolar lines to be horizontal which will allow for searching for feature correspondences between the two images along the same pixel row, as was illustrated in the previous chapter.
2. In the first frame, the system is not initialized yet and the local map is empty. Thus, the following initialization procedures will be performed:

a. Extract and match features between the left and right images of the stereo camera.
b. Use the found feature correspondences to triangle a bunch of 3D points.
c. Add the triangulated 3D points to the local map.
d. Set the current pose as the world coordinate frame. All poses will be reported with respect to this world coordinate frame. This implies, the pose reported for this first frame will be the identity pose.

3. For all subsequent frames:
    a. Extract features from the image pair.
    b. Predict the pose using a motion model.
    c. Track local map by:
        i. Projecting each 3D point from the map onto the left camera image plane.
        ii. For each projected point, look for the best matching extracted feature (from step a).
    d. Use the successful tracked points for estimating the pose.

The details will be illustrated in depth in the upcoming sections.

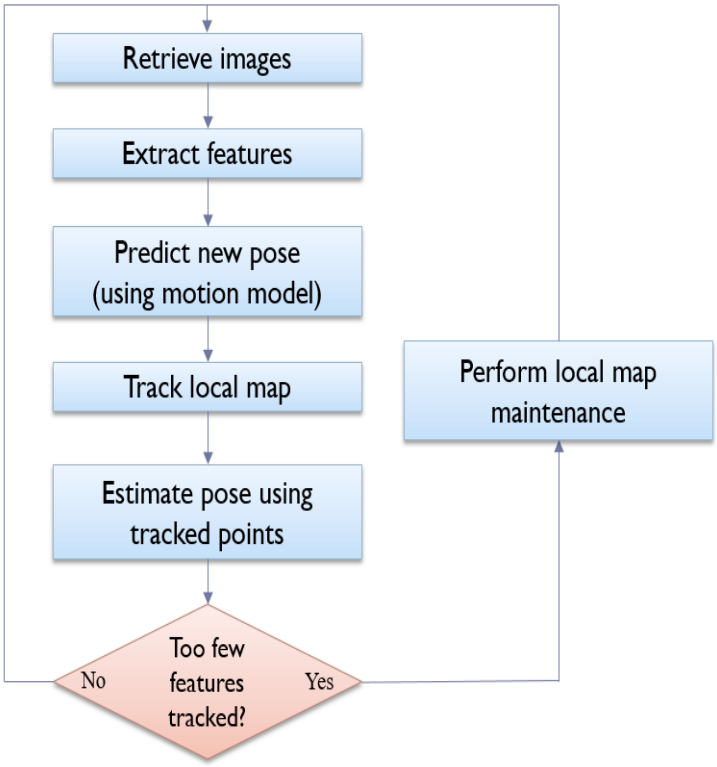


Figure 3-1: Basic steps of the main loop of the VO algorithm.

3.2 Feature extraction

As a feature-based method, the most important decision to be made is the type of feature to be used. The type of feature and its detection strategy will affect the overall performance of the system, as this feature will be used for finding correspondences between left and right images of the stereo camera for the purpose of triangulation of new 3D points. This features will also be used for tracking the local map and thus finally performing the motion estimation. In this algorithm, the chosen feature type is a point or corner-like features, as they are fast to compute and the availability of good corner detection methods.

In this algorithm, *Adaptive and Generic Accelerated Segment Test* (AGAST) [33] corner detector is used. AGAST is based on the *Accelerated Segment Test* (AST) developed by Edward Rosten and Tom Drummond in their *Features from Accelerated Segment Test* (FAST) corner detector [18]. Upon its introduction, FAST outperformed conventional corner detectors in terms of both performance and repeatability, which made it a common choice for real-time tasks in computer vision. The basic steps of FAST corner detector can be summarized as:

1. Select a pixel $\boldsymbol{p}$ in the image which is to be tested if it is a corner or not. Let its intensity be $\boldsymbol{I_p}$.
2. Choose an appropriate threshold value $\boldsymbol{t}$.
3. Consider a circle of 16 pixels around the pixel under test, as shown in Figure 3-2.
4. The pixel $\boldsymbol{p}$ is a corner if there exists a set of $\boldsymbol{n}$ contiguous pixels in the 16-pixel circle, and all of them are either brighter than $\boldsymbol{I_p + t}$, or darker than $\boldsymbol{I_p - t}$.

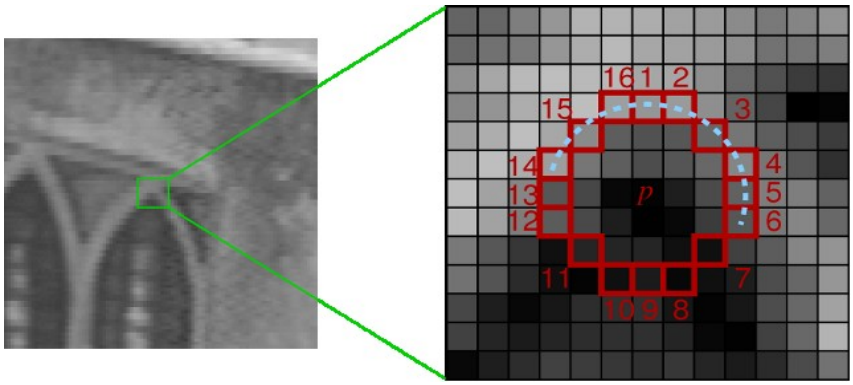


Figure 3-2: A candidate corner at a pixel and the 16-pixel radius circle around it[6].

[6] Image was taken as-is from Dr.Rosten's website at: www.edwardrosten.com/work/fast.html

In the accelerated segment test, $\boldsymbol{n}$ is chosen to be twelve because it enables a high-speed test to exclude a large number of non-corners: first examine only the four pixels at 1, 5, 9, 13 (four compass directions), if $\boldsymbol{p}$ is a corner then three of those pixels must satisfy the corner condition mentioned in item 4 above. Otherwise, it cannot be a corner. If this test passes, then we can proceed to test the full 16 pixels in the circle. Despite high performance, this approach still suffers from several weaknesses including that the high-speed test does not generalize well for $\boldsymbol{n}$ less than twelve, among others. The proposed solution to such weaknesses was to use a machine learning approach where the detector is trained from a selected set of images, preferably from the target application domain in order to compute a decision tree. The full details can be found in the original paper [18].

This approach still has some shortcomings[7]:

- If the camera is rotated, the pixel configuration of a corner may change significantly.
- Some corner configurations may be missing in the training set which leads to false positive and false negative responses of the corner detector.
- The decision tree has to be learned for every new environment from scratch.
- FAST builds a ternary decision tree for a binary target machine - a binary tree would be more efficient.
- ID3 (decision tree classifier) is used to build the decision tree, which is a greedy algorithm and, therefore, the result can be quite suboptimal.

AGAST proposes a technique to compute a binary decision tree (corner detector) which is generic and does not have to be adapted to new environments. By combining two trees, the corner detector adapts to the environment automatically. Hence, it results in a corner detector, which is faster and does not have to be trained while preserving the same corner response and repeatability as the complete FAST corner detector. Refer to the paper for full details [33].

One final important note about feature extraction, in this VO algorithm, *no* scale image pyramid of the target image is built. Corners are extracted and used from the full-size images directly.

[7] http://www6.in.tum.de/Main/ResearchAgast

3.3 Feature distribution

An important problem to take into account is the distribution of the detected features across the image. Poor distribution where detected features are concentrated in one region of the image can lead to poor results. Although both FAST and AGAST corner detectors have built-in means of non-maximal suppression to detect adjacent corners, the resulting corner distribution can still be improved upon.

In this work, in order to overcome this problem, a technique known as *adaptive non-maximal suppression* (ANMS) as described by Brown et al. is implemented [34]. Adaptive non-maximal suppression aims to limit the maximum number of features extracted from the image while at the same time ensuring good distribution across the image. Features are suppressed based on corner strength and only ones that are local maxima in the neighborhood are retained. Figure 3-3 illustrates the effect of applying ANMS. From the figure, it is clear how even in a challenging situation where foliage is dominant and with areas of shadows, ANMS was able to reduce the density of concentrated corners and achieve a better spatial distribution across the image.

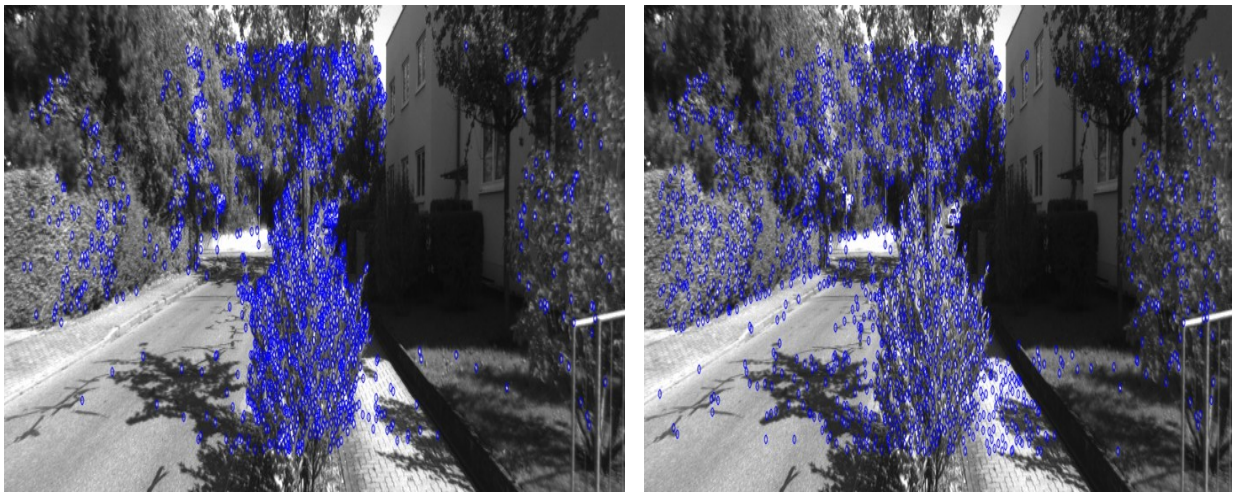


Figure 3-3: Illustration of the effect of applying adaptive non-maximal suppression (ANMS): [LEFT] ANMS disabled, [RIGHT] ANMS enabled.

A well-distributed set of features in the image will maximize the overlap of features across frames and will reduce the probability of the features being detected on a dynamic object, which will violate an assumption in visual odometry that the tracked features lie on static scene. Furthermore, by limiting the number of features used for tracking to a subset of potentially high

quality points, the computational requirements of the VO algorithm are reduced. A pseudo code of the ANMS algorithm is listed in Algorithm 3-1 below. Note that ROBUST_COEFFICIENT is a constant to be tuned manually. In this thesis, it is set to 1.1.

Algorithm 3-1. Adaptive non-maximal suppression

```
Adaptive-Non-Maximal-Suppression(Corners, NumToKeep)
  Sort-Descending(Corners) // Sort the corners in descending order based on corner response.
  radii.length = 0 // Initialize empty radii array
  for i = 1 to Corners.length
    response = Corners[i].response * ROBUST_COEFFICIENT
    radius = ∞
    j = 1
    while j < i and Corners[j].response > response
      current_radius = L2-Distance(Corners[i], Corners[j])
      radius = Minimum(radius, current_radius)
    end while
    radii.append(radius)
  end for
  anms_points.length = 0
  decision_radius = Largest-Element-At(radii, NumToKeep) // NumToKeep is passed as index.
  for i = 1 to radii.length
    if radii[i] ≥ decision_radius
      anms_points.append(Corners[i])
    end if
  end for
  return anms_points
```

3.4 Feature description

After the corners are extracted and adaptive non-maximal suppression applied, the next step is to compute a descriptor for the image patch around each corner. In this thesis, *Binary Robust Independent Elementary Features* (BRIEF) [19] descriptors are used. BRIEF is a binary descriptor where the descriptor vector is in the form of a binary string; in fact, BRIEF was the first binary descriptor developed. Binary descriptors offer many advantages over Histogram of Oriented Gradients (HOG) based patch descriptors such as SIFT [24] for constrained and real-time systems. The computations involved to arrive at these HOG descriptors are costly for many real-time and embedded systems. Furthermore, SIFT uses 128-dimension vector for its descriptor, and it uses floating-point numbers, which result in a descriptor size of 512 bytes.

Now, assuming hundreds or thousands of such feature descriptors, this will require a significant amount of memory which is most probably not feasible for constrained systems. Moreover, SIFT is patent-protected and thus cannot be freely used in applications.

Binary descriptor, on the other hand, are very fast to compute. For example, the chosen BRIEF algorithm computes the binary descriptor string by following the basic steps outlined below:

1. Smooth the target image patch.
2. Select a set of location pairs in the image patch in a unique way. The original paper describes five different sampling strategies.
3. For each selected location pair, compare the pixel intensity at the first point with that of the second one. If the intensity is larger, then append 1 to the descriptor string; otherwise append 0.

That is, the number of selected pairs will determine the length of the binary string. In this thesis, 256 pairs are used, resulting in a descriptor of size of 32 bytes. This, with the fact that descriptor computation has reduced to simple pixel intensity comparisons, offer significant advantage over HOG descriptors both in terms of computation and memory requirement.

As for comparing two descriptors with each other, *hamming distance* is used. Hamming distance is defined as the number of positions at which the corresponding bits are different. Hamming distance is simple and fast to compute because it is just an Exclusive-OR (XOR) operation and a bit count. That is, sum(xor(descriptor1, descriptor2)). This operation is especially fast on modern CPUs where *Single Instruction Multiple Data* (SIMD) instruction sets are employed. Note that BRIEF descriptors lack rotational invariance but this is not considered an issue because the ground mobile robot or vehicle is assumed to be moving on a locally planar ground.

3.5 Pose prediction

Before proceeding to the next step, a prediction of the current camera pose is performed. Dead-reckoning using wheel odometry can be used to perform such prediction since it is usually available in ground vehicles. In this thesis, however, a *constant velocity* motion model is used instead. Furthermore, in this simple motion model, constant frame rate is assumed, i.e., time is

not considered in the calculations. The model is calculated as follows: assuming the pose at frame k to be C_k, which consists of orientation represented as a quaternion $q_k \in SO(3)$, and position $t_k \in \mathbb{R}^3$. C_k is the pose to be predicted. The linear velocity at frame k is computed as:

$$v_k = t_{k-1} - t_{k-2} \tag{3.1}$$

This linear velocity is then smoothed over time in order to be robust against fast camera movements:

$$v_k = (v_k + v_{k-1})/2 \tag{3.2}$$

Similar approach is followed for the rotational component:

$$w_k = q_{k-1} * q_{k-2}^{-1} \tag{3.3}$$

and then the rotational velocity is also smoothed over time using the *spherical linear interpolation* (SLERP) operation:

$$w_k = slerp(w_k, w_{k-1}, 0.5) \tag{3.4}$$

After computing the new velocities, the predicted pose is easily computed as follows:

$$t_k = t_{k-1} + v_k \tag{3.5}$$

$$q_k = q_{k-1} * w_k \tag{3.6}$$

The predicted pose is then used to guide in the next local map tracking step.

3.6 Local map tracking

The next task is tracking the local map. The goal of this step is to correctly associate visible 3D map points with 2D image features. Note the associated 2D image features here are the ones extracted from the left camera image. The first step is to find the visible map points in both the left and right cameras. This is performed by means of a *view frustum*. A view frustum defines the region in space viewable by a camera. View frustums for both the left and right cameras are computed and the map points visible in both are passed to the next step. A sample illustration of a view frustum is shown in Figure 3-4. Note that for purposes of computing the view frustums and the subsequent operations, the stereo camera pose is assumed to be the predicted one.

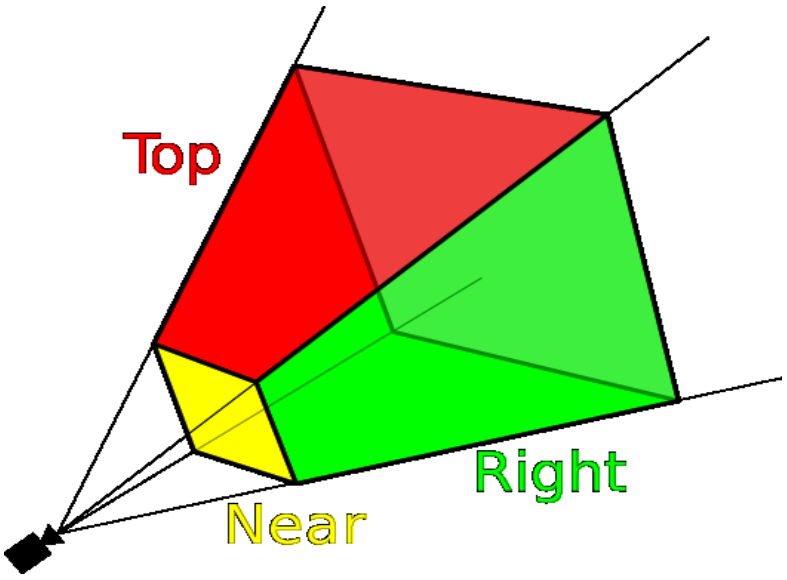


Figure 3-4: An illustration of a view frustum of a camera[8].

Visible 3D map points are then projected onto the left camera image plane by means of a projection matrix P:

$$P_{3\times4} = K_{3\times3} * [R_{3\times3}|t_{3\times1}] \tag{3.7}$$

Where K is the rectified camera intrinsic parameters matrix, R, t are the rotation and translation, respectively. The relations between rotation and orientation; and between translation and position are:

$$rotation = orientation^{-1} \tag{3.8}$$

$$translation = -orientation^{-1} * position \tag{3.9}$$

Each 3D map point X to be projected, will do so using the matrix P. However, this projection matrix is of size 3×4 which means the 3D map point will need to be transformed into homogeneous coordinates first and hence become the 4D vector $\hat{X}$. This is done by simply appending 1 as the fourth component:

$$\hat{X} = [x, y, z, 1]^T \tag{3.10}$$

[8] By MithrandirMage - Own workThis vector image was created with Inkscape., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=19952662

The projection is then simply performed as $\hat{x} = P\hat{X}$, the projected point $\hat{x}$ is in homogeneous 3D space which is transformed back to 2D space simply by dividing the first two components by the third one, to arrive at the 2D projected point $x = (u, v)$, where u, v are its location in the image.

The neighborhood of each projected map point is then examined for the best matching feature from the feature extraction step. In our current implementation, the search neighborhood is set to 25-pixel radius around each projected feature. Neighborhood search is accelerated by means of *spatial hashing*. During feature extraction step, the image plane is divided into a two-dimensional grid. Each cell of this grid is assigned the list of features that happen to be within its boundaries. Now, the projected feature's prospective grid cell is computed and thus the list of potential matches is readily available. The cells are assumed to be squares with a side length L of 25 pixels. The target cell of an image point is then simply calculated as $(\lfloor u/L \rfloor \quad \lfloor v/L \rfloor)$.

Once candidate neighborhood features are identified, finding the best match of the projected feature proceeds by using the widely accepted *ratio test* proposed by Lowe [24]. The ratio test works by comparing the distance of the closest neighbor to that of the second closest one. Nearest neighbor here is defined as the one with the minimum hamming distance for the BRIEF descriptor as described before. If the nearest feature is much closer than the second nearest one, then it has a higher probability of being the correct match. This projection then matching process is sketched in Figure 3-5. Once successful matches are found, the set of 3D points (of which matched projected points were projected) and their matched 2D detected points are passed to the next step.

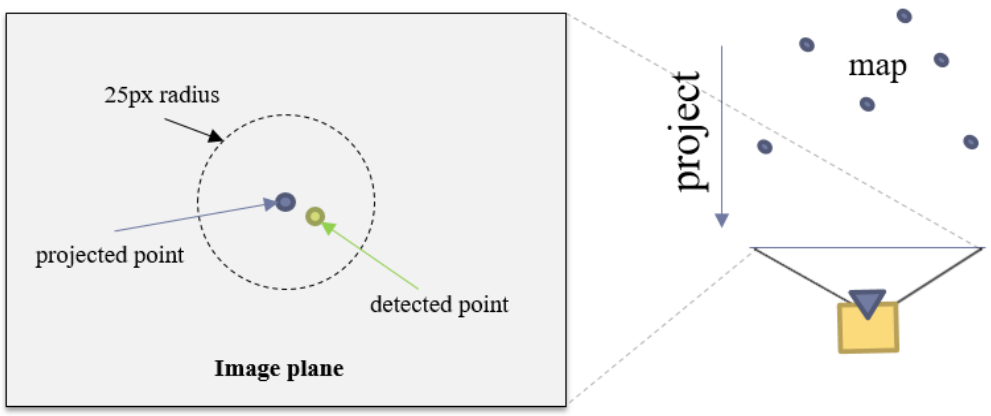


Figure 3-5: An illustration of the projection of local map [RIGHT] then searching for best match [LEFT].

3.7 Pose estimation

In the previous section, we have tried to associate 3D points from the local map with 2D image points detected in the current frame. These 3D-2D associations will be used to estimate the current pose, which is done by solving the *perspective from n points* (PnP) problem. PnP is the problem of estimating the pose of a calibrated camera given a set of 3D world points and their corresponding 2D image projections. There are several methods in the literature to solve the PnP problem, EPnP method proposed by Lepetit et.al. [35] will be used. EPnP is an accurate non-iterative solution to the PnP problem, with computational complexity that grows linearly with n, i.e., a computational complexity of $O(n)$. Furthermore, EPnP works in the general case of PnP for both planar and non-planar control points.

The effect of outliers, wrongful 3D-2D associations, was mentioned earlier along with RANSAC as a standard technique that is widely used to circumvent such effects on pose estimation. Over the years, there has been several variations to improve the standard RANSAC algorithm. For this work, we will employ *PROgressive SAmple Consensus* (PROSAC) [9] as our main outlier rejection scheme, i.e., EPnP computation is wrapped in PROSAC outlier rejection scheme. RANSAC treats all the correspondences (3D-2D associations) equally and draws random samples uniformly from the full set. On the other hand, PROSAC exploits the fact that all correspondences are not created equal. That is, they utilize a-priori knowledge of the quality of correspondences in order to favor high-quality matches. Basically, PROSAC sorts the correspondences according to their quality score, then progressively considers larger subsets for sampling correspondences, starting with top-ranked samples. Note that PROSAC draws the same samples that RANSAC would, just in different order. Depending on the data, this can be orders of magnitude faster than RANSAC. PROSAC provides the same solution guarantees as RANSAC does. The quality function of the correspondences employed for PROSAC is the descriptor distance ratio value of the nearest two neighbors computed during tracking. A sample size of 5 is used to compute the hypothesize model using EPnP.

The computed pose is further optimized in a second stage of computation. In this stage, the maximum consensus set of 3D-2D correspondences found from the previous PROSAC step will further enhance the pose by optimizing it with the goal of minimizing their image reprojection error:

$$arg\,min_{T_k} \sum_i \|x_i - \hat{x}_i\|^2 \tag{3.11}$$

where $x_i = (u, v)$ and $\hat{x}_i = (\hat{u}, \hat{v})$ are the projection and the measurement (i.e., associated 2D image feature) respectively of the 3D map point X_i. This minimization problem is solved in a least-squared sense.

The minimization problem is made robust by employing a robust M-Estimator in the cost function. Huber's M-Estimator [36] is used in this work. This minimization function is solved iteratively using the Levenberg–Marquardt [37] [38] algorithm. Levenberg–Marquardt algorithm is an optimization algorithm used to solve non-linear least squares problems, it combines the Gauss-Newton and the gradient descent method.

3.8 Local map initialization and maintenance

As was mentioned previously, the developed VO algorithm relies on tracking a sparse local map consisting of a set of 3D map points for its operation. When the VO system starts initially, this local 3D map is empty and thus needs to be initialized with 3D map points. By using a stereo camera, this map initialization process becomes relatively easy. Features are extracted and matched across the left and right camera images. The matches are used for triangulating new 3D points (Refer to Figure 2-5).

Ideally, a 3D point X in space visible in the left and right cameras, can be projected into two 2D image points in both images. In this setting, the 3D point can be reconstructed, i.e., triangulated, from the two image projections by extending rays from each projection and finding their intersection. Practically, the process is not that easy and straightforward. This is due to several potential noise factors involved, such as the digitization error of the images. Hartley and Sturm [39] discuss this issue in depth. In their paper, they properly formalize the triangulation problem with a noise model and compare different solution methods. The **Linear-LS** method is adopted for this work.

As the vehicle is moving, tracked features may go out of view. In order to keep the VO system going, once the number of tracked map points drops below 90% of the total number of local map points, map maintenance consisting of two operations is initiated:

1. Triangulating new 3D map points. For each feature in the left image that was not successfully tracked in the current frame, the best match is sought in the right image. The matches are then used for new triangulations of 3D map points. The same row matching and triangulation functions from the map initialization are used here.
2. Any map point that was not successfully tracked in the current frame is removed from the local map. This ensures that only useful tracked map points are kept alive, and the map memory requirements do not needlessly grow.

It should be emphasized how map points are kept alive in the map as long as they are successfully tracked. Once any map point cannot be tracked anymore, it is removed permanently from the local map and thus is not used for pose estimation anymore. This implies that the information contained in the full history of any tracked map point is utilized in pose estimation. This is an important aspect of the developed algorithm which will greatly reduce drift in the estimated motion trajectory.

3.9 Implementation remarks

The algorithm is implemented using the C++ programming language. OpenCV library was used to perform image processing tasks. Corner detector and descriptor extractor implementations available as part of OpenCV library was used. The general graph optimization (g2o) library [40] was used to perform the Levenberg–Marquardt minimization in the motion estimation step.

In its current implementation, the algorithm runs purely on the CPU with no GPU acceleration used. When a new stereo frame arrives, we spawn two threads to perform the feature extraction step on each image of the stereo image pair in parallel. That is, one thread will be performing corner detection, adaptive non-maximal suppression, and descriptor computation for the left image while another thread will be performing the same operations on the right image simultaneously. After the two threads finish the feature extraction step, the VO main loop will proceed normally on the main thread.

Finally, necessary data visualization and debugging tools were implemented, a snapshot is shown in Figure 3-6. This includes a 3D visualization of the local map points rendered as blue-colored 3D points and frustums each representing the estimated pose for each frame.

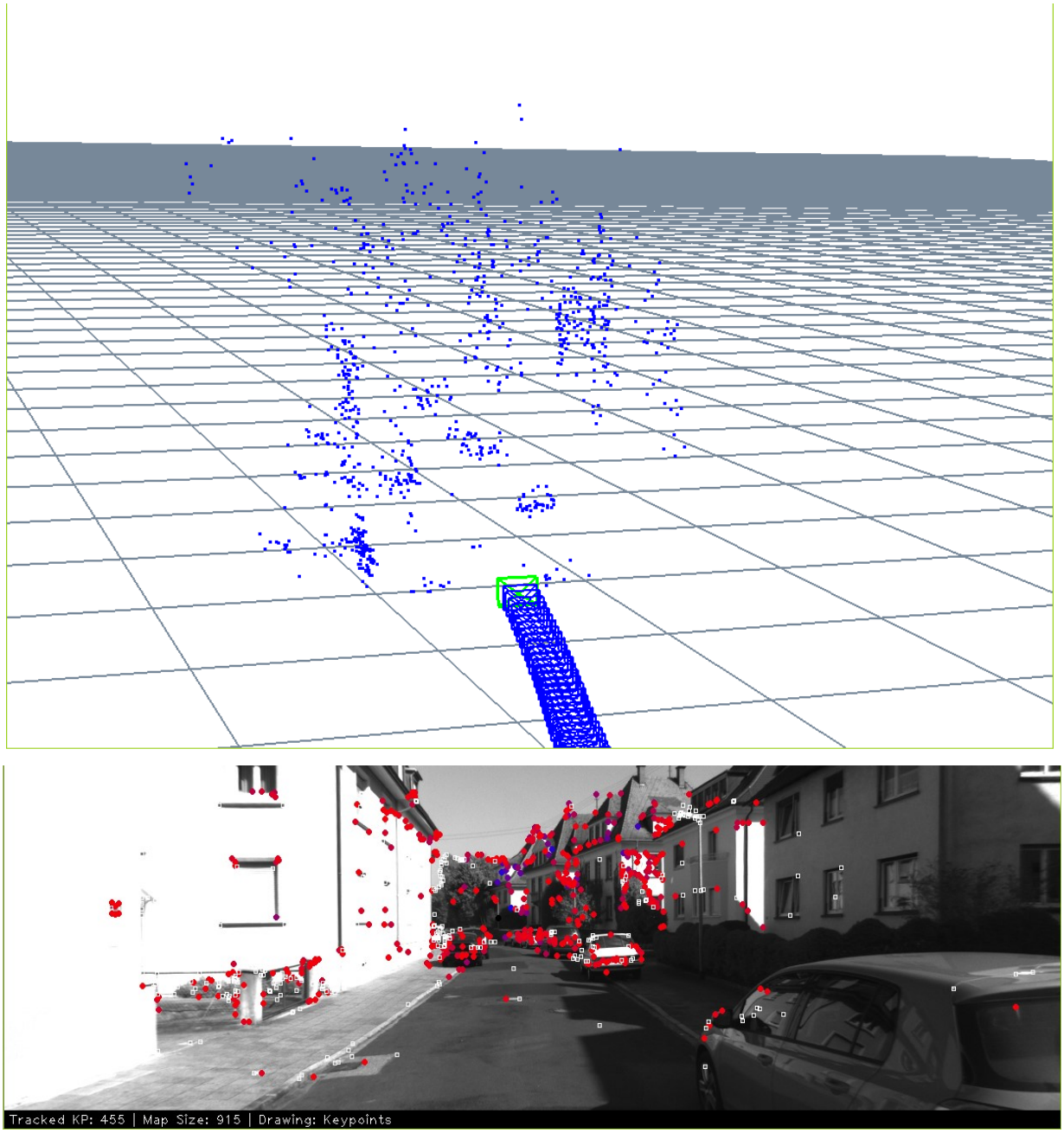


Figure 3-6: Visualization tools: [TOP] 3D view showing the points of the local map and camera path, [BOTTOM] Left image being processed showing features and several statistics.

Chapter 4

Evaluation

In this chapter, the results of extensive experimental evaluation of the visual odometry algorithm are presented and discussed. Experiments were performed on publicly available datasets and custom collected data as well.

4.1 Error metrics

In order to properly asses the localization accuracy of the visual odometry algorithm, two error metrics will be defined first. Metrics similar to the ones proposed by Sturm et.al. [41] will be followed. Assume the sequence of estimated poses from visual odometry is $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_n \in SE(3)$ and from the ground truth is $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_n \in SE(3)$.

The first metric is the *relative pose error* (RPE) which measures the local accuracy of the trajectory over a fixed time interval Δ. Thus, the relative pose error corresponds to the drift in the trajectory. The relative pose error at time step i is defined as:

$$\boldsymbol{E}_i := (\boldsymbol{Q}_i^{-1}\boldsymbol{Q}_{i+\Delta})^{-1}(\boldsymbol{P}_i^{-1}\boldsymbol{P}_{i+\Delta}) \tag{4.1}$$

In this way, we obtain $m = n - \Delta$ individual relative pose errors from a sequence of n camera poses. From these errors, we will compute the root mean squared error (RMSE) of the translational component:

$$RMSE(\boldsymbol{E}_{1:n}, \Delta) := \left(\frac{1}{m}\sum_{i=1}^{m}\|trans(\boldsymbol{E}_i)\|^2\right)^{1/2} \tag{4.2}$$

where $trans(\boldsymbol{E}_i)$ is the translational component of the relative pose error $\boldsymbol{E}_i$. Rotational errors are not computed explicitly as they show up as translational errors when the camera moves.

Whereas the first error metric is defined exactly the same as in [41], the second one is different. For the second metric, the *absolute translation error* (ATE), we compute the distance between the estimated trajectory from visual odometry and the ground truth along the plane in which the vehicle is moving, assuming this plane is the x-z plane, we have:

$$\boldsymbol{T}_i := \sqrt{(\boldsymbol{Q}_i^x - \boldsymbol{P}_i^x)^2 + (\boldsymbol{Q}_i^z - \boldsymbol{P}_i^z)^2} \tag{4.3}$$

where Q^x, Q^z are the x, z translation components of the ground truth pose, and P^x, P^z are the x, z translation components of the estimated pose from visual odometry. Then, we will compute the RMSE value:

$$RMSE(\boldsymbol{T}_{1:n}) := \left(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{T}_i^2\right)^{1/2} \tag{4.4}$$

4.2 KITTI evaluation

The KITTI dataset [11] is widely used for evaluating autonomous driving algorithms. The data set includes stereo greyscale and color image sets, alongside a ground truth which includes LIDAR data and high accuracy GPS position measurements. The dataset was collected by driving in different traffic scenarios in the city of Karlsruhe, Germany. Some of the challenging aspects of the dataset are the presence of dynamic moving objects (vehicles, cyclists, and pedestrians), the different lighting and shadow conditions as the vehicle is moving, and the presence of foliage which results in the detection of many non-stable and hard to track corners. We will use the first 11 sequences (00-10) of the visual odometry benchmark in our experiments as they have the ground truth data provided. The KITTI dataset has data recorded at 10Hz, this means that for RPE calculation, Δ will be set to 10, thus RMSE(RPE) represents the effective drift per second.

In all of the performed experiments on the KITTI dataset, the threshold for the AGAST corner detector has been set to 40 and the maximum number of retained corners after adaptive non-maximal suppression set to 1000. For our first experiment, we will assess the effect of tracking a local map and thus using full feature history in contrast to the traditional way of

tracking features and estimating motion between consecutive frames only. For this task, sequence 00 was executed using both methods. The result path plots are shown in Figure 4-1.

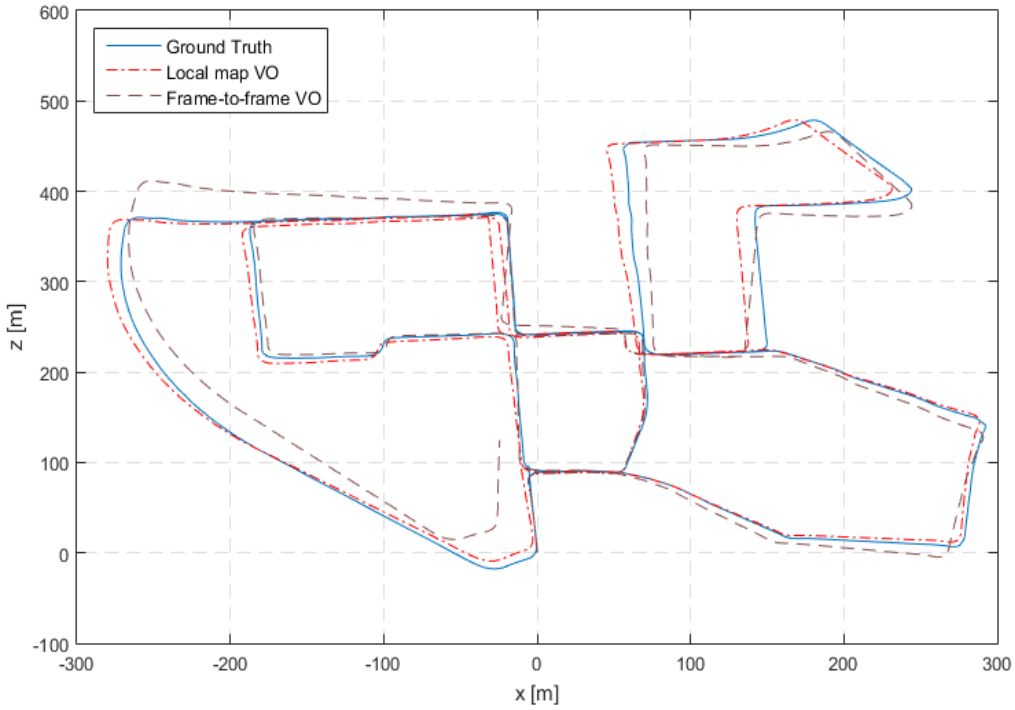


Figure 4-1: Path plot comparing the ground truth with our method (local map VO) and frame-to-frame VO using the sequence KITTI 00.

Sequence 00 is one of the long sequences in the dataset. It has a total length of 3.72km and consists of 4541 frames. It is clear from Figure 4-1 how employing a local map greatly reduces drift in the estimated path. To quantify the accuracy of both approaches, error metrics are computed. RMSE(RPE) of our local map approach is 0.19m/s with Δ set to 10 because the algorithm uses more than one frame for estimation. The unit is m/s in this case because Δ is set to 10 which is the same value at which the KITTI sequence was recorded. RMSE(ATE) is 8.00m. As for the frame-to-frame VO approach, RMSE(RPE) is 0.030m/frame with Δ set to 1 because only features from the previous frame are used in computing current frame motion. Thus it is expected to drift by 0.30m/s. RMSE(ATE) for this approach is 19.11m.

In order to gain more insight into the important role that tracking a feature for more than one frame plays in reducing drift, tracking age of each feature for the same KITTI sequence 00 has been recorded. *Feature age* is defined as the number of consecutive frames at which this feature

was successfully tracked and thus used in pose estimation. When a feature is first triangulated into a 3D point and added to the local map it has an age of zero. Figure 4-2 shows a histogram showing the number of features versus their age. We can see that the majority of features have age less than 10 frames. The maximum feature age was found to be 71 frames, while the mean is 2.74 frames.

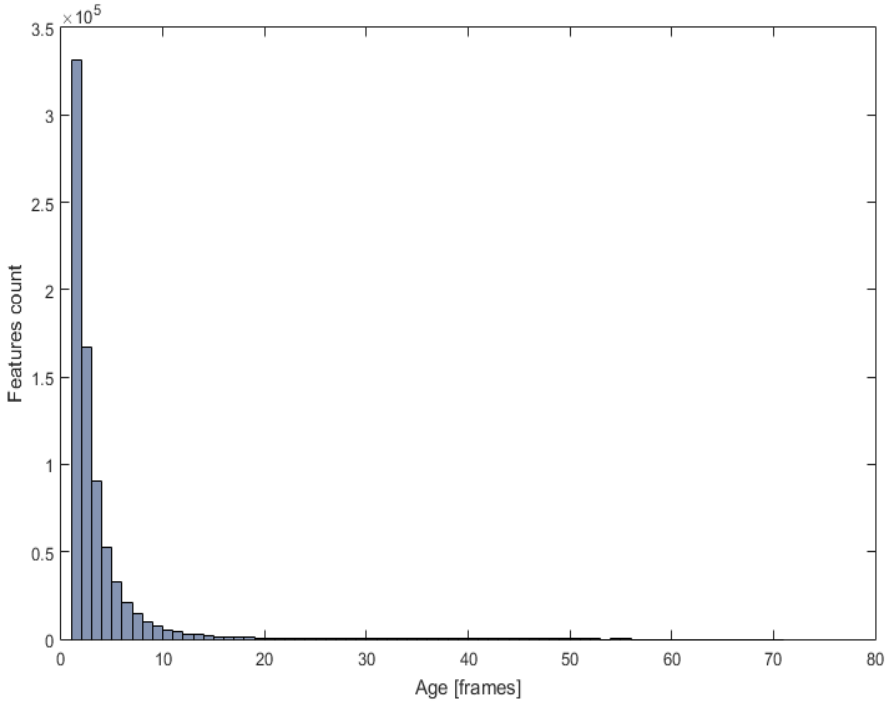


Figure 4-2: A histogram showing the number of features and their age.

Figure 4-3 shows a sample frame from the dataset depicting features and their tracks. In this illustration, new features are drawn in red and as they get older, they start turning into blue gradually. Small white boxes represent detected features that was not tracked.

The result path plots on the rest of KITTI sequences are shown in Figure 4-4, and the error metrics in Table 1. RPE was calculated with Δ set to 10. All the sequences were processed using the same parameter tunings. The algorithm was able to process the sequences successfully except sequence 01 which is a highway and has very few features to track. Sequences 00, 02, 05, 06, 07, 09 contain loops, however, being only a VO algorithm, these loops are not used in attempt to correct the estimated trajectory.

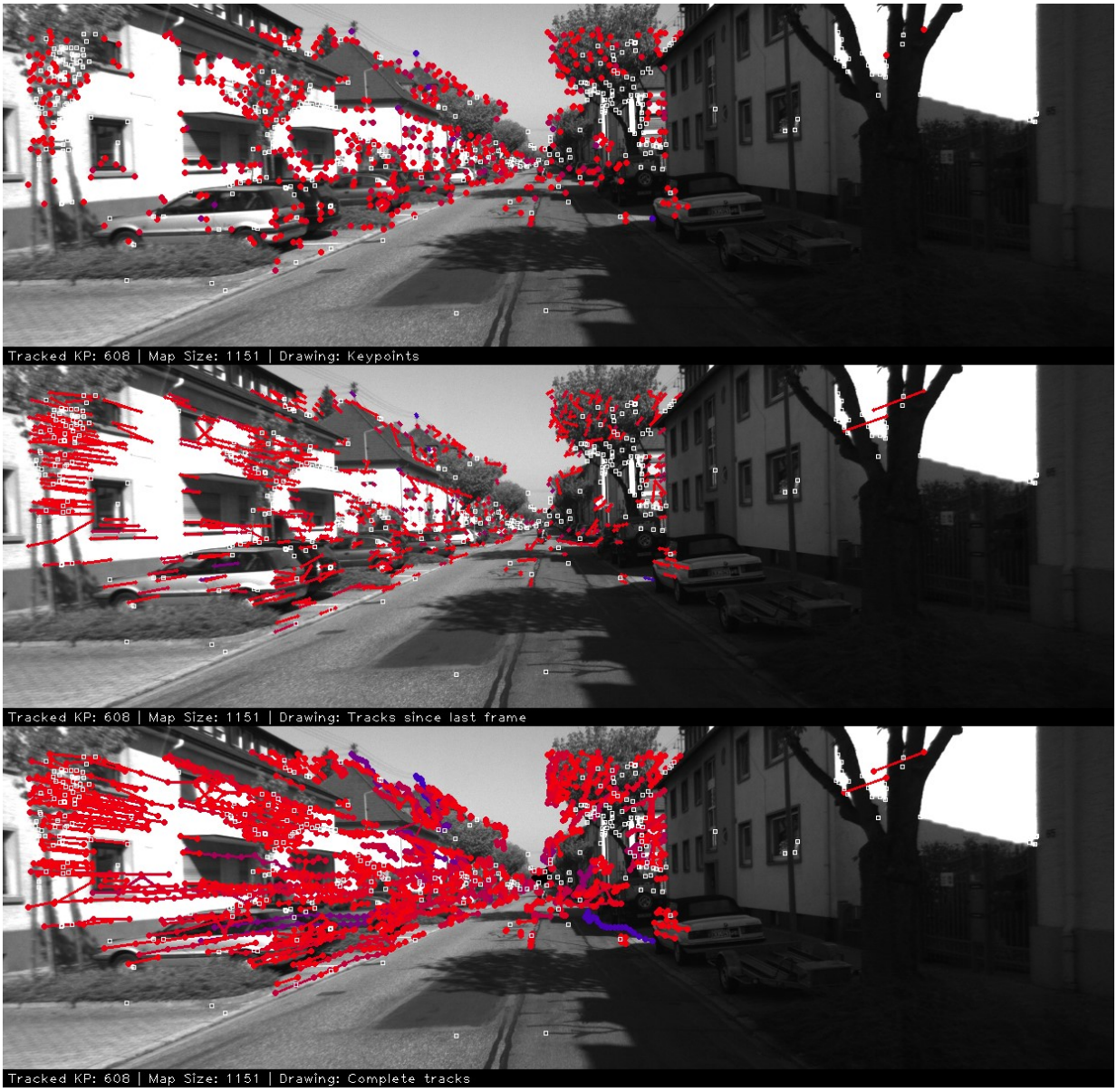


Figure 4-3: An example showing features and their tracks: [TOP] detected features only [MIDDLE] features tracks from the previous frame only [BOTTOM] the complete feature tracks where each knot represents the location where this feature was tracked in its corresponding frame.

The computational performance of the algorithm was evaluated using a standard personal computer with an Intel i7-4790 CPU at 3.60GHz. The images in the KITTI dataset has a resolution of 1241×376. The same KITTI sequences (00-10) with the exception of KITTI 01 are used in this performance evaluation. The mean frame processing time was found to be 27.57ms with standard deviation of 9.92ms. That is, the algorithm is processing frames at an average rate of 36.23Hz. Note that the images used are grayscale and pre-rectified and thus these computations are not included in the reported measurements.

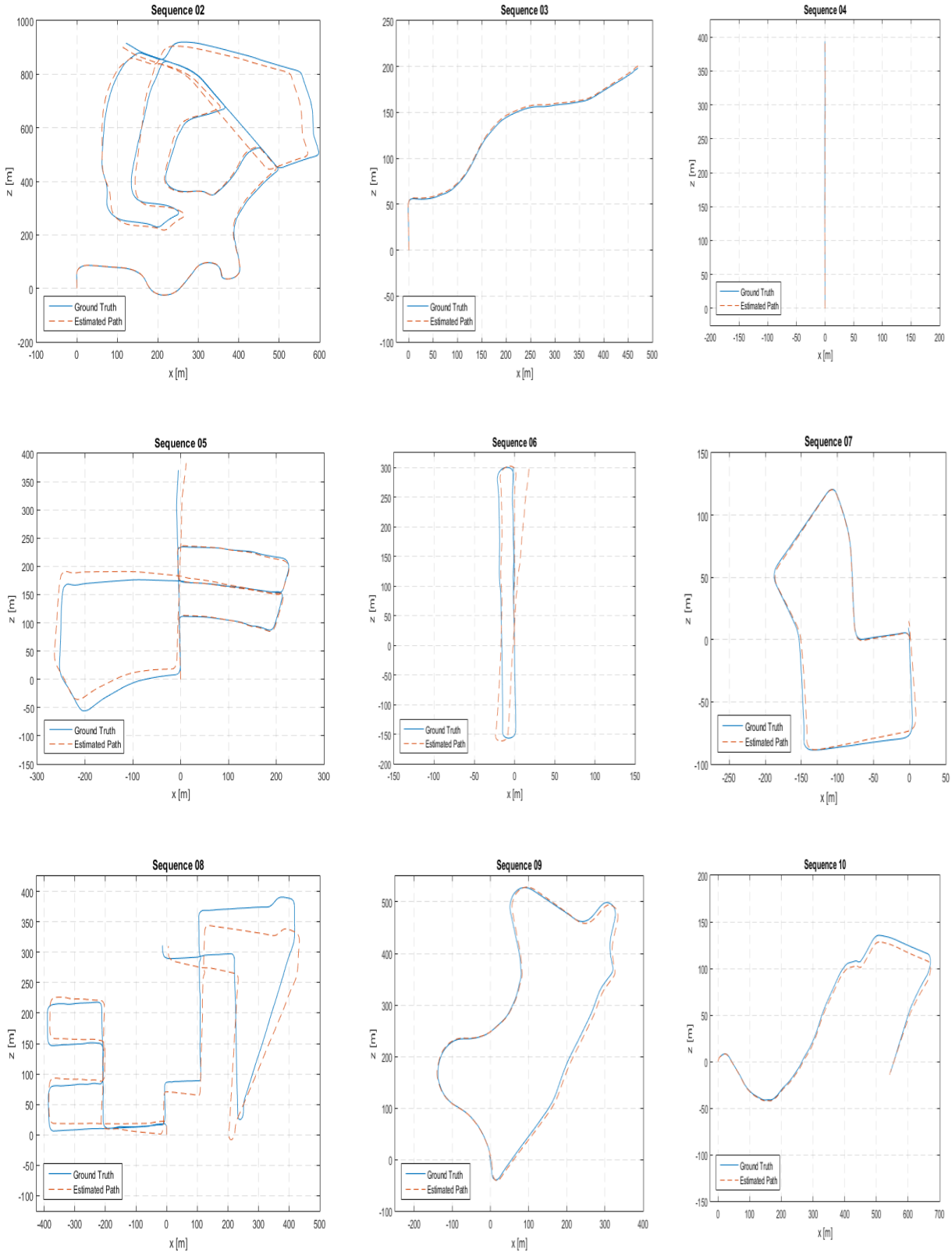


Figure 4-4: Result path plots of our algorithm on KITTI sequences (02-10).

Sequence	Length [km]	RMSE(RPE) [m/s]	RMSE(ATE) [m]
KITTI 00	3.7223	0.1994	8.0042
KITTI 01	xx	xx	xx
KITTI 02	5.0605	0.1996	18.0647
KITTI 03	0.5590	0.1424	2.0544
KITTI 04	0.3936	0.1760	1.5584
KITTI 05	2.2046	0.1370	12.4687
KITTI 06	1.2326	0.2201	7.3284
KITTI 07	0.6944	0.1167	3.8801
KITTI 08	3.2137	0.3396	27.7067
KITTI 09	1.7025	0.2416	5.3240
KITTI 10	0.9178	0.1207	4.9404

Table 1: Results of the VO algorithms in the KITTI dataset.

4.3 New College evaluation

The next evaluation experiment is performed using the New College dataset. The dataset was collected using a mobile robot while traversing 2.2km through New College in Oxford campus grounds and adjacent parks. The images were captured by a stereo camera at a rate of 20Hz, and the rectified images have a resolution of 512×384. What is especially challenging about this dataset is the fast rotations of the robot. The dataset also has several loops. The developed algorithm was able to process the complete sequence successfully. For this dataset, the corner detector threshold was set to 25 and the maximum number of retained corners to 500.

No accurate ground truth is provided as part of the official dataset distribution. In order to assess the accuracy of our path estimation, the computed results are compared to that of ORB-SLAM2 [42]. ORB-SLAM2 is one of the most accurate feature-based Visual SLAM systems available. It was able to successfully detect and close some of the loops in the sequence, and it performed full bundle adjustment. Assuming the path estimated from ORB-SLAM2 to be the ground truth, the estimated RMSE(RPE) for the developed algorithm is 0.0614m/s with Δ set to 20, and the estimated RMSE(ATE) is 20.35m. A plot of the developed VO's estimated path against that from ORB-SLAM2 is shown in Figure 4-5. As for computational performance, the mean frame processing time is 14.8ms with a standard deviation of 6.69ms.

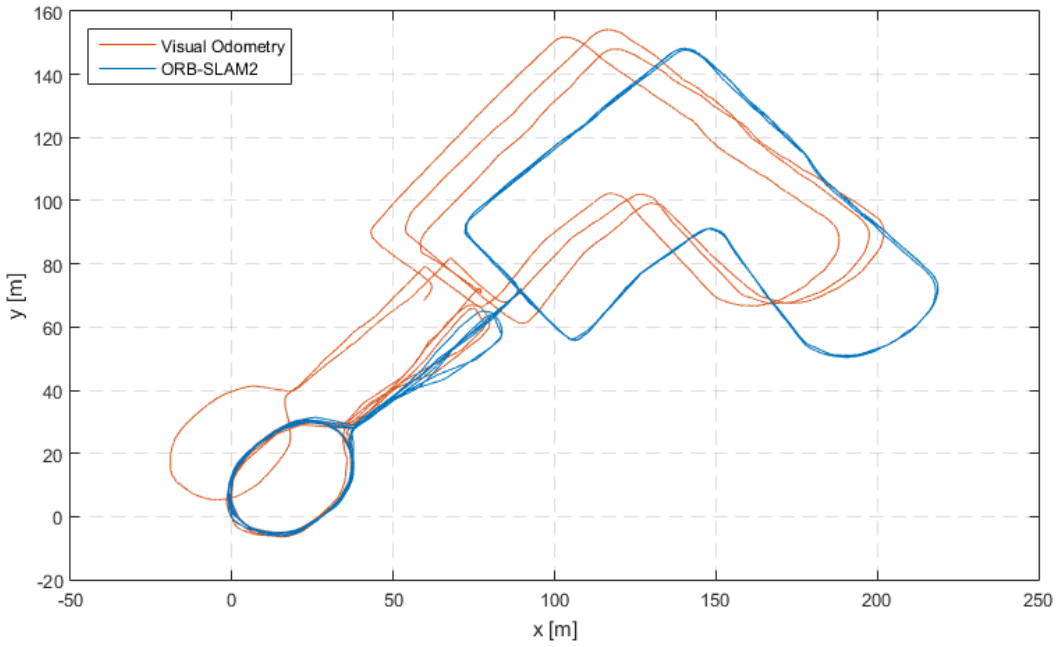


Figure 4-5: A plot showing the estimated path using the developed algorithm against ORB-SLAM2 estimation on the New College dataset.

4.4 Manually collected data

In this section, the results of two experimental evaluations on manually collected data are shown. For both experiments, ZED stereo camera from STEREOLABS[9] is used.

4.4.1 Mobile robot experiment

The first experiment is a simple proof-of-concept. The ZED stereo camera is installed on a 3-wheeled mobile robot shown in Figure 4-6, and was configured to capture data at 720p resolution at 30Hz. The robot was then driven in the University of Michigan-Dearborn campus near The Institute for Advanced Vehicle Systems (IAVS). A sample frame from this sequence is shown in Figure 4-7, and the path plot of this simple trajectory is shown Figure 4-8. The experiment was performed on a sunny day with the sun shines on the camera on some frames causing some difficulties in corner detection. Furthermore, the robot had abrupt left and right movements during its course, i.e., its movement wasn't smooth trajectory, hence adding more challenges to the VO system.

[9] www.stereolabs.com

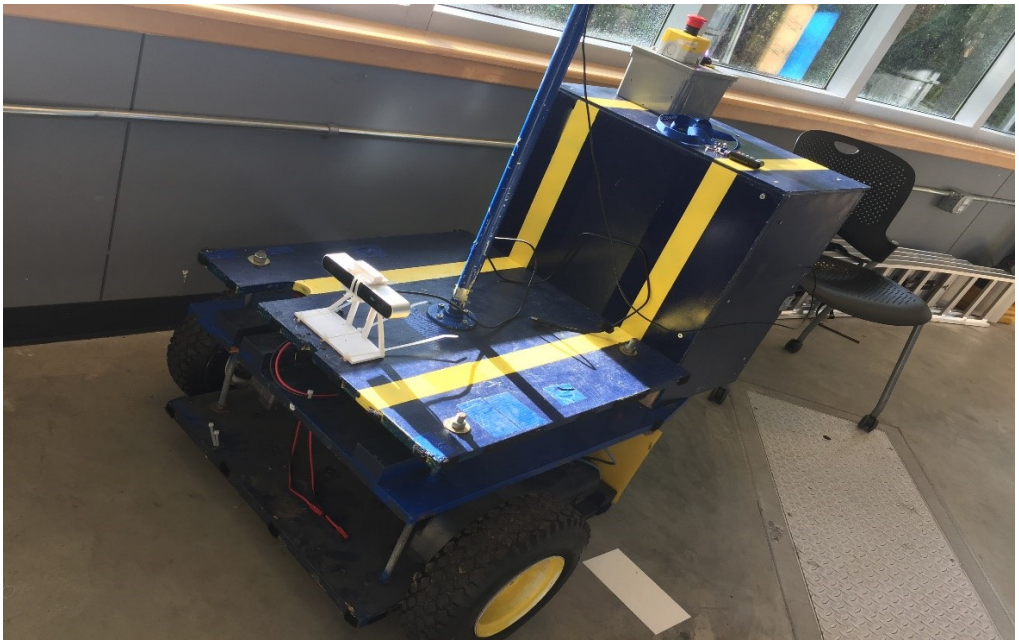


Figure 4-6: Mobile robot setup used in the first experiment.

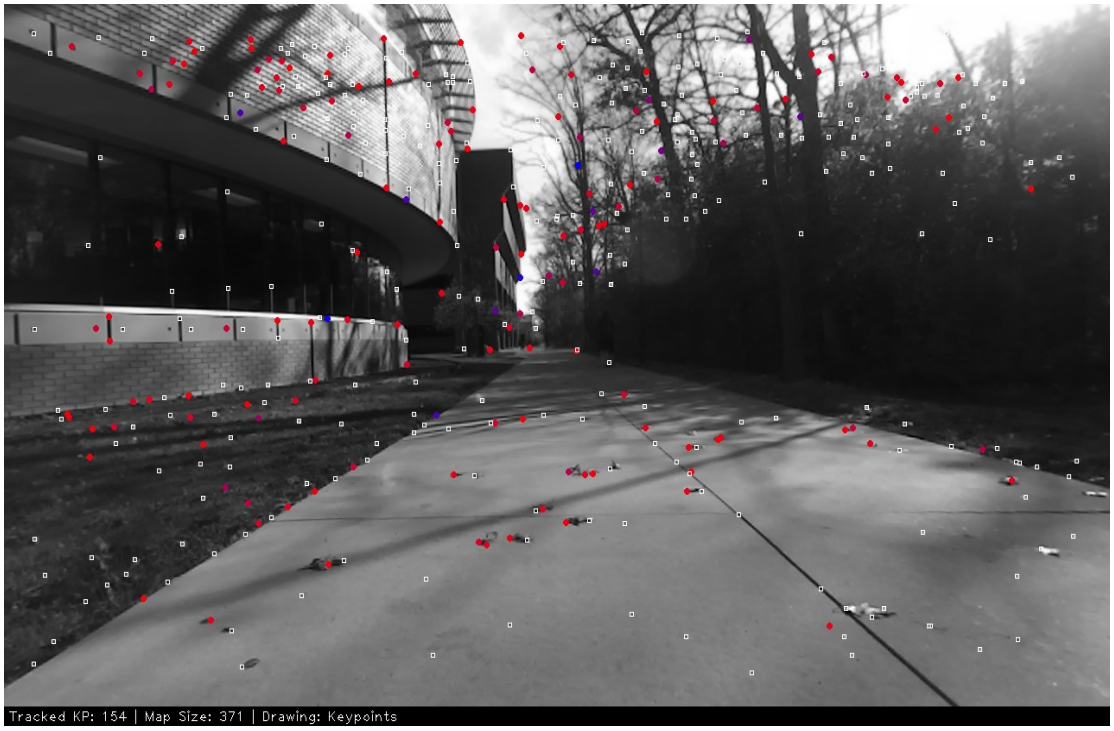


Figure 4-7: A sample frame from the mobile robot experiment sequence.

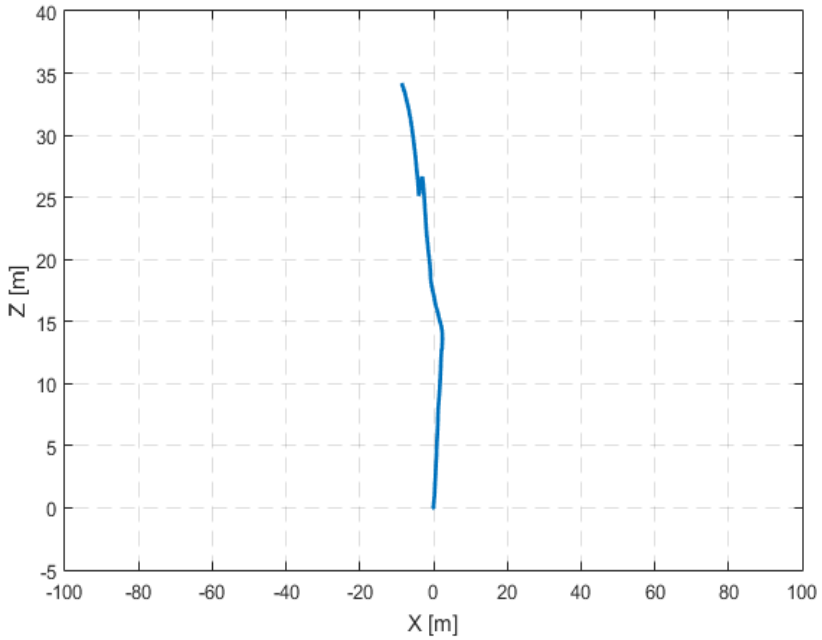


Figure 4-8: Path plot for the mobile robot experiment.

4.4.2 Car experiment

In this experiment, the ZED camera was installed on the roof of a car as shown in Figure 4-9.

Figure 4-9: Illustration showing the ZED camera installation for the car experiment.

The ZED camera was set to collect data at 720p resolution at 30Hz. The car was driven in a loop around the University of Michigan-Dearborn. The collected data was extremely challenging for the VO algorithm. First, the sky was very cloudy causing major portions of the scene to be shadowed (in shade). This, coupled with the presence of foliage, caused the corner detector to fail in detecting reliable corners for tracking with the default threshold value. Only after the corner threshold was lowered down to 10, it was able to detect enough corners for the VO algorithm to work. Such corners, however, are expected to be of low quality and hence result in low quality tracks. Second, the video stream suffered from many dropped and some corrupted frames. The VO algorithm in its current implementation, is however unable to handle such intermittent interruptions. For this reasons, the algorithm was unfortunately unable to process this data sequence and provide an accurate estimate of the trajectory. A sample frame from this experiment where the VO algorithm fails is shown in Figure 4-10. This experiment has motivated many improvements in the data collection process for future experiments, and inspired research questions for future enhancements for the VO algorithm.

Figure 4-10: Illustration showing a frame from the car sequence where the VO algorithm fails.

Chapter 5

Conclusion

This thesis presented the development and evaluation of a stereo visual odometry algorithm. Visual odometry, or the process of estimating motion trajectory from visual input, is a powerful technique. It was shown how cameras play an important role in the sensor suite enabling advanced autonomy applications. For example, visual odometry plays a central role in Mars Exploration Rovers as was discussed. Such real-time estimates of motion provided by visual odometry are necessary for timely action by navigation and control modules. The thesis then went and discussed essential concepts from computer vision and multi-view geometry.

The developed visual odometry algorithm was then presented in detail. The developed algorithm employs a stereo camera which allows reconstruction in true scale as the fixed baseline distance of the stereo camera is known. This also allows for a *power-and-go* system with no tricky initialization procedures which are typically necessary for monocular approaches. The algorithm is *feature-based* relying on detecting corner features and using them for triangulation and tracking. Adaptive non-maximal suppression for feature distribution across the image, spatial hashing for speeding up feature look-up during tracking, and using the binary BRIEF descriptors for feature matching were all illustrated. The algorithm then performs motion estimation by solving the PnP problem wrapped in PROSAC outlier rejection scheme. The found consensus set is fed into a final optimization step by minimizing its re-projection errors.

The main distinguishing feature of the developed visual odometry algorithm is its sparse 3D point-based local map. The visual odometry algorithm will keep updating this local map by triangulating new 3D points and appending them to it, and cleaning-up the 3D points that are no longer tracked. This in-turn enables long tracks across multiple frames which means the full history of each tracked feature is utilized for pose estimation. This technique was shown to greatly reduce drift in the estimated motion trajectory and hence enables long-term operation.

The visual odometry algorithm was then evaluated on public and custom datasets. KITTI dataset, which is the most common one used for evaluating self-driving algorithms, was used as the main evaluation and testing dataset. The algorithm was also evaluated on the New College dataset which was collected by a mobile robot traversing college campus and surrounding parks. The main challenge with this dataset was the abrupt movements of the mobile robot causing challenging motion blur. The developed algorithm was able to perform competitively on the said datasets as was demonstrated, especially with the fact that it is a visual odometry algorithm with no global mapping or loop-closing.

The algorithm was also tested on custom data, however due to project timing constraints, these experiments were minimal. The main finding from these experiments was that the algorithm failed in adverse weather condition. Such results motivate enhancements to the data collection process, and to the visual odometry system. For example, the current system does not handle intermittent interruptions or loss of tracking.

Potential future work includes:

1. Employing line segment features. Line segments are expected to have more meaningful information especially in urban environments than corner features. Furthermore, line segments are expected to be more detectable during motion blur when the mobile robot is performing quick turns.
2. Relying more on geometrical constraints rather than appearance-based approaches. This is expected to provide superior results in adverse weather and low-light conditions.
3. Employing a Kalman Filter at the output of visual odometry in order to provide smoother motion estimates and to *bridge* intermittent loss of tracking. As the system is non-linear, Extended Kalman Filter (EKF) or Unscented Kalman Filter (UKF) shall be used.

Bibliography

[1] A. R. Richardson, "Learning and Searching Methods for Robust, Real-Time Visual Odometry," *PhD dissertation, University of Michigan,* 2015.

[2] Y. Cheng, M. Maimone and L. Matthies, "Visual Odometry on the Mars Exploration Rovers," *IEEE International Conference on Systems, Man and Cybernetics,* vol. 1, pp. 903-910, 2005.

[3] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics & Automation Magazine,* vol. 18, no. 4, pp. 80-92, 2011.

[4] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover (No. STAN-CS-80-813)," STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.

[5] D. Nister, O. Naroditsky and J. Bergen, "Visual Odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* vol. 1, 2004.

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

[7] M. Pollefeys, D. Nist´er, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stew´enius, R. Yang, G. Welch and H. Towles, "Detailed real-time urban 3d reconstruction from video," *International Journal of Computer Vision,* vol. 78, no. 2-3, pp. 143-167, 2008.

[8] E. S. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *International Journal of Robotics Research,* vol. 30, no. 4, pp. 407-430, April 2011.

[9] O. Chum and J. Matas, "Matching with PROSAC – Progressive Sample Consensus," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05),* vol. 1, pp. 220-226, 2005.

[10] H. Badino, A. Yamamoto and T. Kanade, "Visual Odometry by Multi-frame Feature Integration," *Proceedings of the IEEE International Conference on Computer Vision Workshops,* pp. 222-229, 2013.

[11] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 3354-3361, 2012.

[12] M. Smith, I. Baldwin, W. Churchill, R. Paul and P. Newman, "The new college vision and laser data set," *The International Journal of Robotics Research,* vol. 28, no. 5, pp. 595-599, 2009.

[13] I. E. Lopez, "Large scale semantic 3D modeling of the urban landscape," *PhD thesis,* 2012.

[14] J.-Y. Bouguet, "Camera calibration toolbox for MATLAB," 2004.

[15] R. Szeliski, "Computer Vision: Algorithms and Applications," *Springer Science & Business Media,* 2010.

[16] "A Short Introduction to Descriptors." Gil's CV Blog. Web. 28 Jan. 2017.".

[17] C. Harris and M. Stephens, "A combined corner and edge detector," *Alvey vision conference,* vol. 15, no. 50, 1988.

[18] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *European conference on computer vision,* pp. 430-443, 2006.

[19] M. Calonder, V. Lepetit, C. Strecha and P. Fua, "Brief: Binary robust independent elementary features," *European conference on computer vision,* pp. 778-792, 2010.

[20] A. Alahi, R. Ortiz and P. Vandergheynst, "FREAK: Fast Retina Keypoint," *IEEE conference on computer vision and pattern recognition (CVPR),* 2012.

[21] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *International conference on computer vision,* pp. 2564-2571, 2011.

[22] R. G. von Gioi, J. Jakubowicz, J.-M. Morel and G. Randall, "LSD: a Line Segment Detector," *Image Processing On Line 2,* pp. 35-55, 2012.

[23] J. Matas, O. Chum, M. Urban and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," *Image and vision computing,* vol. 22, no. 10, pp. 761-767, 2004.

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

[25] H. Bay, T. Tuytelaars and L. Van Gool, "SURF: Speeded Up Robust Features," *Computer vision and image understanding,* vol. 110, no. 3, pp. 346-359, 2008.

[26] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE robotics & automation magazine,* vol. 13, no. 2, pp. 99-110, 2006.

[27] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE transactions on pattern analysis and machine intelligence,* vol. 29, no. 6, 2007.

[28] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), Nara, Japan,* p. 225–234, November 2007.

[29] H. Strasdat, J. M. M. Montiel and A. J. Davison, "Real-time Monocular SLAM: Why Filter?," *IEEE International Conference on Robotics and Automation (ICRA),* 2010.

[30] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics,* vol. 31, no. 5, pp. 1147-1163, October 2015.

[31] "LSD-SLAM: Large-scale direct monocular SLAM," *European Conference on Computer Vision (ECCV), Zurich, Switzerland,* pp. 834-849, September 2014.

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

[33] E. Mair, G. D. Hager, D. Burschka, M. Suppa and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," *Proceedings of the European Conference on Computer Vision (ECCV'10),* 2010.

[34] M. Brown, R. Szeliski and S. Winder, "Multi-image matching using multi-scale oriented patches," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) ,* vol. 1, pp. 510-517, 2005.

[35] V. Lepetit, F. Moreno-Noguer and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal Computer Vision,* vol. 81, no. 2, pp. 155-166, 2009.

[36] P. J. Huber, Robust statistics, Berlin Heidelberg: Springer , 2011.

[37] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics,* vol. 2, no. 2, pp. 164-168, 1944.

[38] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics,* vol. 11, no. 2, pp. 431-441, 1963.

[39] R. Hartley and P. Sturm, "Triangulation," *Computer vision and image understanding,* vol. 68, no. 2, pp. 146-157, 1997.

[40] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, "g2o: A general framework for graph optimization," *IEEE International Conference on Robotics and Automation (ICRA),* pp. 3607-3613, 2011.

[41] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," *Proceedings of the International Conference on Intelligent Robot Systems (IROS),* October 2012.

[42] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras," *arXiv preprint ,* 2016.