Università degli Studi di Padova

Facoltà di Ingegneria Industriale

Corso di Laurea Magistrale in
Ingegneria Aerospaziale

Implementation and experimental verification of a 3D-to-2D Visual Odometry algorithm for real-time measurements of a vehicle pose

Giulia Soldà

Supervisor:
Prof. Stefano Debei

Assistant supervisor:
Marco Pertile

Anno accademico 2014-2015

Abstract

This work describes the implementation of a software for real-time vehicle pose reconstruction using stereo visual odometry.

We implemented the visual odometry procedure firstly by guessing motion by a linear 2D-to-3D method solving a PnP problem embedded within a random sample consensus process (i.e., a RANSAC scheme) to remove outliers. Then, a maximum likelihood motion estimation is performed minimizing a non-linear problem. The software is tested on various datasets in order to evaluate its accuracy and performance.

Regarding the image processing steps, many feature detectors and descriptors have been tested to provide best accuracy with the least computational time. To achieve real-time execution, GPU implementations of feature extraction and matching algorithms are used.

The developed software is tested on the NVIDIA Jetson TK1. This board is the main computer of a project called MORPHEUS that involves the design and building of the first robotic rover of the University of Padova. The system proposed allows the rover to determine its real-time location and 3D orientation.

In this paper we present the results obtained, demonstrating an high degree of reliability and an accuracy of better than 1% over 1350 mm of travel.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In planetary exploration space mission, motion estimation - and then localization - of a vehicle on the surface of a planet is a key task. The typical approach for vehicle localization involves the use of a combination of wheel odometry (from joint encoders) and inertial sensing (e.g., gyroscopes and accelerometers), but presents some limitations: inertial sensors are subjects to unacceptable drifts and wheel odometry is unreliable in sandy or slippery terrain since wheels tend to slip and sink. Moreover, the robot may operate in areas where GPS transmissions can not be received or on planets that has not yet been equipped with a set of GPS satellites, like on Mars.

Another process adopted in order to determining vehicle orientation and position is called Visual Odometry (VO). It is the process of determining a camera (or cameras) 6 degree of freedom position and orientation within a rigid scene using only a sequence of images. When the cameras are mounted on a vehicle this technique is also known as *ego-motion* estimation because the cameras are moving with the vehicle and the camera motion is the vehicle one. The term "visual odometry" was coined by Nister in 2004 in his landmark paper [1]. Likewise wheel odometry, which relies on rotary wheel encoders, VO incrementally estimates vehicle pose examining the changes that motion induces on the images of its on-board cameras.

The advantage of VO is that it is insensitive to soil mechanics (e.g., is not affected by wheel slip in uneven terrain or other adverse conditions), produce a full 6 degrees of freedom motion estimate, and has lower drift rates than all but the most

expensive IMUs. Additionally, its usage is not limited to wheeled vehicles. On the other hand, for VO to work effectively, sufficient illumination in the environment and at least partially static scene with enough texture should be present. This capabilities makes VO a natural complement to wheel odometry and to other navigation systems such as GPS and IMUs.

The problem of estimating a vehicle ego-motion from visual input alone has been some of the most active fields of research in computer vision for the last three decades. Very impressive results have been obtained over long distances using monocular, stereo systems and omni-directional cameras. Most of the early research in VO was done for planetary rovers and was motivated by the NASA Mars exploration program for provide all-terrain rovers with the capability to measure their 6 degree of freedom motion in the presence of wheel slippage in uneven and rough terrains. Only during the last decade real-time implementations expanded, allowing VO to be used on another planet by three Mars exploration rovers for the first time.

As mentioned before the image pairs may be generated from either single camera systems (i.e., monocular VO) or stereo cameras (i.e., stereo VO). Using stereo image pairs for each frame helps reduce error and provides additional depth and scale information. When compared to monocular video, motion estimation from stereo images is relatively easy and tends to be more stable and well behaved [1].

In contrast to the SLAM method, VO not maintain a map of the environment. In this way VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera.

A historical review of the first 30 years of research in stereo VO is presented in [2], starting from the early 1980s with the Moravec's work [3]. Cheng et al. [4] give an interesting insight into the importance of stereo odometry during NASA's *Mars Exploration Rover* missions with the rovers Spirit and Opportunity.

Many others recent applications of stereo odometry on different types of robots in various environments can be found. For example in [1] and [5] Nister et al. presented a system that estimates the motion of a stereo head or a single moving camera based on video input. In particular, the stereo camera system mounted on an autonomous ground vehicle demonstrated surprisingly accurate results for hundreds

of meters. In [6] a real-time, low-cost system to localize a mobile robot in outdoor environments is presented. This system combines a visual odometry approach with inertial measurements to fill in motion estimates when visual odometry fails. This incremental motion is then fused with a low-cost GPS sensor using a Kalman Filter to prevent long-term drifts. Olson et al. [7] described a methodology for long-distance rover navigation that uses robust estimation of ego-motion. They also noted that the use of an orientation sensor reduces the error growth to linear in the distance travelled and results in a much lower error in practice. Another approach presented in [8] uses a specialized method of Sparse Bundle Adjustment (SBA). Howard [9] describes a real-time stereo visual odometry algorithm that is particularly well-suited to ground vehicle applications.

As can be seen there are a lot of studies that evaluated several visual odometry approaches and compare their implementations. Most of these algorithms have also on-line source code available. Visual odometry is therefore become in these last ten years a very attractive field in robotics and computer vision.

Chapter 2

General problem

2.1 Introduction

The present work wants to describe a real-time stereo visual odometry algorithm that is particularly well-suited to ground vehicle applications.

As mentioned in the previous chapter, the goal of the VO procedure is to calculate the displacement of a calibrated stereo system using the images acquired in an initial position and in a second one.

Referring to figure 2.1, let us consider a vehicle that moves and takes images with a rigidly attached camera system at discrete time instant k. At every instant of time the stereo system take a left image and a right one, denoted by I_{Lk} and I_{Rk}. For simplicity, the coordinate system of the left camera is assumed to be the origin.

Two camera positions - at time $k-1$ and k - are related by the rigid body transformation

$$T_k^{k-1} = \begin{bmatrix} R_k^{k-1} & t_k^{k-1} \\ 0 & 1 \end{bmatrix}, \tag{2.1}$$

where R_k^{k-1} is the rotation matrix and t_k^{k-1} the translation vector.

The set of camera poses $C = \{C_0, ..., C_n\}$ contains the transformation of the camera with respect to the initial coordinate frame at $k = 0$.

The main task of VO is therefore to compute the relative transformation T_k^{k-1} from the images I_k and I_{k-1} using a feature-based method and then to concatenate the transformations to recover the full trajectory of the camera. In this way VO

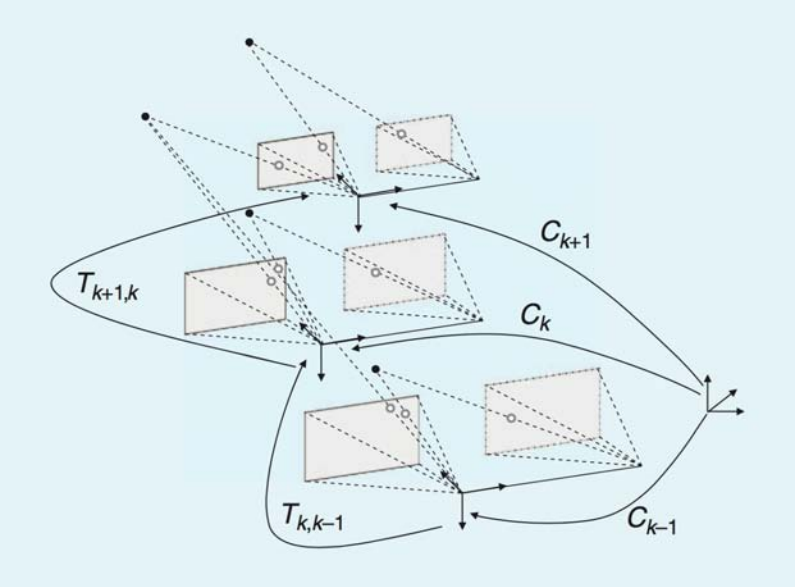


Figure 2.1: An illustration of the visual odometry problem. The relative poses T_k^{k-1} of adjacent camera positions are computed and concatenated to get the absolute pose C_k with respect to the initial coordinate frame at $k = 0$ [2].

recovers the path incrementally, pose after pose.

Therefore assuming that the robot equips a calibrated stereo camera system, whose intrinsic parameters and rigid relative pose are known through the camera calibration process (see section 2.2 for the camera model details), the stereo visual odometry process involves different steps (see the diagram in figure 2.2).

- Capture two images and undistort them to mathematically remove radial and tangential lens distortion. This is called *undistortion* and is detailed in section 2.2.2.

- Extract features from each new frame in the left image (*feature detection*) and obtain corresponding feature point in the right image. The output of this step is a disparity map that contains the differences in coordinates on the image plane of the same feature viewed in the left and right cameras. Moreover, we can find a second type of correspondences observing the same feature at two different instants of time. The first process is called *feature matching*, while the second is named *feature tracking*. Feature detection and matching are described in section 2.3.

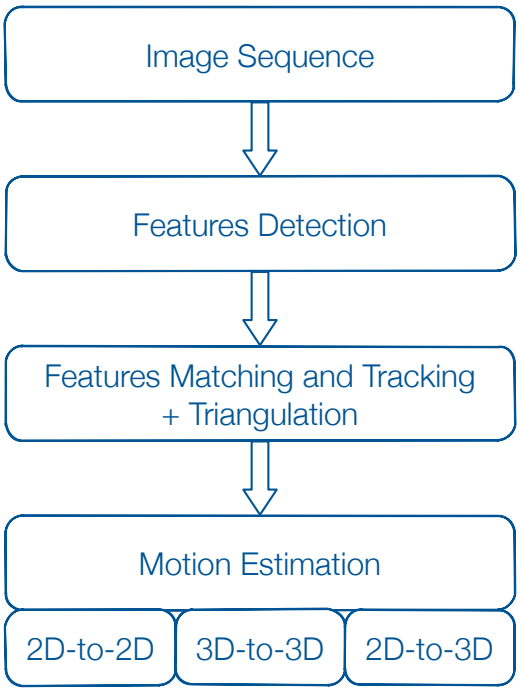


Figure 2.2: A block diagram showing the main steps of the software.

- Calculate the 3D location of the physical landmarks relative to the present frame through the *triangulation* step, which is possible knowing the relative position between the cameras.

- Perform the motion estimation step (see chapter 3).

These steps are performed for each pair of consecutive stereo frames. The overall motion estimate is determined as the combination of motions from each pair of frames.

There are three distinct approaches to tackle the problem of motion estimation, depending on whether the correspondences are specified in three or two dimensions [10]:

1. 3D-to-3D correspondences: we have 3D locations of N corresponding features (points or lines) at two different times.

2. 2D-to-3D correspondences: we have correspondence of N features (f_i, f_i') such that f_i are specified in three dimensions and f_i' are their projection on the 2D

image plane.

3. 2D-to-2D correspondences: N corresponding features are specified on the 2D image plane either at two different times for a single camera or at the same instant of time but for two different cameras.

The second approach is that proposed by Nister et al. [1]. They computed the relative motion as a 3D-to-2D camera-pose estimation, incorporating RANSAC outlier rejection into the motion estimation.

The feature matching stage inevitably produces some incorrect correspondences (i.e., wrong data associations named *outliers*), which will bias the frame-to-frame motion estimate. A common solution to this problem is to use a robust estimator that can tolerate some number of false matches. Therefore in the motion estimation step we used RANSAC (see section 3.3) and P3P motion estimation method on three points (see section 3.2) using 2D-3D correspondences to exclude outliers from the uncertain matches and obtain a consensus estimation of motion. A refinement via non-linear optimization is then performed to polish the solution.

2.2 Camera model

The mathematical camera model which is used in the following to describe the two cameras is the pinhole camera model (see figure 2.3). It describes the central projection of 3D points through the center of projection onto the image plane. A detailed discussion about this model can be found in [11].

Optical center of a pinhole camera coincides with the origin of a cartesian coordinate system (O, x, y, z) and the positive z-axis is the direction of view. The image plane is located at a distance equal to the focal length f from O along the direction of view.

Let p be a generic scene point, with coordinates $\boldsymbol{X}_0 = \begin{bmatrix} X_0 & Y_0 & Z_0 \end{bmatrix}^T$ relative to the world reference frame. The coordinates ${}^i\boldsymbol{X} = {}^i\begin{bmatrix} X & Y & Z \end{bmatrix}^T$ of the same point p relative to the camera frame i are given by a rigid-body transformation of

$\boldsymbol{X}_0$. In homogeneous coordinates

$$ {}^{i}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} R_0^i & t_0^i \\ 0\,0\,0 & 1 \end{bmatrix} \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{bmatrix} . \tag{2.2} $$

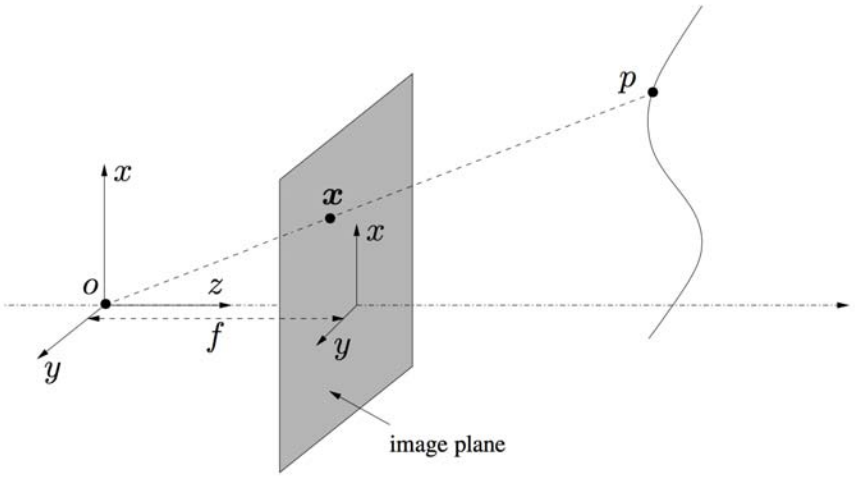


Figure 2.3: Frontal pinhole imaging model: 3D point p is projected at a 2D point on the image plane as the point x at the intersection of the ray going through the optical center O and the image plane at a distance f in front of the optical center [11].

Adopting the frontal pinhole camera model the point ${}^{i}\boldsymbol{X}$ is projected onto the image plane at the point

$$ \lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \end{bmatrix} {}^{i}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} {}^{i}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} , \tag{2.3} $$

where ${}^{i}\boldsymbol{X} \doteq {}^{i}\begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$ and $\boldsymbol{x}_i \doteq \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^T$ are in homogeneous representation. The index i is 1 or 2 depending on which camera is considered: ${}^{i}\boldsymbol{X}$ is the point position expressed in frame i associated with camera i, while $\boldsymbol{x}_i$ is the

projection of the point ${}^{i}\boldsymbol{X}$ with an ideal camera aligned like camera i with a focal length f. $\lambda \in \mathbb{R}_+$ is an arbitrary positive scalar associated with the depth of the point.

To summarize, the mapping from the 3D world to the 2D image (in length unit) is given by

$$\lambda \boldsymbol{x}_i = K_f \Pi_0 {}^{i}\boldsymbol{X} = K_f \Pi_0 g \boldsymbol{X}_0 \,. \tag{2.4}$$

The matrix $\Pi_0 \in \mathbb{R}^{3\times 4}$ is often referred to as the *standard projection matrix*.

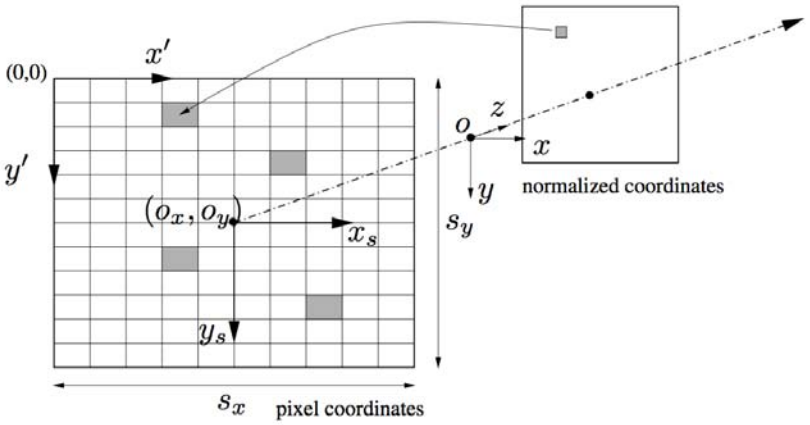


Figure 2.4: Transformation from normalized coordinates to coordinates in pixels [11].

In order to render this model usable, we need to specify the relationship between the retinal plane coordinate frame (centered at the optical center with one axis aligned with the optical axis) and the pixel array. Referring to figure 2.4, we obtain the following relation

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & S_\theta & O_x \\ 0 & S_y & O_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} , \tag{2.5}$$

where x_i' and y_i' are actual image coordinates in pixels, S_x and S_y represent the pixels density along the x and y directions and (O_x, O_y) are the coordinate in pixels of the *principal point* (where the z-axis intersects the image plane) relative to the

image reference frame. S_θ is called *skew factor* and it is introduces in the case that pixels are not rectangular. It is proportional to $\cot\theta$, where θ is the angle between the image axes x_s and y_s. In many practical applications it is common to assume that $S_\theta = 0$.

In matrix form

$$\boldsymbol{x}'_i = K_s \boldsymbol{x}_i \,. \tag{2.6}$$

Now, combining the projection model with the scaling and translation yields a more realistic model of a transformation between homogeneous coordinates of a 3D point relative to the camera frame and homogeneous coordinate of its image expressed in terms of pixels,

$$\lambda \boldsymbol{x}'_i = K_s K_f \Pi_0 {}^i\boldsymbol{X} \,. \tag{2.7}$$

The constant 3×4 matrix Π_0 represent the perspective projection.

To summarize, the overall model for image formation is therefore captured by the equation

$$\lambda \boldsymbol{x}'_i = K \Pi_0 {}^i\boldsymbol{X} = K \Pi_0 g \boldsymbol{X}_0 \,. \tag{2.8}$$

The 3×3 matrix K collects all parameters that are "intrinsic" to a particular camera and is therefore called the *intrinsic parameter matrix*, or the *calibration matrix* of the camera,

$$K \doteq \begin{bmatrix} fS_x & fS_\theta & O_x \\ 0 & fS_y & O_y \\ 0 & 0 & 1 \end{bmatrix} \,. \tag{2.9}$$

The camera matrix is then a 3×4 matrix defined as

$$CM = K\Pi_0 = \begin{bmatrix} fS_x & fS_\theta & O_x & 0 \\ 0 & fS_y & O_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \,. \tag{2.10}$$

The entries of the matrix K have the following geometric interpretation:

- O_x: x-coordinate of the principal point in pixels,

- O_y: y-coordinate of the principal point in pixels,
- $fS_x = \alpha_x$: size of unit length in horizontal pixels,
- $fS_y = \alpha_y$: size of unit length in vertical pixels,
- α_x/α_y: aspect ratio σ,
- fS_θ: skew of the pixel, often close to zero.

When the calibration matrix K is known, the *calibrated* coordinates $\boldsymbol{x}_i$ can be obtained from the pixel coordinates $\boldsymbol{x}'_i$ by a simple inversion of K,

$$\lambda \boldsymbol{x}_i = \lambda K^{-1} \boldsymbol{x}'_i \,. \tag{2.11}$$

A stereo system is also characterized by its *extrinsic parameters* that describe the mutual position and orientation between the two cameras. The intrinsic (i.e., matrix K) and extrinsic parameters can be obtained through the process of camera calibration.

2.2.1 Camera calibration

The goal of calibration is to accurately measure the intrinsic and extrinsic parameters of the camera system.

The most popular method uses a planar checkboard-like pattern. Taking several pictures of the board shown at different positions and orientation, the intrinsic and extrinsic parameters are found through a least-square minimization method. The input data are the 2D positions of the corner of the squares on the board and their corresponding pixel coordinates in each image.

A C implementation of camera calibration for perspective cameras can be found in OpenCV [12].

2.2.2 Lens distortion

Since no lens is perfect, in addition to linear distortion (described by the parameters in K) we must consider at least other two types of distortion.

Radial distortion : arises as a result of the shape of lens; this distortion is due to manufacturing defects resulting from the lens not being exactly parallel to the imaging plane.

Tangential distortion : arises from the assembly process of the camera as a whole.

Concerning the radial distortion, the lenses of real cameras often noticeably distort the location of pixels near the edges of the image. This bulging phenomenon is the source of the "barrel" or "fish-eye" effect. This type of distortion is 0 at the optical center of the image and increases as we move toward the periphery. In practice, this distortion is small and can be characterized by the first few terms (generally k_1, k_2 and k_3) of a Taylor series expansion around $r = 0$. In general, the radial location of a point on the image will be rescaled according to the following equations

$$\begin{aligned} x_{corrected} &= x\left[1 + k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)^2 + k_3\left(x^2 + y^2\right)^3\right] \\ y_{corrected} &= y\left[1 + k_1\left(x^2 + y^2\right) + k_2\left(x^2 + y^2\right)^2 + k_3\left(x^2 + y^2\right)^3\right] \end{aligned}, \tag{2.12}$$

where (x, y) is the original location on the image of the distorted point and $(x_{corrected}, y_{corrected})$ is the new location as a result of the correction.

Tangential distortion is minimally characterized by two additional parameters, p_1 and p_2, such that

$$\begin{aligned} x_{corrected} &= x + \left[2p_1 y + p_2\left(3x^2 + y^2\right)\right] \\ y_{corrected} &= y + \left[p_1\left(x^2 + 3y^2\right) + 2p_2 x\right] \end{aligned}. \tag{2.13}$$

Thus there are five distortion coefficients (typically bundled into one distortion vector $[k_1, k_2, p_1, p_2, k_3]$) required overall.

2.3 Feature detection and matching

Once the intrinsic and extrinsic parameters of a stereo system are determined, two steps must be considered.

- The first step is to detect 2D keypoints in both frames with an interest point detector.

- In the second step, a feature descriptor is used to represent the detected keypoints in a distinctive way. Correspondences are then obtained by using a matching strategy that compares feature descriptors based on a similarity measurement.

Substantially detectors find out regions that are projection of landmarks and can be used as features (i.e., an image pattern that differs from its immediate neighbourhood in terms of intensity, color, and texture), while descriptors provide representation of the detected region. These description allow to search similar regions between image and to perform their matching.

The accuracy and robustness of the feature detector and matching algorithm have a direct impact on the accuracy of the camera motion prediction.

2.3.1 Feature detection

During the feature-detection step, salient keypoints that match well in other images, are searched in the image. Many features can be considered, e.g. points, straight lines, curved lines, and corners or blobs (i.e., an image pattern different from its immediate neighbourhood).

The appealing properties that a good feature detector should have are [13]: localization accuracy (both in position and scale), repeatability (i.e., a large number of features should be re-detected in the next images), computational efficiency, robustness (to noise, compression artifacts, blur), distinctiveness (so that features can be accurately matched across different images), and invariance to both photometric (e.g., illumination) and geometric changes (rotation, scale, perspective distortion).

Every feature detector consists in two stages. The fist is to apply a feature-response function on the entire image (such as the difference-of-Gaussian operator of the SIFT). The second step is to apply nonmaxima suppression on the output of the fist step to identify all local minima (or maxima) of the feature-response function. The output of the second step represents detected features.

The literature is characterized by many feature detectors, with their pros and cons. An overview of these detectors can be found in [13], [14], [15] and [16]. An approximate comparison is given in table 2.1 and in figure 2.5.

The choice of the appropriate feature detector should be carefully considered, depending on the environment type, computational constraints, real-time requirements and how nearby images are taken.

	Scale invariance	Feature type
Harris	✗	Corner
FAST	✗	Corner
Agast	✗	Corner
BRISK(O)	✗	Corner
BRISK	✓	Corner
SIFT	✓	Blob
SURF	✓	Blob
CensurE	✓	Blob

Table 2.1: Summarize of the detectors [16].

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

Figure 2.5: Comparison of feature detectors: properties and performance [13].

2.3.2 Feature description

After features are detected, their regions have to be represented by a descriptor in order to compare them. In the feature description step, the region around each detected feature is converted into a compact descriptor that can be matched against

other descriptors. The simplest descriptor of a feature is its local appearance (e.g., the intensity of the pixel in a patch around the feature point), but in many case is not a good information because its appearance will change with orientation, scale and viewpoint changes.

	Rotation invariance	Descriptor type
SAD	✗	Image patch pixels
NCC	✗	Image patch pixels
SIFT	✓	Gradient histogram
SURF	✓	Gradient histogram
U-SURF	✗	Gradient histogram
BRIEF	✗	Brightness comparison
BRISK	✓	Brightness comparison
U-BRISK	✗	Brightness comparison

Table 2.2: Summarize of the descriptors [16].

Figure 2.5 presents a comparison of feature detectors. Different combination of detectors and descriptors have been used for stereo VO: [13] and [16] for example describe the state of the art feature detectors and descriptors in VO field and evaluate which one may be the most appropriate solution. However, the selection of an optimal procedure remains difficult, since the results substantially depend on the implementation.

Sections 2.3.3, 2.3.4 and 2.3.5 briefly describe some of the popular detectors and descriptors. Refer to the original papers for more details.

2.3.3 SIFT

Lowe proposed a "Scale Invariant Feature Transform" (SIFT) detector/descriptor scheme [17]. The SIFT features are invariant to image scale and rotation and are also robust to changes in illumination, noise and minor changes in viewpoint.

SIFT algorithm uses a Difference of Gaussians (DoG) detector, which is an approximation of Laplacian of Gaussian. DoG is obtained as the difference of Gaussian blurring of an image with two different σ. In summary SIFT searches keypoints at multiple scales by constructing a "Gaussian scale space", that is a collection of images obtained by progressively smoothing the input image with Gaussian filters

with different σ (see figure 2.6). The smoothing level is called scale of the image: for example, increasing the scale by an octave means doubling the size of the smoothing kernel. This is repeated after down-sampling the image of a factor two. DoG images are then computed by taking the difference between successive Gaussian-smoothed images.

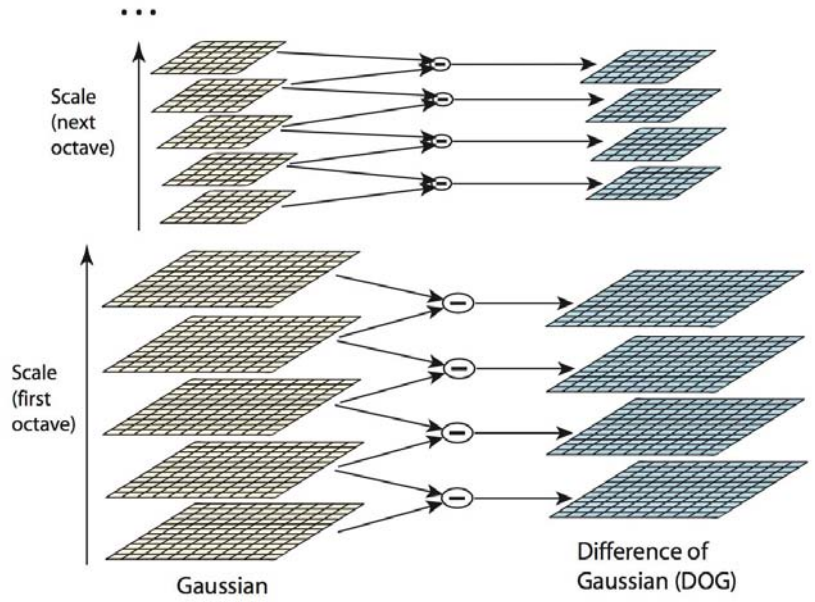


Figure 2.6: An illustration of the approach to construction of DoG. For each octave of scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated. [17]

SIFT features are found as local minima or maxima of DoG images across scales and space. For example, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales (see figure 2.7). If it is a local extrema, it is a potential keypoint. In this way we obtain a list of (x, y, σ) values which means there is a potential keypoint at (x, y) at σ scale.

Regarding different parameters, the paper [17] gives some empirical data which can be summarized as: number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, $k = \sqrt{2}$ etc. as optimal values.

The potential keypoints locations have to be refined to get more accurate results. We have to eliminate those that are likely to be unstable, either because they

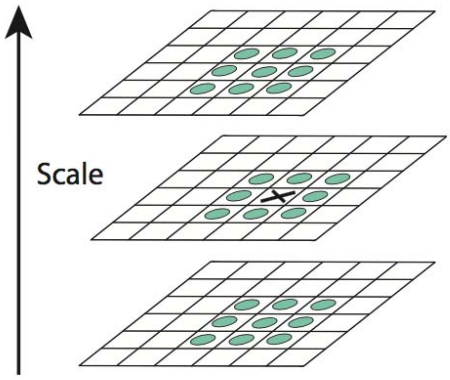


Figure 2.7: Maxima and minima of the Difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3×3 regions at the current and adjacent scales (marked with circles). [17]

are selected nearby an image edge, rather than an image blob, or are found on image structures with low contrast. Filtering is then controlled by two parameters:

- the minimum amount of contrast to accept a keypoint (this threshold value is set to 0.03 in the paper);
- the edge rejection threshold (set to 10 in the paper).

In this way any low-contrast keypoints and edge keypoints are eliminated and what remains is strong interest points.

The keypoint descriptor is then created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location. To do that a 16×16 neighbourhood around the keypoint is taken. These samples are also weighed by the Gaussian window to give less importance to gradients farther away from the keypoint center. Each 16×16 block is then divided into sub-blocks of 4×4 size and for each sub-block, 8 bin orientation histogram summarizing the subregion contents is created (see figure 2.8). The length of each arrow corresponds to the sum of the gradient magnitudes near that direction within the region. In this way a total of 128 bin values are available. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation, etc.

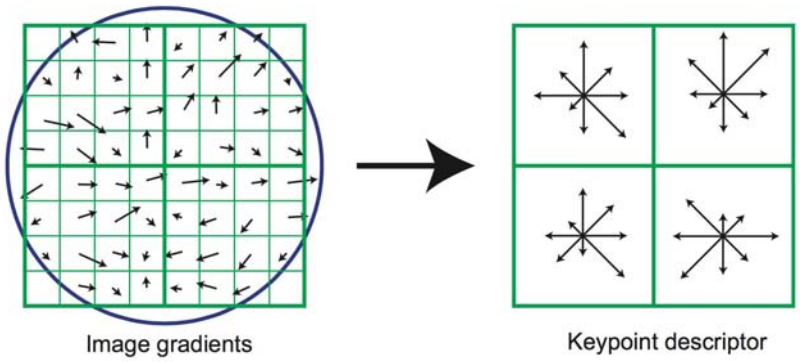


Figure 2.8: Schematic illustration of SIFT descriptor. The patch around the feature is decomposed into a 4×4 grid and for each quadrant a histogram of eight gradient orientation is built. All these histograms are then concatenated together forming a 128-element descriptor vector. Note that this figure shows a 2×2 descriptor array computed from an 8×8 set of samples, whereas the experiments in [17] use 4×4 descriptors computed from a 16×16 sample array. [17]

2.3.4 SURF

The SIFT algorithm described in the section above was comparatively slow and people needed more speeded-up version. In 2006 Bay et al. proposed a "Speeded Up Robust Features" (SURF) detector/descriptor scheme [18] that, as name suggests, is a speeded-up version of SIFT.

Unlike SIFT, SURF approximates LoG with box filters. These approximate second order Gaussian derivatives and can be evaluated very fast using integral images, independently of size. The 9×9 box filters in figure 2.9 are approximations for Gaussian second order derivatives with $\sigma = 1.2$ and represent the lowest scale (i.e., highest spatial resolution).

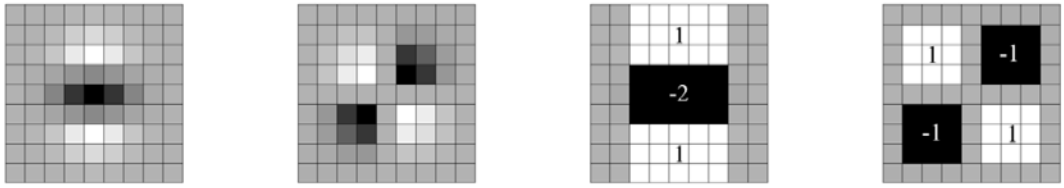


Figure 2.9: Left to right: the (discretized and cropped) Gaussian second order partial derivatives in y-direction and xy-direction, and the approximations thereof using box filters. The grey regions are equal to zero. [18]

Also the SURF rely on determinant of Hessian matrix for both scale and location. Like SIFT, the interest points can be found in different scales. However in this case scale spaces are implemented by applying box filters of different sizes. Therefore, the scale space is analysed by up-scaling the filter size rather than iteratively reducing the image size. The output of the 9×9 filter is considered as the initial scale layer (i.e., scale $s = 1.2$, corresponding to Gaussian derivatives with $\sigma = 1.2$). The following layers are obtained by filtering the image with gradually bigger masks. Specifically, this results in filters of size 9×9, 15×15, 21×21, 27×27, etc. In order to localize interest points in the image and over scales, non-maximum suppression in a $3 \times 3 \times 3$ neighbourhood is applied. The maxima of the determinant of the Hessian matrix are then interpolated in scale and image space.

Regarding SURF descriptor, the first step consists of fixing a reproducible orientation based on information from a circular region around the interest point. Then, we construct a square region aligned to the selected orientation, and extract the SURF descriptor from it.

In order to achieve rotational invariance, the orientation of the point of interest needs to be found. For orientation assignment, SURF uses the Haar-wavelet responses both in x and y directions within a circular neighbourhood of radius 6s around the point of interest, where s is the scale at which the point of interest was detected. Adequate Gaussian weights are also applied to it. The obtained responses are then plotted as points in a two-dimensional space, with the horizontal response in the abscissa and the vertical response in the ordinate. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. The horizontal and vertical responses within the window are summed. The two summed responses then yield a local orientation vector. The longest such vector overall defines the orientation of the point of interest. Note that wavelet response can be found out using integral images very easily at any scale.

For many applications, rotation invariance is not required, so no need of finding this orientation, which speeds up the process. SURF provides such a functionality called Upright-SURF or U-SURF.

For feature description, SURF uses Wavelet responses in horizontal and vertical

direction (again, use of integral images makes things easier). A square window of size $20s \times 20s$ is extracted around the interest point and oriented along the orientation as selected above (examples of such square regions are illustrated in figure). This regions is then split up into smaller 4×4 subregions. For each subregion, horizontal and vertical Haar-wavelet responses, called respectively d_x and d_y, are taken and a vector is formed like this, $v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$. This results in a descriptor vector for all 4×4 sub-regions of length 64. Lower the dimension, higher the speed of computation and matching, but provide better distinctiveness of features.

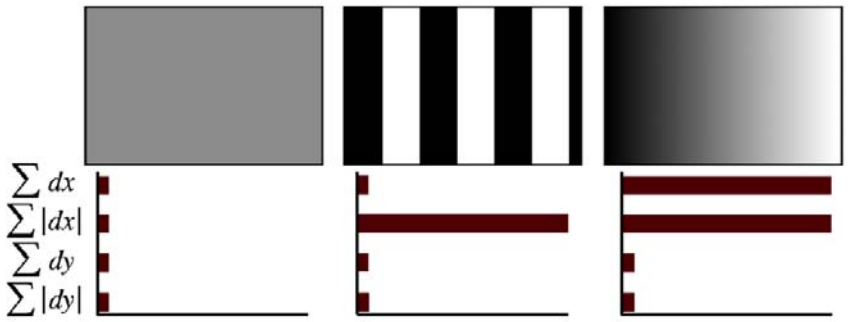


Figure 2.10: Example of SURF square regions. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of $\sum |dx|$ is high, but all others remain low. Right: If the intensity is gradually increasing in x direction, both values of $\sum dx$ and $\sum |dx|$ are high.[18]

In short, SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

2.3.5 FAST

There exist several feature detectors and many of them are really good. But when looking from a real-time application point of view, they are not fast enough. As a solution to this, "Features from Accelerated Segment Test" (FAST) algorithm was proposed by Rosten and Drummond in 2006 [19].

Briefly, after selecting a pixel p of intensity I_p, a circle of 16 pixels is considered around it (see figure 2.11). The pixel p is a corner if there exists a set of n contiguous pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$. In figure 2.11 n was chosen to be 12: the dash lines shown 12 contiguous pixels which are brighter than p by more than the threshold.

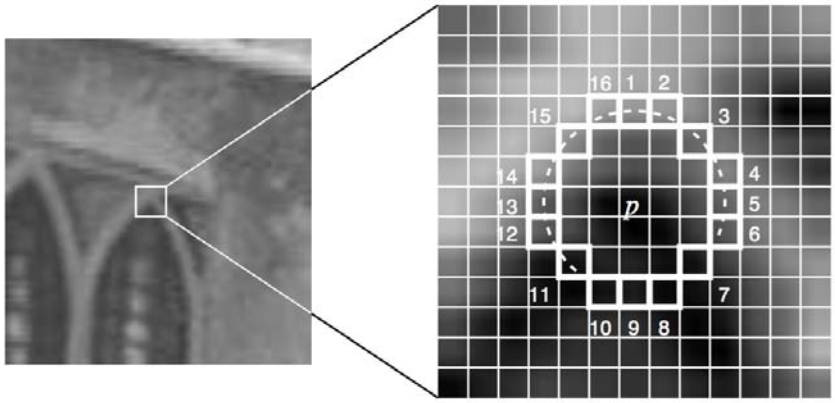


Figure 2.11: 12 point segment test corner detection in an image patch. The pixel at p is the centre of a candidate corner. The highlighted squares are the pixels used in the corner detection. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold.[19]

A high-speed test was proposed to exclude a large number of non-corners. This test examines only the four pixels at 1, 9, 5 and 13. If p is a corner, then at least three of these must all be brighter than $I_p + t$ or darker than $I_p - t$. If neither of these is the case, then p cannot be a corner. The full segment test criterion can then be applied to the passed candidates by examining all pixels in the circle.

FAST detector is several times faster than other existing corner detectors, but it is not robust to high levels of noise.

Note that FAST is only a feature detector and doesn't provide any method to describe the features. We can use any other feature descriptors, such as BRIEF, "Binary Robust Independent Elementary Features" [20], that is a faster method feature descriptor calculation proposed by Calonder et al. [20]. It uses binary strings such as feature point descriptor. It is highly discriminative even when only using few bits and can be computed using simple intensity difference tests. The descriptor similarity can be evaluated using the Hamming distance, which is very

efficient to compute, instead of the L2 norm distance. So, BRIEF is very fast both to build and match.

2.4 Triangulation

Once that both cameras of the stereo system are calibrated, the 3D position of a feature point in space may be obtained by triangulation of the image points, for example using the algorithm of the middle point (see figure 2.12).

Triangulated 3D points are determined by intersecting back-projected rays from 2D image correspondences of two image frames. In perfect conditions, these rays would intersect in a single 3D point. However, because of image noise, camera model and calibration errors, and feature matching uncertainty, they never intersect. Therefore, the point at a minimal distance from all intersecting rays can be taken as an estimate of the 3D point position.

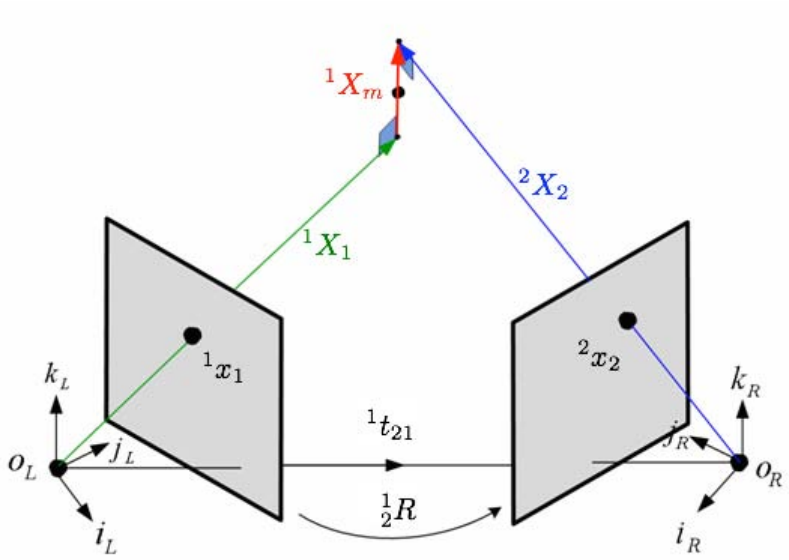


Figure 2.12: An illustration of the midpoint triangulation problem. For each feature $\boldsymbol{x}_i$ the algorithm finds 3D points $\boldsymbol{X}_{1,s}$ and $\boldsymbol{X}_{2,s}$, with the minimum distance, belonging respectively to the preimage lines (i.e. the lines that starts from the optical center of each camera and project the 2D features from the image plane to the 3D observed scene) of camera 1 and 2. Points $\boldsymbol{X}_{1,s}$ and $\boldsymbol{X}_{2,s}$ define a segment orthogonal to the two skew preimage lines. Middle point X of this segment is defined as the measured 3D point of the feature. See [21] for details.

Chapter 3

Motion estimation

3.1 Introduction

Motion estimation is the core computation step performed for every image in VO system. More precisely, in this step the camera motion between the current image pair and the previous one is computed.

As mentioned in section 2.1, depending on whether the features correspondences are specified in two or three dimensione, there are three different methods.

Nister [1] compared the VO performance of the 3D-to-3D case to that of the 3D-to-2D one for a stereo camera system and found the latter being greatly superior to the former. In his opinion the 3D-to-2D method is preferable compared to the 2D-to-2D case too, since the lower point correspondences needed results in a much faster outlier rejection and then motion estimation.

Following the Nister et al. approach, in the present work 3D-to-2D motion estimation is considered.

The general formulation is to find T_k^{k-1} that minimizes the image reprojection error

$$\arg\min_{T_k^{k-1}} \sum_i \left\| \boldsymbol{x}_k^i - \hat{\boldsymbol{x}}_{k-1}^i \right\|^2, \tag{3.1}$$

where $\boldsymbol{x}_k^i$ is the 2D point and $\hat{\boldsymbol{x}}_{k-1}^i$ is the reprojection of the 3D point $\boldsymbol{X}_{k-1}^i$ into image I_k according to the transformation T_k^{k-1}.

This problem is known as "Perspective from n Points" (PnP) and there are many different solutions to it in the literature. The minimal case involves three 3D-to-2D correspondences and this is called "perspective from three points" (P3P, see section 3.2). In the 3D-to-2D case, P3P is the standard method for robust estimation in the presence of outliers.

3.2 The P3P problem

The Perspective-n-Point (PnP) problem, also known as pose estimation, aims at retrieving the position and orientation of the camera with respect to a scene object from n corresponding 3D points.

Fischler and Bolles [22] summarized the problem as follows (see figure 3.1):

> "Given the relative spatial locations of n control points, and given the angle to every pair of control points P_i from an additional point called the center of perspective C, find the lengths of the line segments joining C to each of the control points."

The next step then consists of retrieving the orientation and translation of the camera with respect to the object reference frame.

We are interested in the particular case of PnP for $n = 3$, that is known as "Perspective-Three-Point" (P3P) problem. The P3P is the smallest subset of control points that yields a finite number of solution. Given the intrinsic camera parameters and $n \geq 4$ points, the solution is generally unique.

The P3P is then the most basic case of the PnP problem. It aims at determining the position and orientation of the camera in the world reference frame from three 2D-3D point correspondences. Using these three correspondences we can find up to four possible pose configurations, each consisting of a rotation matrix and a translation vector, that can then be disambiguated using a fourth point.

Referring to figure 3.2, the P3P principle can be summarized with the following steps.

- Four 2D-3D correspondences are given ($A \leftrightarrow a$, $B \leftrightarrow b$, $C \leftrightarrow c$, $D \leftrightarrow d$): 3D points are defined in the world coordinate system, while 2D points are defined in the image coordinate system.

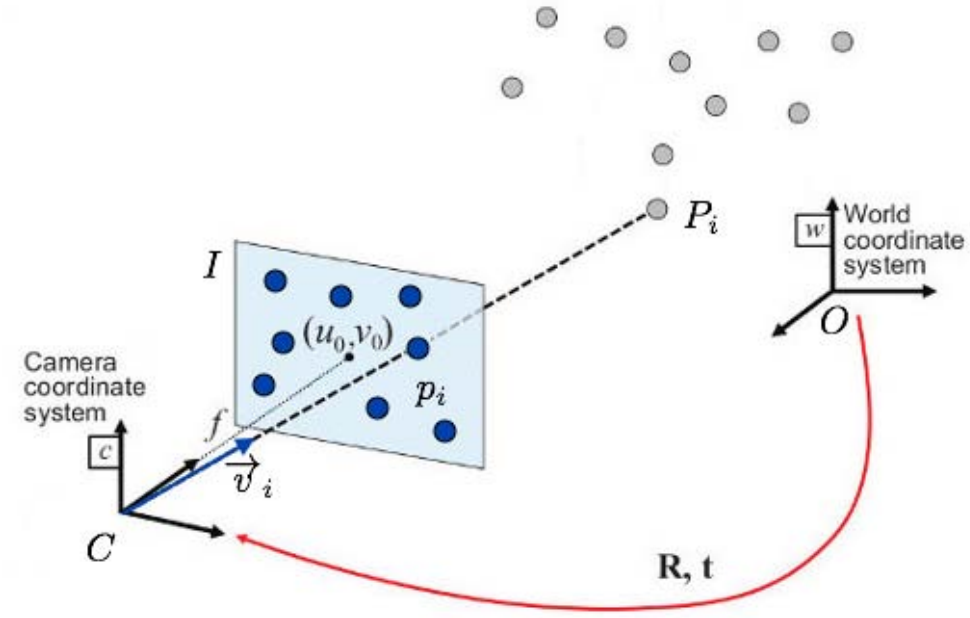


Figure 3.1: PnP problem representation [12]. Given a set of n 3D points P_i whose coordinates are known in some object coordinate frame O, let p_i be a set of n 2D points which are the projection of P_i on the image plane I. Let $\overrightarrow{v}_i = \overrightarrow{Cp_i}$ be n directional vectors with C as the camera center of perspective (note that since the camera is assumed to be calibrated, one can determine the vectors $\overrightarrow{v}_i$ from the camera calibration matrix K). The PnP problem is defined as to determine the position of C and its orientation relative to O.

- The solution of the P3P equation system with three points (A, B, C) leads to four possible set of distance $\|PA\|$, $\|PB\|$ and $\|PC\|$, where P is the camera optical center.

- These sets are then converted into four pose configurations.

- The fourth point (D) is then used to select the best pose configuration among the four proposed.

In the literature, there exist many solutions to this problem, which can be classified into iterative, non-iterative, linear and non-linear ones. For example, see [23], [24], [25], [26] and [27].

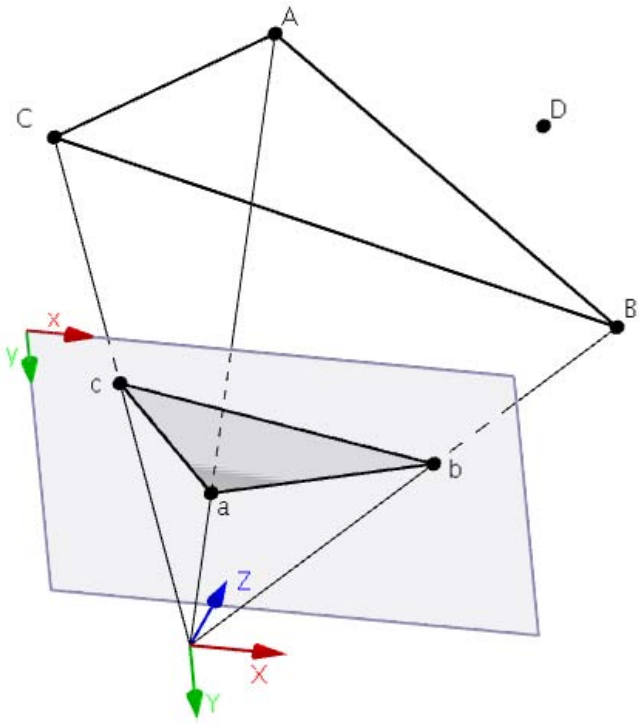


Figure 3.2: 3D points (A, B, C) projected in 2D (u, v, w).

OpenGV [28] consider different solution to the P3P problem proposed by different authors.

Gao et al. This solution represent the typical non-iterative algorithm that involves solving for the roots of an eight-degree polynomial with no odd terms, yielding up to four solutions, so that a fourth point is needed for disambiguation. In their paper, Gao et al. proposed two approaches to solve the P3P problem, one algebraic and one geometric.

Kneip et al. In contrast to all previous approaches, they propose a novel closed-form solution to the P3P problem, which computes the aligning transformation directly in a single stage, without the intermediate derivation of the points in the camera frame. The proposed solution computes directly the position and orientation of the camera in the world reference frame as a function of the image coordinates and the coordinates of the reference points in the world frame. In their paper Kneip et al. stated that the proposed algorithm offers

accuracy and precision comparable to a popular, standard, state-of-the-art approach but at much lower computational cost. The superior computational efficiently is particularly suitable for any RANSAC-outlier rejection step.

EPnP. It stand for "Efficient Perspective-n-Point" Camera Pose Estimation. Lepetit et al. proposed this non-iterative solution to the PnP problem that, according to their paper, has better accuracy and much lower computational complexity than non-iterative state-of-the-art methods, and much faster than iterative ones with little loss of accuracy. This technique allows the computation of an accurate and unique solution in $O(n)$ for $n \geq 4$. As in most of the existing PnP solution techniques, the idea in the implementation of EPnP is to retrieve the locations of P_i relative to the camera coordinate frame. Then one can retrieve the camera orientation and translation which aligns both sets of coordinates. The algorithm has an efficient implementation because it represent the P_i as a weighted sum of $m \leq 4$ control points $C_1, ..., C_m$ and perform all further computation only on these points. For large n this yields a much smaller number of unknowns compared to other algorithms and therefore accelerates further computations.

The derivations that lead to these solutions of the P3P algorithm are presented in [29], [30] and [31] respectively.

3.3 Outliers rejection

In VO real-time application we deal with hundreds or even thousands of noisy feature points and outliers, which requires computationally efficient methods. There exists several methods to reject outliers in the motion estimation process.

The standard approach consist of first using P3P in a RANSAC scheme [22] - in order to remove outliers - and then PnP on all remaining inliers.

In the following section we will see in detail the RANSAC approach.

3.3.1 RANSAC algorithm

The RANSAC algorithm (abbreviation for "RANdom SAmple And Consensus") is the standard method for model estimation starting from a set of data contaminated by large amounts of outliers. A datum is considered to be an outlier if it will not fit the "true" model instantiated by the "true" set of parameters within some error threshold that defines the maximum deviation attributable to the effect of noise. The percentage of outliers which can be handled by RANSAC can be larger than 50% of the entire dataset. It is a non-deterministic algorithm that produces a reasonable result only with a certain probability which increase as more iterations are allowed.

The RANSAC algorithm, first published by Fischler and Bolles in 1981 [22], is formally stated as follow:

> "Given a model that requires a minimum of n data points to instantiate its free parameters, and a set of data points P such that the number of points in P is greater than n [$\#(P) \geq n$], randomly select a subset $S1$ of n data points from P and instantiate the model. Use the instantiated model $M1$ to determine the subset $S1^*$ of points in P that are within some error tolerance of $M1$. The set $S1^*$ is called the *consensus set* of $S1$.
> If $\#(S1^*)$ is greater than some threshold t, which is a function of the estimate of the number of gross errors in P, use $S1^*$ to compute (possible using least squares) a new model $M1^*$.
> If $\#(S1^*)$ is less than t, randomly select a new subset $S2$ and repeat the above process. If, after some predetermined number of trial, no consensus set with t or more members has been found, either solve the model with the largest consensus set found, or terminate in failure."

The RANSAC paradigm contains three unspecified parameters: (a) the error tolerance used to determine whether or not a point is compatible with a model, (b) the number of subsets to try and (c) the threshold t, which is a number of compatible points used to imply that the correct model has been found.

Let us suppose that we have a dataset that contain both *inliers*, i.e. data whose distribution can be explained by some set of model parameters, and *outliers*, which are data that do not fit the model instantiated by the true set of parameters (within some error threshold that defines the maximum deviation attributable to the effects of noise). RANSAC assumes that:

- given a set of inliers, there exists a model and a set of parameters that optimally fits the observed data in absence of noise;
- we know what is the maximum perturbation of an observed valid measurement.

It is therefore a learning technique to estimate parameters of a mathematical model by random sampling of observed data.

The RANSAC algorithm is essentially composed of two steps iteratively repeated (*hypothesize–and–test framework*).

Hypothesize. A sample of subset containing minimal data items is randomly selected from the input dataset. A fitting model and the corresponding model parameters are then computed using only the elements of this sample subset. The cardinality of the minimal sample set is the smallest sufficient to determine the model parameters (as opposed to other approaches, such as *least squares*).

Test. The algorithm checks which elements of the entire dataset are consistent with the model instantiated by the estimated model parameters obtained from the first step. A data element will be considered as an outlier if it does not fit the model instantiated by the set of estimated model parameters within some error threshold. The set of inliers obtained for the fitting model is called *consensus set*.

The algorithm will iteratively repeat the above two steps until the obtained consensus set has enough inliers.

An advantage of RANSAC is that it allows to do robust estimation of the model parameters evaluating them with a high degree of accuracy even when the dataset contains a significant number of outliers. A disadvantage is that there is no upper bound on the time it takes to compute these parameters (except exhaustion). If we limited the number of iterations, the obtained solution may not be optimal and may not fit the data in a good way. In this way RANSAC offers a trade-off: in fact the probability of a reasonable model being produced increases by computing a greater number of iterations. Furthermore RANSAC is not always able to find the optimal set

and it usually performs badly when the number of inliers is less than 50%. Another disadvantage is that it requires the setting of problem-specific thresholds.

Note that RANSAC draws the elements composing the minimal sample sat randomly from the entire dataset. Therefore at each run the behaviour might change.

As pointed out by Torr et al. [32], RANSAC can be sensitive to the choice of the correct noise threshold that defines which data points fit a model instantiated with a certain set of parameters. If such threshold is too large, then all the hypotheses tend to be ranked equally. On the other hand, when the noise threshold is too small, the estimated parameters tend to be unstable.

The identification of outlying point correspondences is very important in order to receive a reliable camera pose.

The number of iteration needed is another fundamental parameter. Let n is the number of samples needed for instantiating a hypothesis and let p be the probability of sampling in some iteration a set of n inliers from the dataset. Let w denote the probability of one sample being an inlier, that is

$$w = \frac{\text{number of inlier points}}{\text{total number of points}} = \frac{y}{z} \tag{3.2}$$

We don't know beforehand w, but we can give some rough value. Selecting n samples, w^n is the probability of all points being inliers, and therefore $1 - w^n$ is the probability that at least one of the points is an outlier (which implies that a bad model will be estimated from this point set). After k iterations, the probability that in each iteration at least one outlier has been sampled becomes $(1 - w^n)^k$ and must be the same as $1 - p$. This finally leads to the well-known formula introduced by Fischler and Bolles [22] for computing the number of iterations required to satisfy a given p,

$$k = \frac{\log(1-p)}{\log(1-w^n)}\,. \tag{3.3}$$

Chapter 4

Procedure

Our goal was to implement a software that estimates the 6D pose of an agent giving as input some pairs of images captured by its on-board cameras. The procedure is schematically represented in figure 4.1.

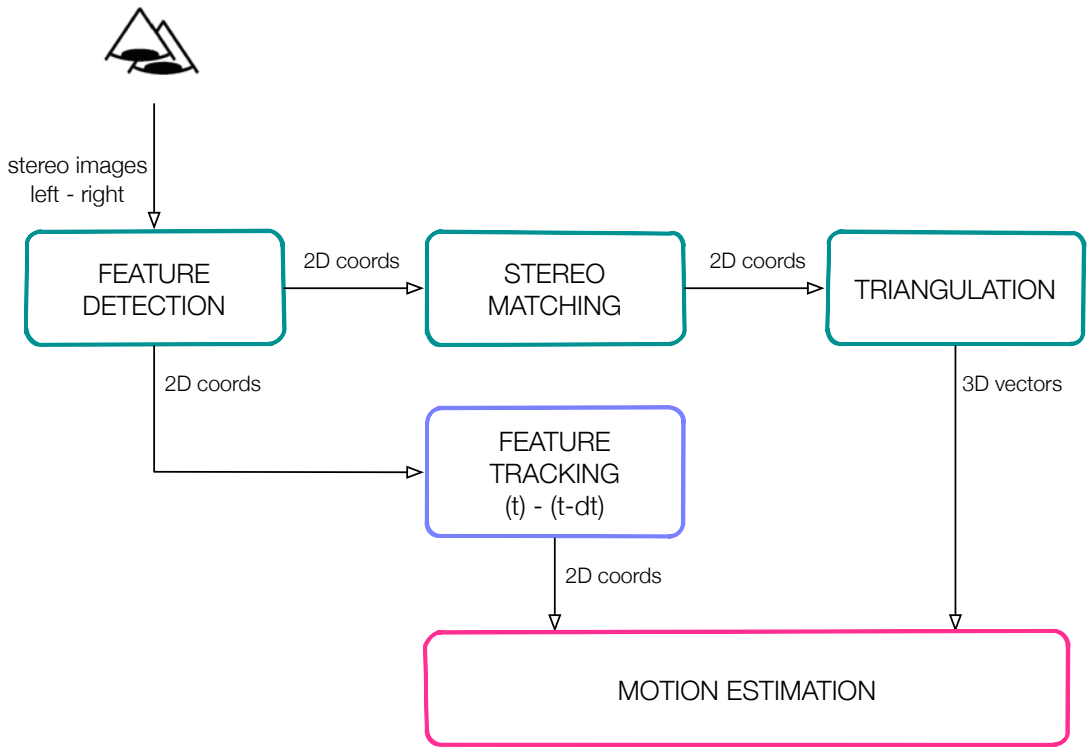


Figure 4.1: Schematic illustration of the steps performed by the software.

Note that once we have the camera parameters (tables 5.1 and 5.2) we can

define the two camera matrices, CM_1 and CM_2. As mentioned in chapter 2.2, a camera matrix is a 3×4 matrix which describes the mapping of a pinhole camera from 3D points in the world to 2D points in an image as

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} fS_x & fS_\theta & O_x & 0 \\ 0 & fS_y & O_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} {}^{i}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} . \tag{4.1}$$

Because the 3D points are expressed in the left camera reference frame, for the left camera we have

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = \begin{bmatrix} fS_{x1} & fS_{\theta 1} & O_{x1} & 0 \\ 0 & fS_{y_1} & O_{y_1} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} {}^{1}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} , \tag{4.2}$$

while for the right camera

$$\begin{bmatrix} x'_2 \\ y'_2 \\ 1 \end{bmatrix} = \begin{bmatrix} fS_{x2} & fS_{\theta 2} & O_{x2} & 0 \\ 0 & fS_{y_2} & O_{y_2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_2^1 & t_2^1 \\ 0\,0\,0 & 1 \end{bmatrix} {}^{1}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} , \tag{4.3}$$

where $[R|t]_2^1$ is the transformation matrix between right and left camera reference frames.

4.1 Image processing

Fist of all the software reads a pair of stereo images and correct it. The dataset images are given in a GBRG Bayer pattern format. The Bayer array is a color filter array for arranging RGB color filters on a square grid of photo-sensors. It consists of alternating green (G) and blue (B) filters for odd rows and alternating red (R)

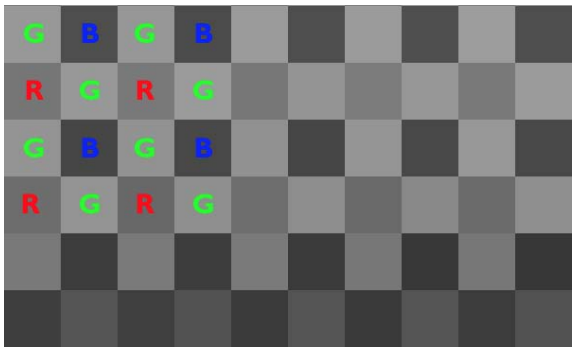
and green (G) filters for even rows. The pattern of the color filters is shown in figure 4.2a). Note that in the Bayer array half of the pixels have a green filter because green is the color for which the human eye is more sensitive. Once the sensor has been exposed to an image, each pixel can be read. A pixels with a green filter provides an exact measurement of the green component, while the red and blue components for this pixel are obtained from the neighbours. For a green pixel, two red neighbours can be interpolated to yield the red value; also two blue pixels can be interpolated to yield the blue value. The demosaicing is then the process that can be used to combine the pixel values in order to obtain a final image which contains full color information at each pixel (see figure 4.2c).

The images were dewarped to compensate for lens distortion. Figure 4.3 shows an example of this last step.

After these steps we applied a feature detector in order to find interest points. Once detected the left and right keypoints, the feature extractor computes a descriptors from the pixels around each interest point, while the matcher finds point correspondences between the two images.

Several algorithms for feature detection and description were considered in the preliminary phases of this work. We then selected a number of detectors and descriptors which have previously shown a good performance in visual odometry and tested them with different dataset. The developed software allows to choose between different detectors and descriptors indeed. The selectable combination of detector and descriptor are SIFT, SURF, FAST+BRIEF and SURF on GPU.

A standard correlation matching algorithm is then used for matching keypoint descriptors. For each descriptor in the first set, the Brute-force matcher used finds the closest descriptor in the second set by trying each one. The same matcher is also used to track keypoints between two time instants. Figures 4.4 and 4.5 show an example of these steps.



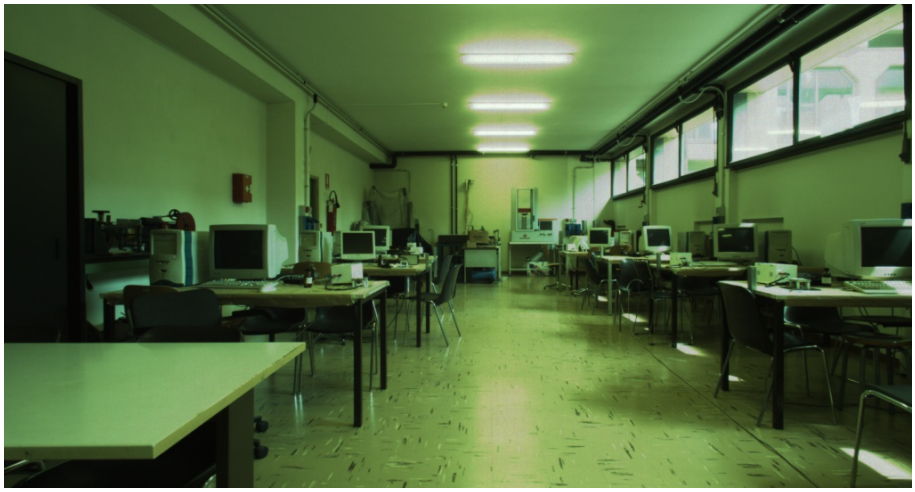
(a) GBRG Bayer pattern: each pixel represents the light intensity captured by the corresponding photo-sensor.

(b) Bayer pattern encoded image.

(c) The image after the demosaicing process.

Figure 4.2: Example of image demosaicing: this algorithm is used to reconstruct a full color image (RGB) from the incomplete color samples output of the sensor overlaid with the color filter array.

(a) Left image before with lens distortion.



(b) Left image after undistortion.

(c) Difference between the fist two images.

Figure 4.3: Example of image undistortion. The images are transformed to compensate for radial and tangential lens distortion.

Figure 4.4: An example of left-right feature matching using SURF detector and descriptor and Brute-force matcher. The circle radius around each keypoint indicates the size of the detected blob.

Figure 4.5: An example of left feature tracking between two instants using SURF detector and descriptor and Brute-force matcher. The circle radius around each keypoint indicates the size of the detected blob.

4.2 Triangulation

Before proceeding we need to clearly define the meaning of a couple of words in the present context. A *bearing vector* is defined as a 3D unit vector pointing at a *landmark* (i.e., a spatial 3D point usually expressed in the world reference frame) from a camera reference frame. It has two degrees of freedom (i.e., azimuth and elevation) in the camera reference frame. A *camera* then denotes a camera reference frame with a set of bearing vectors, all pointing from the origin to landmarks. Finally, an *absolute pose* refers to the position and orientation of a viewpoint (that in the present case coincides with the left camera), with respect to a fixed spatial reference frame called the *world* reference frame.

OpenGV ("Open Geometric Vision") [33] is a C++ library for calibrated real-time 3D geometric vision. It also contains a linear triangulation method that computes the position of a point expressed in the fist camera given a 2D-2D correspondence between bearing vectors from two cameras. Figure 4.6 shows this triangulation problem.

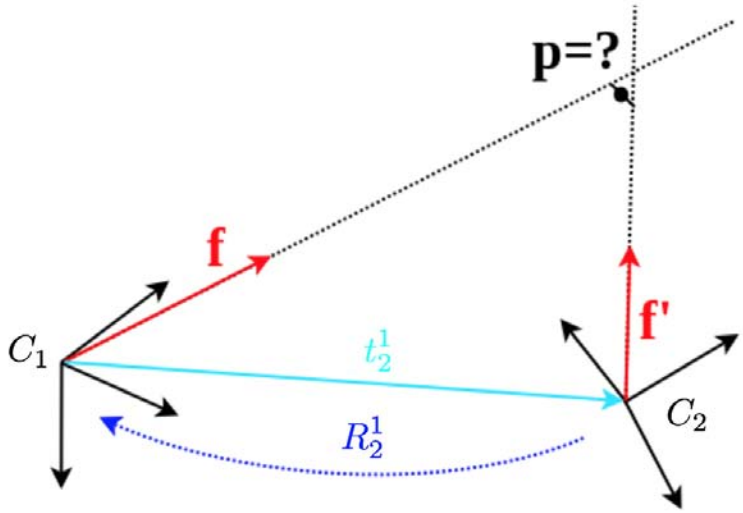


Figure 4.6: An illustration of the OpenGV [28] triangulation problem. The library estimates the 3D position of a point given a 2D-2D correspondence between bearing vectors and the transformation between the cameras, i.e., the position t_2^1 of the second camera seen from the fist one and the rotation R_2^1 from the second camera back to the fist camera frame.

In this way, at every instant the coordinates of every 3D interest point are

known in the left camera reference frame.

4.3 Motion estimation

At this point we have the 2D left image points and the 3D points at the previous instant. Through the feature tracking step we also known the correspondences between these 2D and 3D points. The scheme in figure 4.7 summarized the previous steps, from the images acquisition until the 2D-3D correspondences extrapolation.

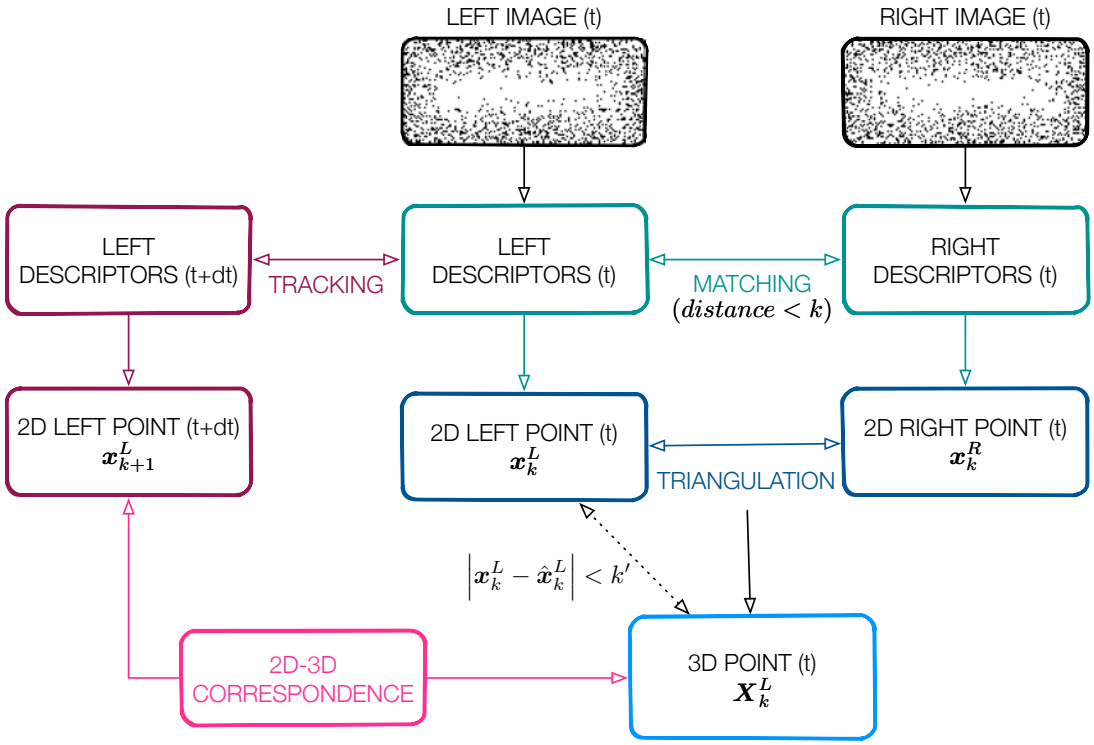


Figure 4.7: Scheme of the 2D-3D correspondences extrapolation.

In the present case we considered a 2D-3D correspondence referring to a bearing vector and a world-point it is pointing at. Figure 4.8 shows an example of these type of correspondences.

The central absolute pose problem consists of finding the pose of a camera in a world frame given a number of 2D-3D correspondences between bearing vectors in the camera frame and points in the world frame. As can be seen in figure 4.9 the sought transformation is given by the position t_c of the camera seen from the

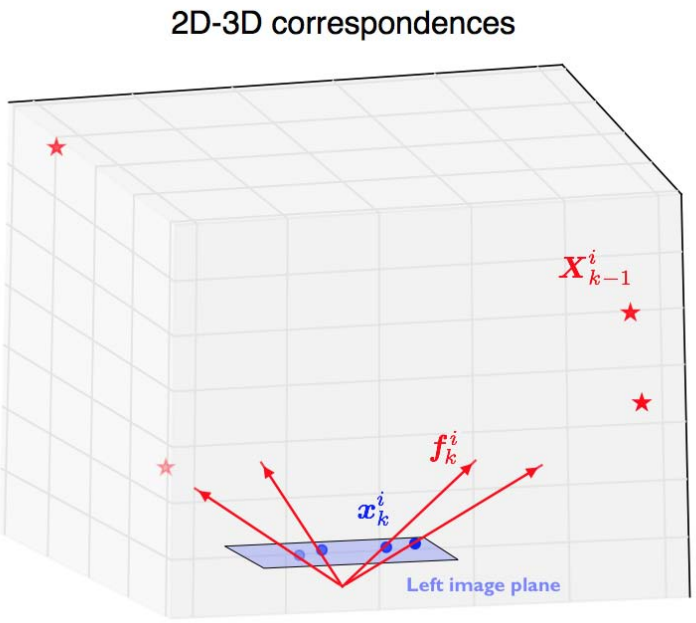


Figure 4.8: An illustration of 2D-3D correspondences. We can see four bearing vectors $\boldsymbol{f}_i$ (that are built from four 2D points $\boldsymbol{x}^i_k$) pointing at four world points $\boldsymbol{X}^i_{k-1}$.

camera frame and the rotation R_c from the camera to the world frame.

The 3D-to-2D motion estimation is implemented using OpenGV [28] library that hosts the minimal P3P solvers presented in [30] and [29], and the n-point solver presented in [31]. The implemented software allows to switch between these three different algorithms, that are named KNEIP, GAO and EPNP algorithm respectively. All these solvers takes as input the 3D unit bearing vectors $\boldsymbol{f}^i_k$ and the 3D world points $\boldsymbol{X}^i_{k-1}$, both expressed in the left camera frame.

4.3.1 Outlier rejection

As written in section 3.3 the standard approach consists of using P3P in a RANSAC scheme in order to remove outliers and then PnP on all remaining inliers.

In detail RANSAC achieves its goal by repeating the following steps.

1. Select a subset of hypothetical inliers (considering only the left image) from the original dataset at random. Each sample contains the smallest number of feature correspondences necessary to obtain a unique solution for the camera

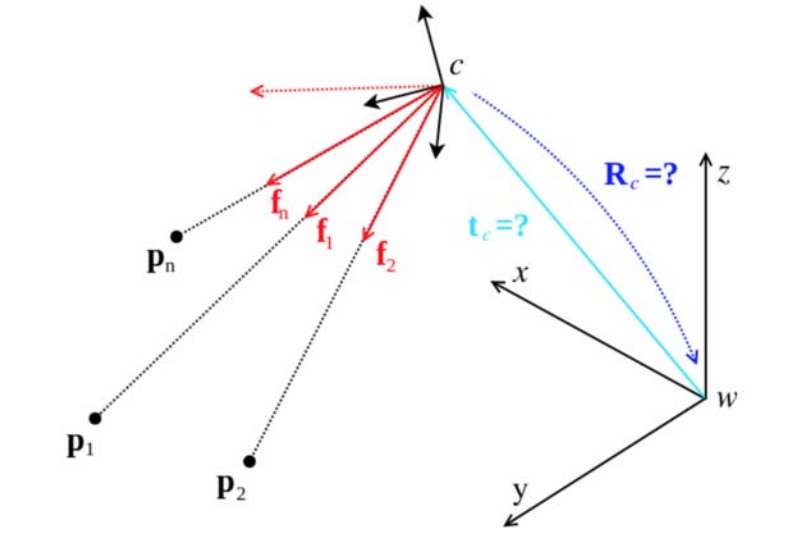


Figure 4.9: An illustration of the OpenGV central absolute pose problem [28].

motion.

2. Fit a model to the set of hypothetical inliers and then find the estimates parameters $\overrightarrow{x}$.

3. Test all the other data and finds how many data items fit the model with parameters $\overrightarrow{x}$ within a user given tolerance. The points that fit the estimated model well (according to the defined threshold) are considered as part of the consensus set, that we call K. The scoring is based on reprojection errors in left frame.

4. The model is considered good if sufficiently many points have been classified as part of the consensus set. Therefore, if K is big enough, we accept fit and exit with success.

5. Next the model may be improved by re-estimating it using all members of the consensus set.

This procedure is repeated a fixed number of times, each time producing either a model which is rejected because too few points are part of the consensus set, or a refined model (with a corresponding consensus set size). In the latter case, we keep refined model if its consensus set is larger than the previously saved model.

The inputs to the algorithm are:

- the dataset for which we are trying to determine a model, which comprises the 3D points (at time t) and the corresponding left bearing vectors (at time $t + dt$);
- the minimum number of points that are required to fit the model, that are 4 for the P3P solvers (i.e., KNEIP and GAO algorithms) and 6 for the EPNP algorithm;
- the threshold value for determining when a data point fits the model, i.e., the maximum distance a data point may be located from the determined model to still be considered an inlier;
- the maximum number of iterations allowed;
- the desired probability of choosing at least one sample free from outliers (default value = 0.99).

While the outputs are:

- the current number of iterations;
- the samples that have been classified as inliers;
- the currently best hypothesis;
- the currently best fit model coefficients, that is the desired transformation.

Because of the 3D nature of the problem, we need a way to compute the threshold reprojection errors in 3D within RANSAC. OpenGV looks at the angle θ between the original bearing-vector $\boldsymbol{f}_{meas}$ and the reprojected one $\boldsymbol{f}_{repr}$ [28]. As illustrated in figure 4.10, by adopting a certain threshold angle $\theta_{threshold}$ we constrain $\boldsymbol{f}_{repr}$ to lie within a cone of axis $\boldsymbol{f}_{meas}$ and opening angle $\theta_{threshold}$.

The threshold angle can be easily approximated with

$$\theta_{threshold} = \arctan \frac{\psi}{l}\,, \tag{4.4}$$

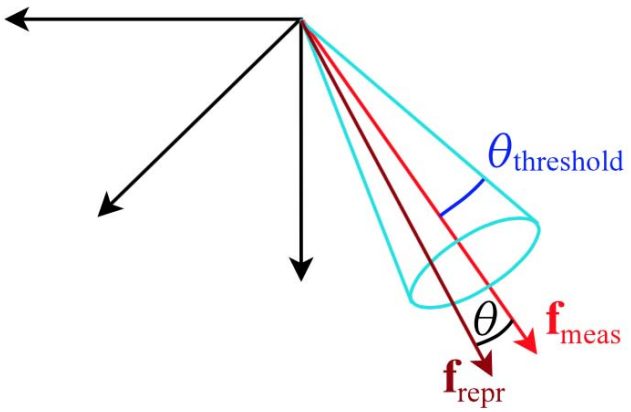


Figure 4.10: Schematic illustration of θ, that is the angle used by OpenGV to compute the reprojection errors in 3D [28].

where ψ is the classical reprojection error-threshold, i.e. the euclidean distance (expressed in pixels) between the 2D point and its corresponding 3D reprojected one, and l is the focal length. The scalar product of $\boldsymbol{f}_{meas}$ and $\boldsymbol{f}_{repr}$, which equals to $\cos\theta$ represent an efficient way to compute the angle between bearing vectors. Since this value is between -1 and 1, and we want an error that minimizes to 0, er express a reprojection error as

$$\varepsilon = 1 - \boldsymbol{f}_{meas}^{T}\boldsymbol{f}_{repr} = 1 - \cos\theta\,. \tag{4.5}$$

The threshold error is therefore given by

$$\varepsilon_{threshold} = 1 - \cos\theta_{threshold} = 1 - \cos\left(\arctan\frac{\psi}{l}\right). \tag{4.6}$$

All the motion estimation algorithms described above are then implemented inside a RANSAC approach. The P3P problem needs three points and a fourth one to disambiguate the solution.

As mentioned in section 3.3, the RANSAC algorithm needs a certain percentage of inliers to work well and the probability of a reasonable model being produced increases with the number of iterations.

In order to reduce the computational cost of the RANSAC step we have to

eliminate a part of the outliers before it. To do this we made two checks.

- In the stereo matching step a part of the outliers are removed if the euclidean distance between their descriptors is too high.

- In order to remove another part of the outliers the reprojection error is calculated. It is a geometric error corresponding to the image distance between a 3D reprojected point and a 2D measured one,

$$d_{repr} = \left\| \boldsymbol{x}_k^L - \hat{\boldsymbol{x}}_k^L \right\| , \tag{4.7}$$

  where $\boldsymbol{x}_k^L$ is the 2D left point and $\hat{\boldsymbol{x}}_{k-1}^L$ is the reprojection of the 3D point $\boldsymbol{X}_{k-1}^L$ into left image plane. If this error is larger than a few pixels, the match can be rejected since it is not geometrically possible (i.e., the ray from the camera through the image feature is not oriented towards the 3D triangulated point). In this way we store only the points whose reprojection error falls within a certain threshold.

Figure 4.11 shows an example of good 2D-3D correspondences after the outlier rejection steps, while figure 4.12 gives an example of outlier rejection using RANSAC.

4.3.2 Non-linear optimization

A further non-linear optimization can also be applied to refine the final solution. OpenGV library contains a non-linear optimization method that minimizes the image reprojection error

$$\arg\min_{T_k} \sum_i \left\| \boldsymbol{x}_k^i - \hat{\boldsymbol{x}}_{k-1}^i \right\|^2 \tag{4.8}$$

where $\boldsymbol{x}_k^i$ is the 2D point of image I_k, while $\hat{x}_{k-1}^i$ is the reprojection of the 3D point $\boldsymbol{X}_{k-1}^i$ into image I_k according to the transformation T_k.

Figure 4.11: Example of 2D-3D correspondences after the outlier rejection steps. The red number indicates the 3D reprojected points on the left image plane, $\hat{\boldsymbol{x}}^i_{k-1}$, while the green ones are the corresponding 2D points, $\boldsymbol{x}^i_k$. The outlier rejection allow to remove almost all of the bad correspondences and thus to reduce the computational time of the motion estimation step.

4.3.3 Trajectory reconstruction

The procedure described above is repeated for all the image pairs, obtaining a transformation matrix T^k_{k-1} between two sequential sets of images. By concatenation of all these single movements, the full trajectory of the left camera can be recovered.

Figure 4.12: An example of outliers rejection through RANSAC. The two figures refer to two different time steps and show the 2D-3D correspondences between 3D reprojected point (at time $t - dt$) and 2D image points (at time t). The green correspondences are those that RANSAC considers inliers, while the red ones are treated as the outliers and not used in order to compute the final camera pose.

Chapter 5

Experimental set-up

In order to measured the algorithm performance, the software was tested on some datasets recorded by the laboratory stereo camera, whose intrinsic and extrinsic parameters are illustrated in tables 5.1 and 5.2. A detailed analysis of this set-up can be found in [34] and [35].

The stereo camera was mounted both on a linear slide and on a motor-driven rotary stage that are provided with a graduate scale in order to compare the measurements obtained by the visual system respectively with known linear displacements and with known rotation angles. Figure 5.1 shows this set-up. The images recorded have a 2040×1086 px resolution.

		Left camera	Right camera
Focal length [mm]	f	6.0	6.0
Focal length * pixel density [px]	$\alpha_x = fS_x$	1133.20108	1136.43126
	$\alpha_y = fS_y$	1133.19673	1135.97654
Principal point [mm]	O_x	1058.25306	1056.95157
	O_y	524.70888	542.00913
Distortion coefficients	k_1	-0.14684	-0.14562
	k_2	0.08434	0.08334
	p_1	-0.00003	-0.00016
	p_2	0.00089	0.00042
	k_3	-0.01645	-0.01609

Table 5.1: Stereo-camera intrinsic parameters.

In order to test the performance of the software on extended sequences, some

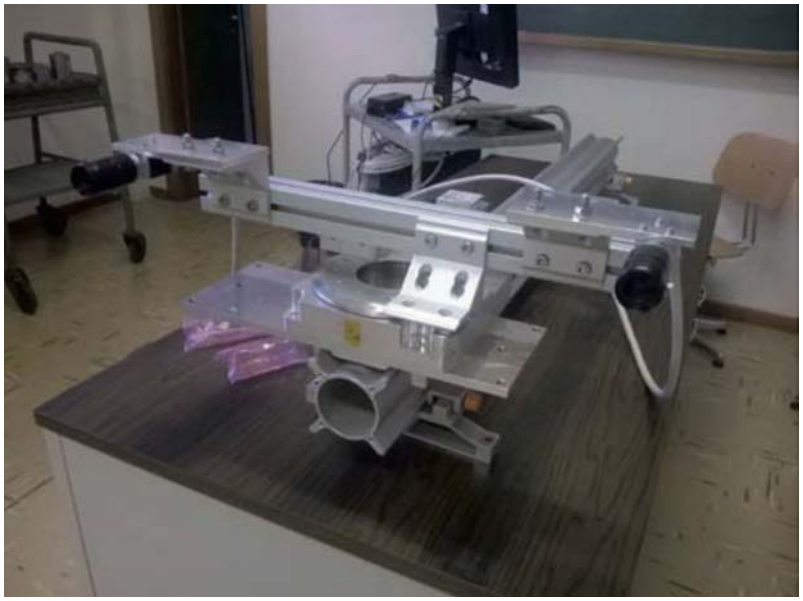


Figure 5.1: The experimental set-up used to record the datasets [34]. It comprises the stereo camera mounted on the rotary stage, which can translate along the linear slide.

Euler vector [rad]	$\begin{bmatrix}0.02 & 0.1 & 0.4\end{bmatrix}$
Translation vector [mm]	$\begin{bmatrix}546 & -1.5 & -3.5\end{bmatrix}$

Table 5.2: Stereo-camera extrinsic parameters.

different datasets of stereo frames are adopted. They relate to two different types of test.

Translation test: it is a straight displacement along $z-$axis with a total travel of 1350 mm; it is performed using the linear slide and acquiring the images every 10 mm.

Rotation test: it is an axial displacement from 0° to 90° around $y-$axis; it is performed using the rotary stage and acquiring the images every 0.5°.

In this way we could also test the VO algorithm with different steps between two consecutive sets of images. Figures 5.2 - 5.5 show some examples of input images used in our experiments.

(a) Left and right first images of the translation dataset sequence after the demosaic step.

(b) Left and right last images of the translation dataset sequence after the demosaic step.

Figure 5.2: Translation sequence No.1.

(a) Left and right first images of the translation dataset sequence after the demosaic step.

(b) Left and right last images of the translation dataset sequence after the demosaic step.

Figure 5.3: Translation sequence No.2.

(a) Left and right first images of the rotation dataset sequence after the demosaic step.

(b) Left and right last images of the rotation dataset sequence after the demosaic step.

Figure 5.4: Rotation sequence No.1.

(a) Left and right first images of the rotation dataset sequence after the demosaic step.

(b) Left and right last images of the rotation dataset sequence after the demosaic step.

Figure 5.5: Rotation sequence No.2.

5.1 The MORPHEUS project

The MORPHEUS project consists in designing and building the first students rover of the University of Padova.

MORPHEUS is designed to be a functional rover, capable to perform different tasks similar to ones that will support the future manned Mars exploration. In detail, the rover will be able to sample surface soil and rocks, to drill, to autonomously drive in harsh terrains and to operate assistant and maintenance tasks. The rover has six wheels, connected to the chassis through three rockers, and it is able to drive using a skid steering configuration. It is also equipped with a robotic arm (with three hot swappable manipulators for different tasks), a sample extraction drill, a stereoscopic vision system and an omnidirectional antenna for communications.

MORPHEUS rover is design to have a full speed of $1\,m/s$ ($15°$) with an acceleration of $0.2\,m/s^2$. In addition to the stereo-vision system, it is equipped with different sensors. The localization and the pose of the rover will be estimated through the measurements collected by a GPS module and an IMU (accelerometers and gyroscopes). While the GPS provides only a rough estimate of the position with low update frequency, the IMU gives more precise data with high update frequency; on the other hand the accelerometers and gyroscopes can only give a relative measure with an error that increases over time (due to the integration of the accelerations and velocities) whereas the GPS measure is absolute, as it doesn't depend on the motion of the rover. A sensor fusion algorithm is then needed to put together all these informations.

Our purpose was to implement a real-time long-run visual odometry algorithm

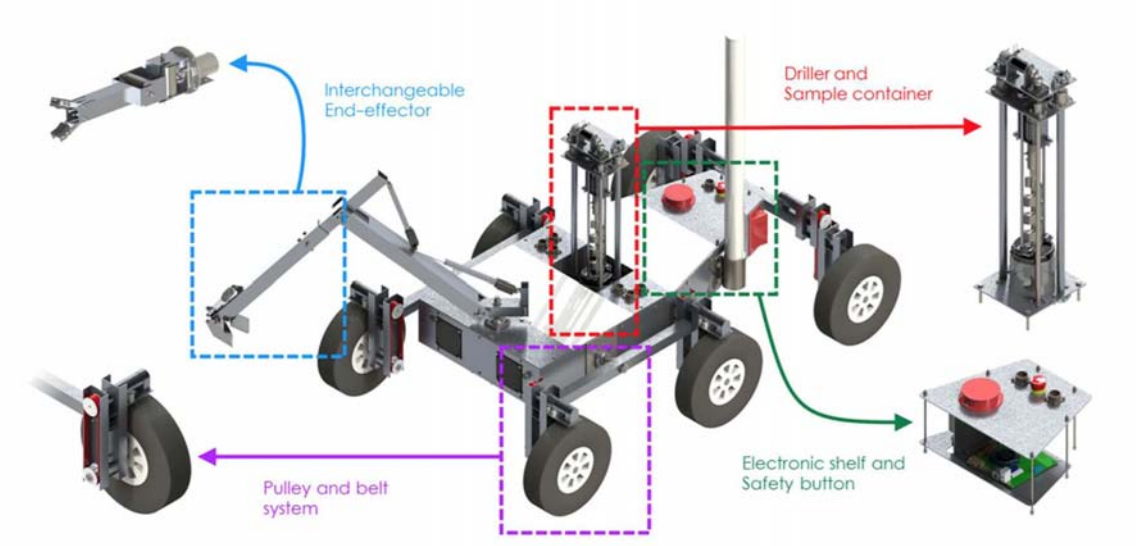


Figure 5.6: MORPHEUS overview.

that enables to determine the rover pose. The vehicle stereo-vision system for autonomous motion control is composed by two Raspberry-Pi camera modules and a Raspberry-Pi compute module, while the main rover board is the Jetson TK1, that is an embedded development platform from NVIDIA (figure 5.8). The visual odometry software developed has been tested on this board.

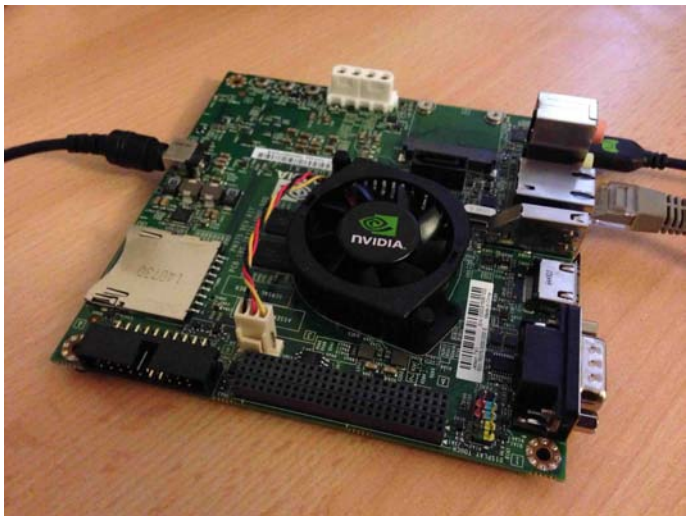


Figure 5.7: The NVIDIA Jetson TK1 is the rover core, exploiting a Tegra K1 SOC.

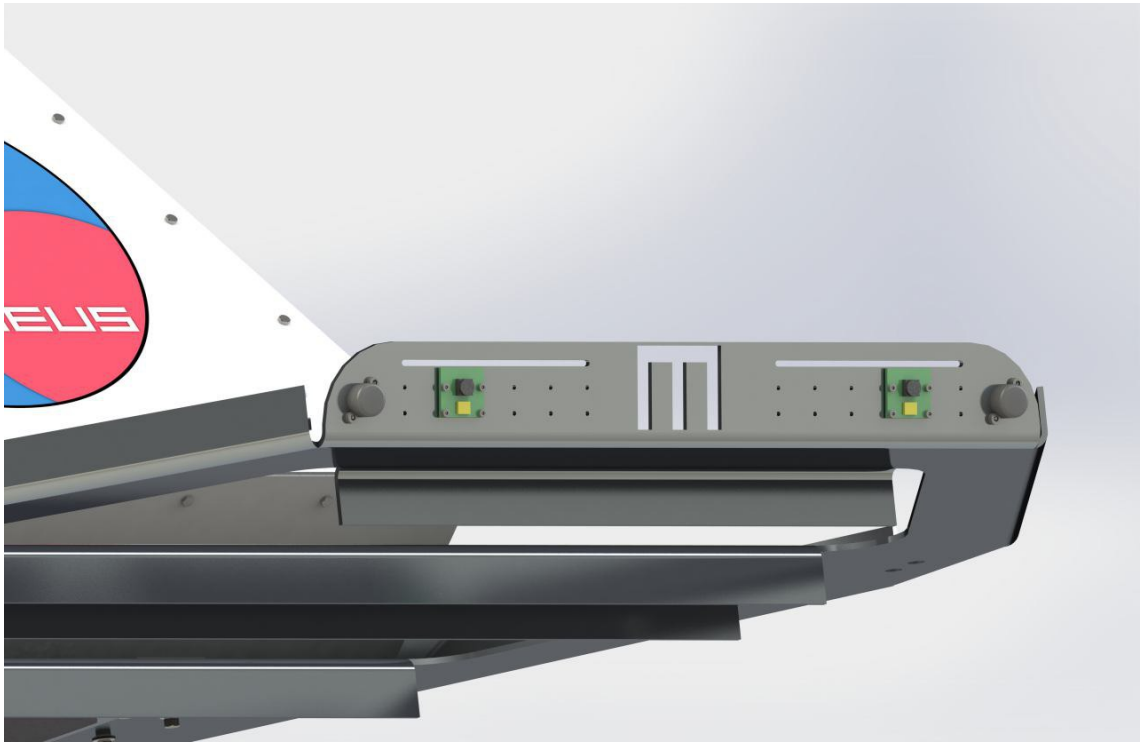


Figure 5.8: Stereo camera system on the front of the MORPHEUS rover. MORPHEUS is equipped with two Raspberry Pi cameras that gain two synchronous images through the Raspberry Compute Module board.

Chapter 6

Results

6.1 Image processing

Accuracy of feature localization and computation cost are crucial aspects for visual odometry. As can be seen in table 6.1, the computational time for the motion estimation step is typically a small fraction of the time required for the visual odometry process, while the image processing is the most expensive stage.

Step	Computational cost
loading images	2.11%
undistortion	10.18%
SURF detector	29.82%
SURF descriptor	50.18%
BF matcher	5.61%
Motion estimation	1.80%

Table 6.1: Computational time percentage of different algorithm parts.

Image processing evaluations have been conducted using different feature detection and description algorithms. Their performance has been estimated using some indirect indicators that are the number of inliers/matches per frame, the computational time, the iterations in RANSAC and the error per frame in the motion estimation.

Figures 6.1, 6.2 and 6.3 display the keypoints detected with the different extractors and the feature matching found using the consistent descriptors. The selected

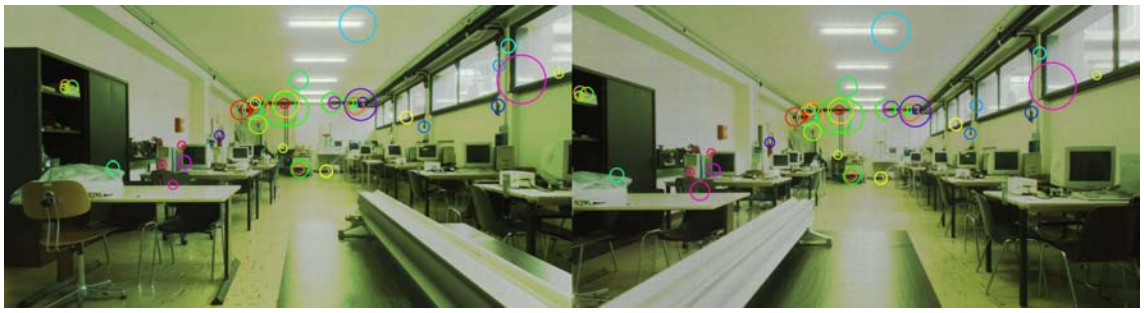
(a)

(b)

Figure 6.1: An example of feature detection and description using SIFT algorithm. In detail: (6.1a) keypoints extracted using the SIFT algorithm; note that at each keypoint is assigned one orientation based on local image gradient directions; (6.1b) matches detected using the SIFT descriptor and the Brute Force matcher.

landmarks appears to be well distributed in the area of the image, although relatively few landmarks are selected close to the cameras. The matching locations were then detected in the corresponding right image using stereo matching. As described in 4.3.1, some matches were discarded at this step through examination of the correlation score and the gap between the 2D points and the 3D reprojected one.

(a)



(b)

Figure 6.2: An example of feature detection and description using SURF algorithm. In detail: (6.2a) keypoints extracted using the SURF algorithm; (6.2b) matches detected using the SURF descriptor and the Brute Force matcher.

(a)

(b)

Figure 6.3: An example of feature detection and description using FAST and BRIEF algorithms. In detail: (6.3a) keypoints extracted using the FAST algorithm; (6.3b) matches detected using the BRIEF descriptor and the Brute Force matcher.

An intuition evaluation criterion for detectors and descriptors is the percentage of matched inliers in all the matched points. The detection of few or even some wrong correspondences will lead to failure in motion estimation indeed. The graph in figure 6.4 and table 6.2 show the percentage of matched inliers obtained using different feature detectors and descriptors.

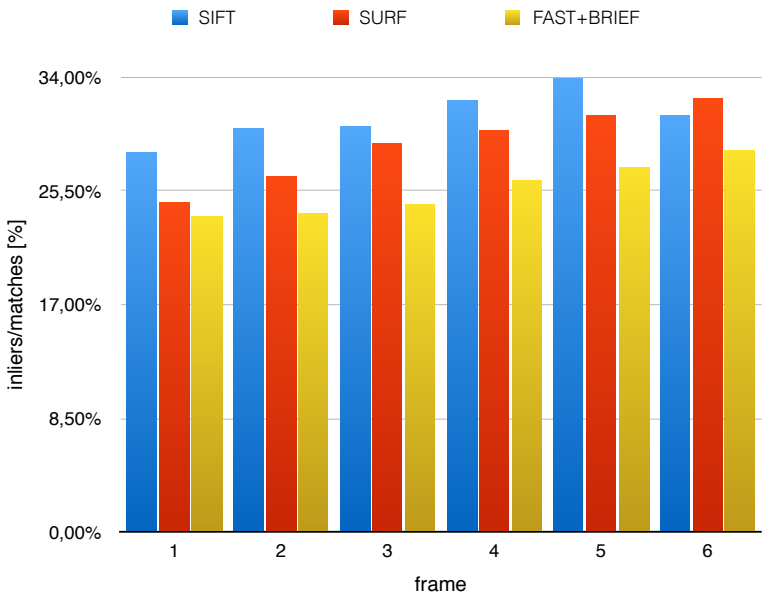


Figure 6.4: Matched inliers percentage using different feature detectors and descriptors for each image pair.

	Matches	Inliers	Inliers/Matches [%]
SIFT	1019	316	30.98
SURF	2352	680	28.92
FAST+BRIEF	5046	1294	25.64

Table 6.2: Average percentage of matched inliers in all the matched points found using the different feature detectors and descriptors.

The localization error must be the most useful criterion in motion estimation, so we also evaluate the relationships of different detectors and descriptors with the error between the estimated position and the true one. It is a comprehensive criterion to represent performance of detectors and descriptors, because accuracies of feature detection and descriptor will all affect the localization error. Figures 6.5 and 6.6 display the errors in the motion estimation, highlighting the error computed as the

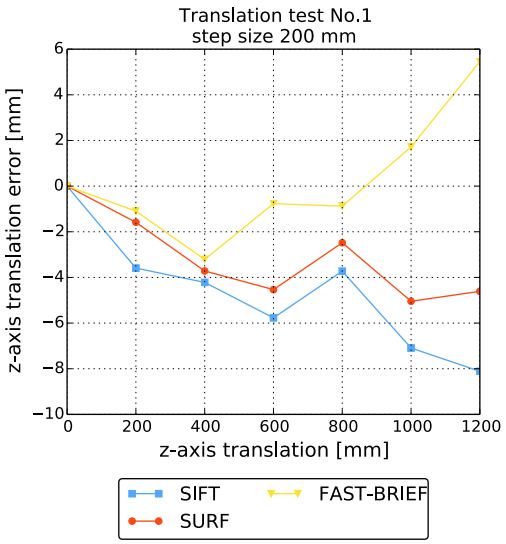
difference between the estimated position and the correct one in the ground truth, the percentage error and the error cumulated at each step.

The sample iterations in RANSAC may be another interested criterion to evaluate the performance of descriptors. The iteration number indicates how easily RANSAC is able to obtain accurate initial motion estimation and thus it is a direct measure of time spent for outlier removal. Table 6.3 shows the iteration number of RANSAC for different descriptors.

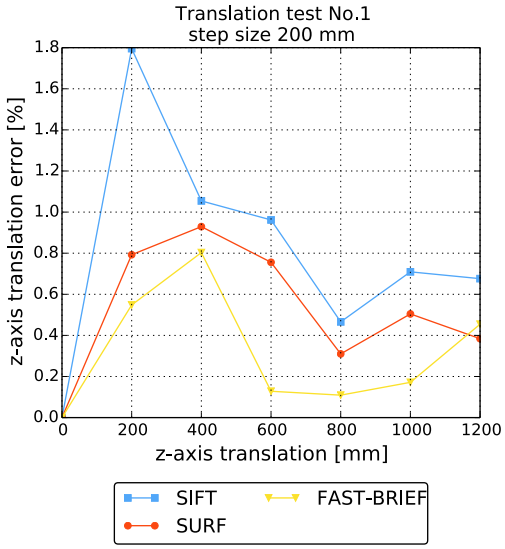
	frame	Iterations number	Consensus set size
SIFT	1	11/10000	139
	2	12/10000	138
	3	24/10000	113
	4	13/10000	125
	5	12/10000	139
	6	19/10000	124
SURF	1	8/10000	97
	2	9/10000	90
	3	7/10000	89
	4	12/10000	92
	5	11/10000	98
	6	8/10000	120
FAST + BRIEF	1	31/10000	184
	2	44/10000	182
	3	34/10000	170
	4	39/10000	172
	5	10001/10000	7
	6	55/10000	168

Table 6.3: RANSAC iterations number and consensus set size for different detectors and descriptors. Note that when the number of iterations exceeds the maximum, there aren't enough inliers that fit the model and the motion estimation fails.

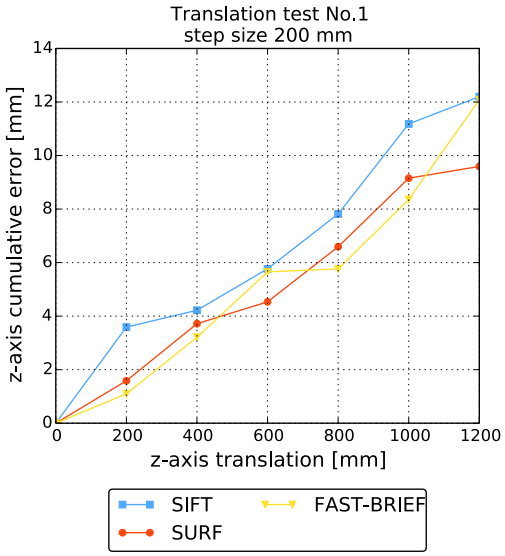
As can be seen the accuracies of the different feature detectors and descriptors are similar. However, while SIFT and SURF have proven to be reliable with all the datasets used, FAST and BRIEF gave random bad results sometimes. This is because the RANSAC step exceed the maximum number of iteration allowed. The output is then a wrong model since none of the ones considered agrees with a quite large consensus set. This probably happens because the descriptor failed the feature matching or tracking step.



(a) Error relative to the ground truth.

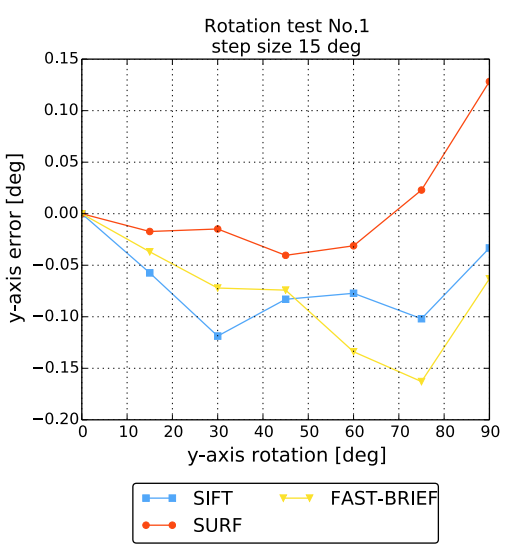


(b) Percentage error.

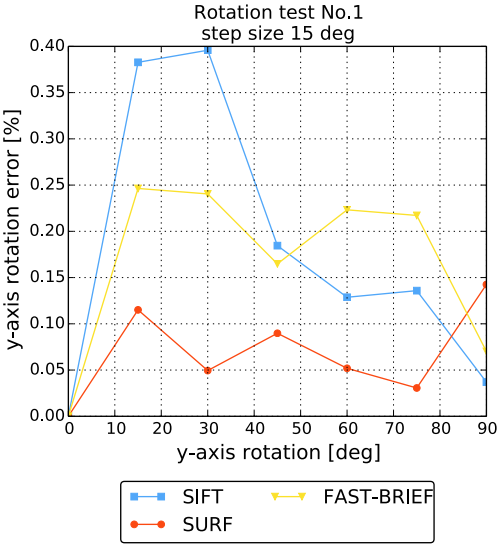


(c) Cumulative error.

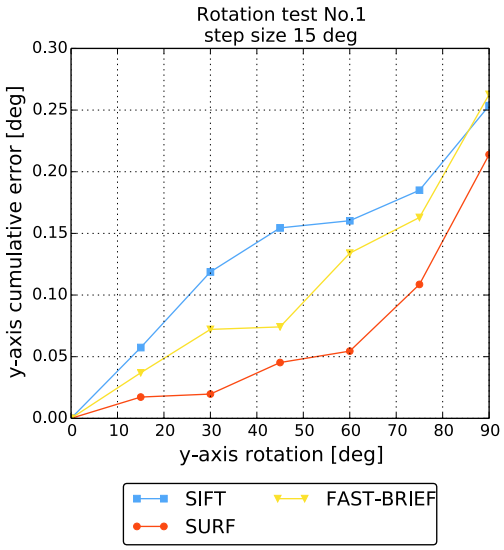
Figure 6.5: A comparison between different detectors and descriptors accuracy on the translation sequence. Graph 6.5a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.5b displays this percentage error; finally 6.5c shows the error cumulated at each step.



(a) Error relative to the ground truth.



(b) Percentage error.



(c) Cumulative error.

Figure 6.6: A comparison between different detectors and descriptors accuracy on the rotation sequence. Graph 6.6a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.6b displays this percentage error; finally 6.6c shows the error cumulated at each step.

Because visual odometry is a real-time application, we also compared the executing time for different detectors and descriptors using the same hardware. In this experiment, all the detectors and descriptors are implemented to process the same image sequence, with a resolution of 2040×1086 px. We recorded the average executing time of each detector and descriptor to process a pair of images. The results averaged over the number of image pairs are shown in table 6.4, that illustrates the times required on the Jetson TK1 to process the software main steps highlighting the image processing stages.

The experimental results show SIFT detector and descriptor spend much more time in computation than other detectors and descriptors, while FAST detector and BRIEF descriptor are much faster then SURF, but take too long on matching and tracking steps.

Therefore all these results suggest that SURF may be a proper solution for stereo visual odometry when considering the robustness, accuracy and executing time in all.

	Execution time [s]		
	SIFT	SURF	FAST+BRIEF
Loading images	0.144789	0.144789	0.144789
Images demosaicing	0.027656	0.027656	0.027656
Images undistortion	0.498810	0.498810	0.498810
Detector	6.902458	2.141598	0.055885
Descriptor	6.621392	3.074577	0.134562
Matching	0.269174	0.342911	2.136082
Tracking	0.279834	0.345526	2.036842
[R|t] computation	14.087950	5.915622	4.558575
Total time	14.753033	6.578997	5.222868

Table 6.4: Comparison of times required on Jetson TK1 to perform the software main steps using different feature detectors and descriptors on 2040×1086 px images. Note that the $[R|t]$ computation time includes the image processing steps (i.e., detection, description, matching and tracking).

Because we are interested in real-time performance we chose to use the Jetson TK1 GPU for the image processing stages. OpenCV [12] contains different feature detectors, e.g., SIFT, SURF, BRISK, FREAK, STAR, FAST, ORB. All of these

have implementation on CPU, but only SURF and ORB on GPU. We have chosen SURF because it is actually the only scale/rotate-invariant feature detector with GPU support in OpenCV. In detail, `SURF_GPU` contains a fast multi-scale Hessian keypoint detector and implements Speeded Up Robust Features descriptor.

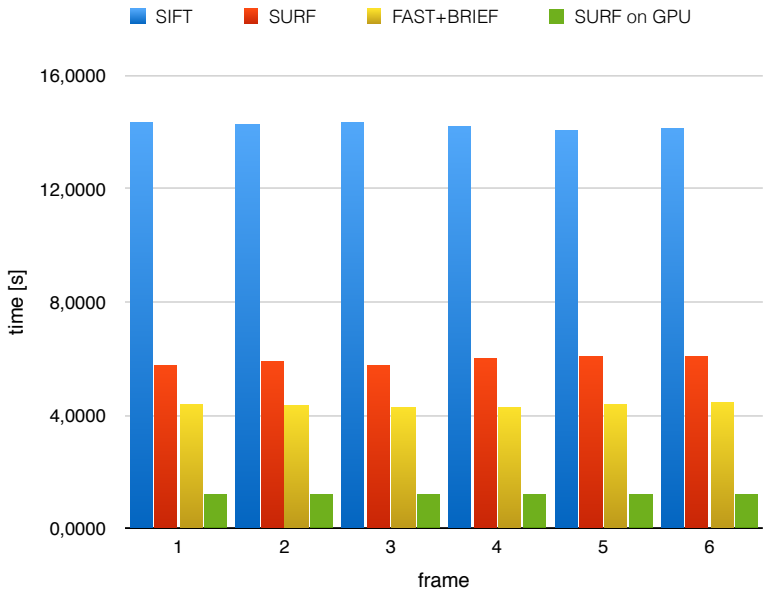
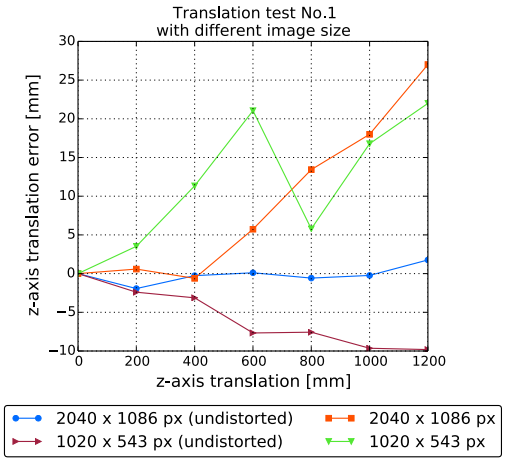
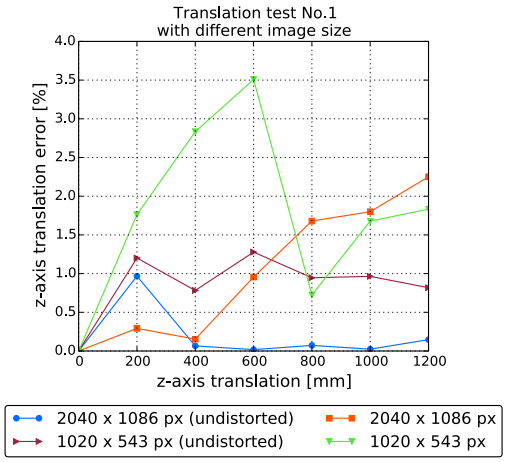


Figure 6.7: Different detector and descriptor times on Jetson TK1 for each image pair. These times includes the detection, the description and the matching steps on two 2040×1086 px images.

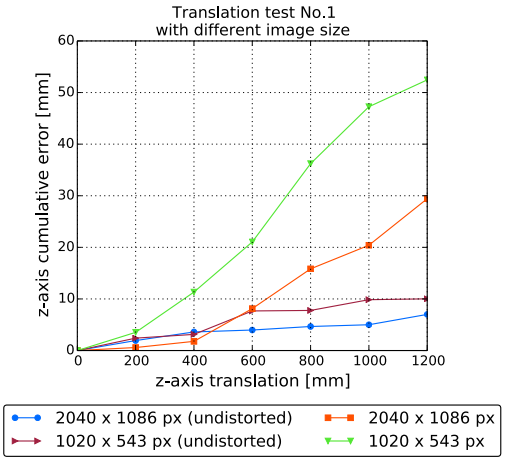
We tested the GPU implementation of SURF on the same sequence of 2040×1086 px images and on a resize version of 1020×543 px images. Table 6.5 shows the results. Figure 6.8 displays the motion estimation error using different images size and shows the loss of accuracy that occur removing the undistortion stage. Note that accuracy worsens slightly reducing the image size: the average error increases from 0.6% to 0.84% halving the image size.



(a) Error relative to the ground truth.



(b) Percentage error.



(c) Cumulative error.

Figure 6.8: A comparison between the motion estimation accuracy on images of different size on the translation sequence No.1. Graph 6.8a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.8b displays this percentage error; finally 6.8c shows the error cumulated at each step.

	Execution time [s]	
	2040 × 1086 px	1020 × 543 px
Loading images	0.144789	0.141727
Images demosaicing	0.027656	0.011658
Images undistortion	0.498810	0.124475
Detector + Descriptor	1.154352	0.587790
Matching	0.039996	0.010242
Tracking	0.042186	0.011710
[R|t] computation	1.248073	0.337477
Total time	1.907913	0.614721

Table 6.5: Comparison of times required on Jetson TK1 to perform the software main steps using the implementation on GPU of SURF for different image size: the first column refers to 2040 × 1086 px images, while the second one to 1020 × 543 px images. Note that the $[R|t]$ computation time includes the image processing steps (i.e., detection, description, matching and tracking).

(a) Hypothesis.

(b) Consensus set.

Figure 6.9: Example of (a) a subset of four points used to determine the model parameters (hypothesis) and (b) the consensus set that fits this model used in the RANSAC outlier-rejection step.

6.2 Motion estimation accuracy

As described in chapter 3 three different motion estimation algorithms were tested. We will refer to them as Kneip, Gao and EPnP. Each algorithm was run on the same set of input images. These algorithms were performed inside a RANSAC scheme, followed by a non-linear optimization.

Figure 6.9 displays an example of four point RANSAC hypothesis with the relative consensus set, while table 6.6 shows an average value of the iterations needed, of the output inliers number and of the time required by the RANSAC scheme using the different algorithms.

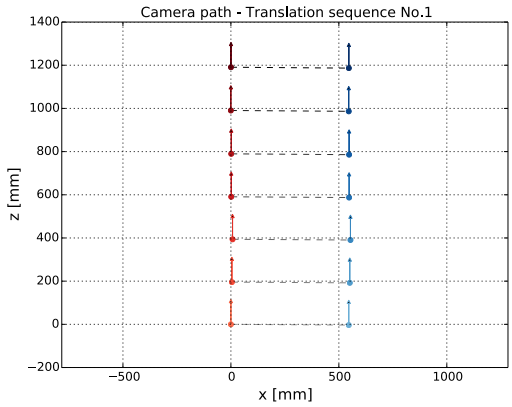
	Iterations number	Inliers number	Time [ms]
Kneip algorithm	11.00	105.67	0.39772
Gao algorithm	10.67	104.59	0.66118
EPnP algorithm	18.17	106.17	4.25493

Table 6.6: RANSAC parameters using different algorithms. These results represents the mean value of the data collected for each sequence step.

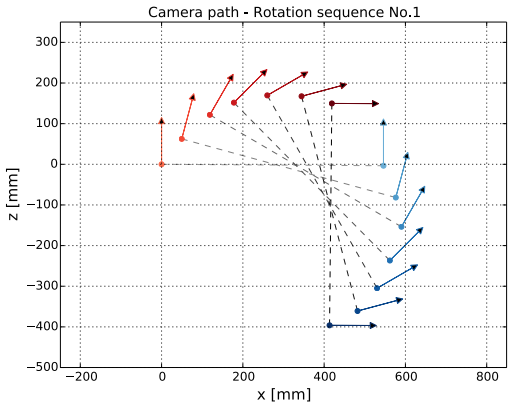
As mentioned in the previous section, the most useful criterion in motion estimation must be the localization error. We then evaluate the three algorithms measuring the error between the estimated position and the true one. A sample calculation based on the expected vehicle translation and rotation rates allows us to compute this position error. Figures 6.11 - 6.18 summarize the errors in the motion estimation for each dataset, highlighting the error computed as the difference between the estimated position and the correct one in the ground truth, the percentage error and the error cumulated at each step. Figure 6.19 and 6.20 show the errors on the three axis.

Note that the error relative to the ground truth may be either positive or negative and therefore has a zero mean. The cumulative error is a more significant parameter to evaluate the algorithms accuracy instead, since some methods show an higher variance then others. Moreover the results for each run are slightly different because of the random nature of RANSAC.

The graphs show that the non-linear optimization significantly improves the



(a) Cameras path on $x - z$ plane for the translation test No.1.



(b) Cameras path on $x - z$ plane for the rotation test No.1.

Figure 6.10: Different stereo-camera paths corresponding respectively to a translation sequence and a rotation one. The red dots represent the left camera, while the blue ones the right camera. The arrows illustrates the cameras orientation at each step. The initial position of the left camera is defined by the coordinates $(0, 0)$.

motion estimation precision, while we can't notice any appreciable difference among the three algorithms. Furthermore, their computational costs are comparable within the whole process.

Table 6.7 and 6.8 illustrate different tests performed on the image sequences showing the average error in the motion estimation process. The cross symbol indicates some configuration where the motion estimation fails because RANSAC iterations exceed the limit without finding a proper consensus set. As you can see from the tables, this may occur if the step size is too small. Figure 6.14a shows an example of this occurrence: suddenly the error abruptly increases because the software ignores a wrong measure.

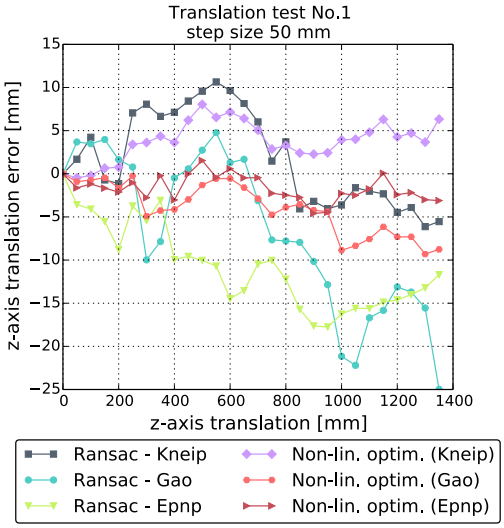
Figure 6.10 shows the camera paths of translation sequence No.1 and rotation sequence No.1.

		Average error [mm]		
	Step size [mm]	KNEIP	GAO	EPnP
Translation sequence No.1	10	✗	✗	✗
	20	0.98	1.05	1.09
	50	1.05	1.11	1.21
	100	1.12	1.69	0.90
	200	1.92	0.72	1.01
	300	3.45	2.28	2.12
Translation sequence No.2	10	✗	✗	✗
	20	✗	✗	✗
	50	1.26	1.09	1.29
	100	1.30	1.00	1.06
	200	2.12	1.62	1.89
	300	2.57	2.55	2.91

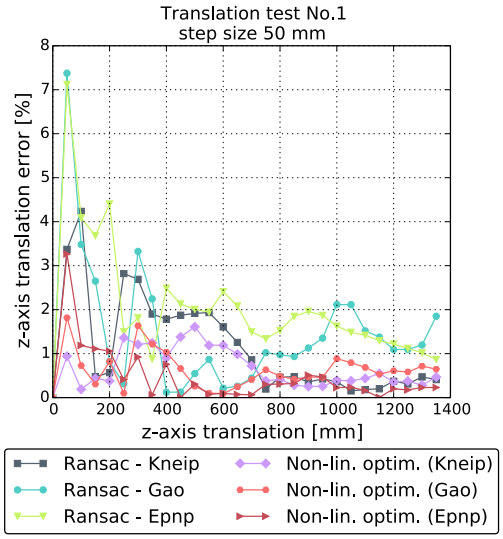
Table 6.7: Average errors on the translation sequences obtained using different step sizes. The cross symbol indicates that the algorithm fails in one or more motion estimation step. This is because the RANSAC iterations exceed the limit without finding a proper consensus set.

		Average error [deg]		
	Step size [deg]	KNEIP	GAO	EPnP
Rotation sequence No.1	1	0.024	0.023	0.020
	2	0.020	0.020	0.022
	5	0.035	0.044	0.027
	10	0.047	0.030	0.079
	15	0.063	0.059	0.068
	20	0.055	0.054	0.033
Rotation sequence No.2	1	✗	✗	✗
	2	0.034	0.038	0.033
	5	0.041	0.057	0.034
	10	0.039	0.059	0.060
	15	0.055	0.040	0.031
	20	0.081	0.031	0.037

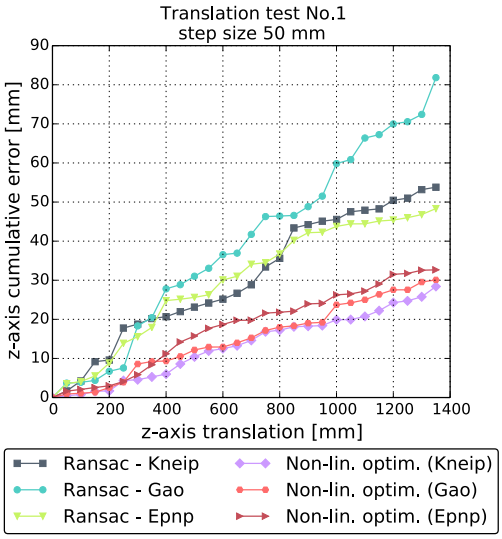
Table 6.8: Average errors on the rotation sequences obtained using different step sizes. The cross symbol indicates that the algorithm fails in one or more motion estimation step. This is because the RANSAC iterations exceed the limit without finding a proper consensus set.



(a) Error relative to the ground truth.

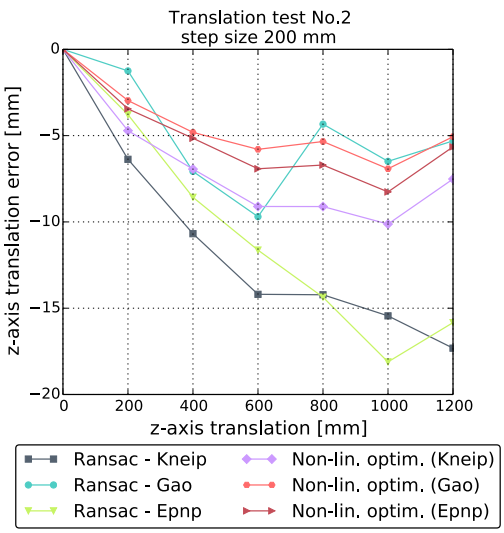


(b) Percentage error.

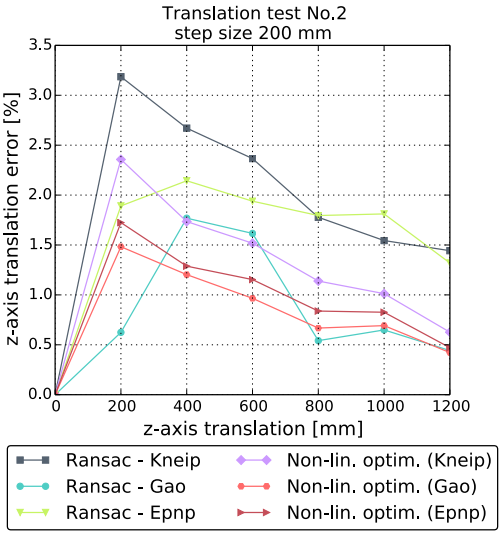


(c) Cumulative error.

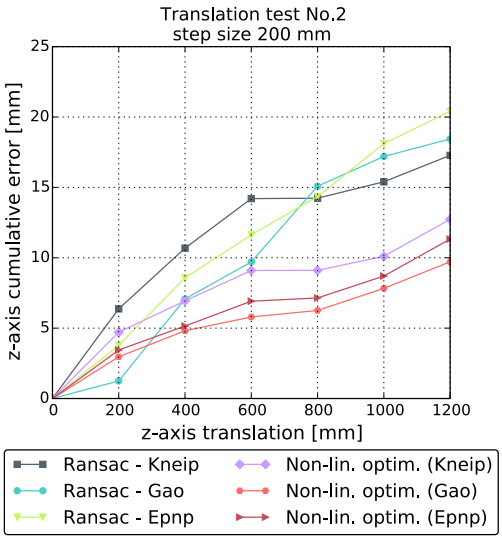
Figure 6.11: A comparison between different algorithm accuracies on the translation sequence No.1. Graph 6.11a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.11b displays this percentage error; finally 6.11c shows the error cumulated at each step.



(a) Error relative to the ground truth.

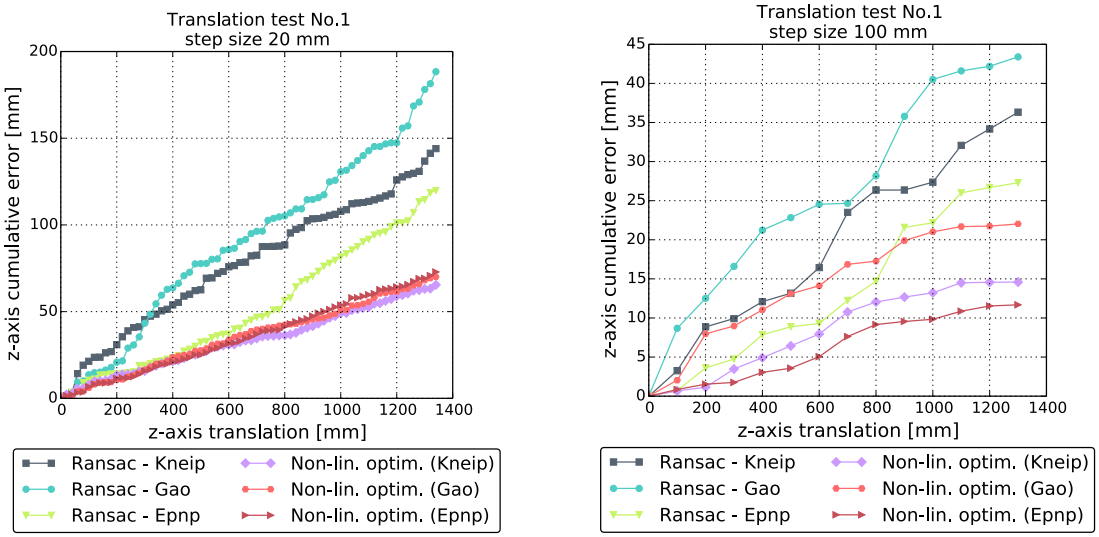


(b) Percentage error.



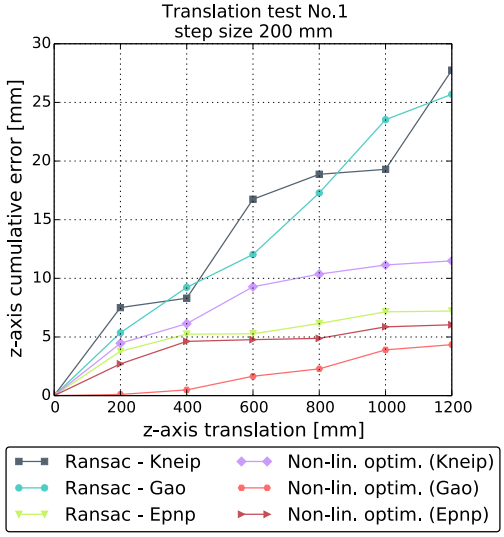
(c) Cumulative error.

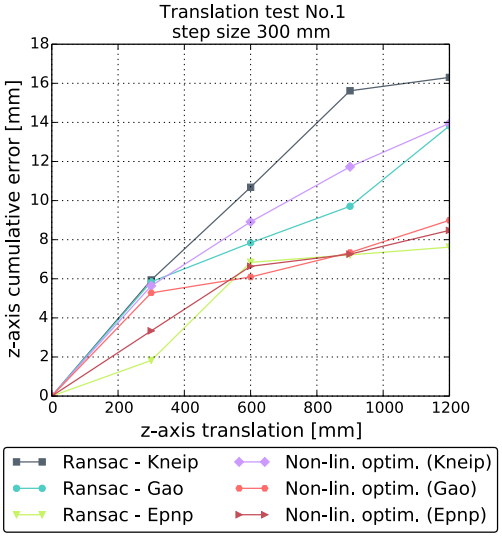
Figure 6.12: A comparison between different algorithm accuracies on the translation sequence No.2. Graph 6.12a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.12b displays this percentage error; finally 6.12c shows the error cumulated at each step.



(a)

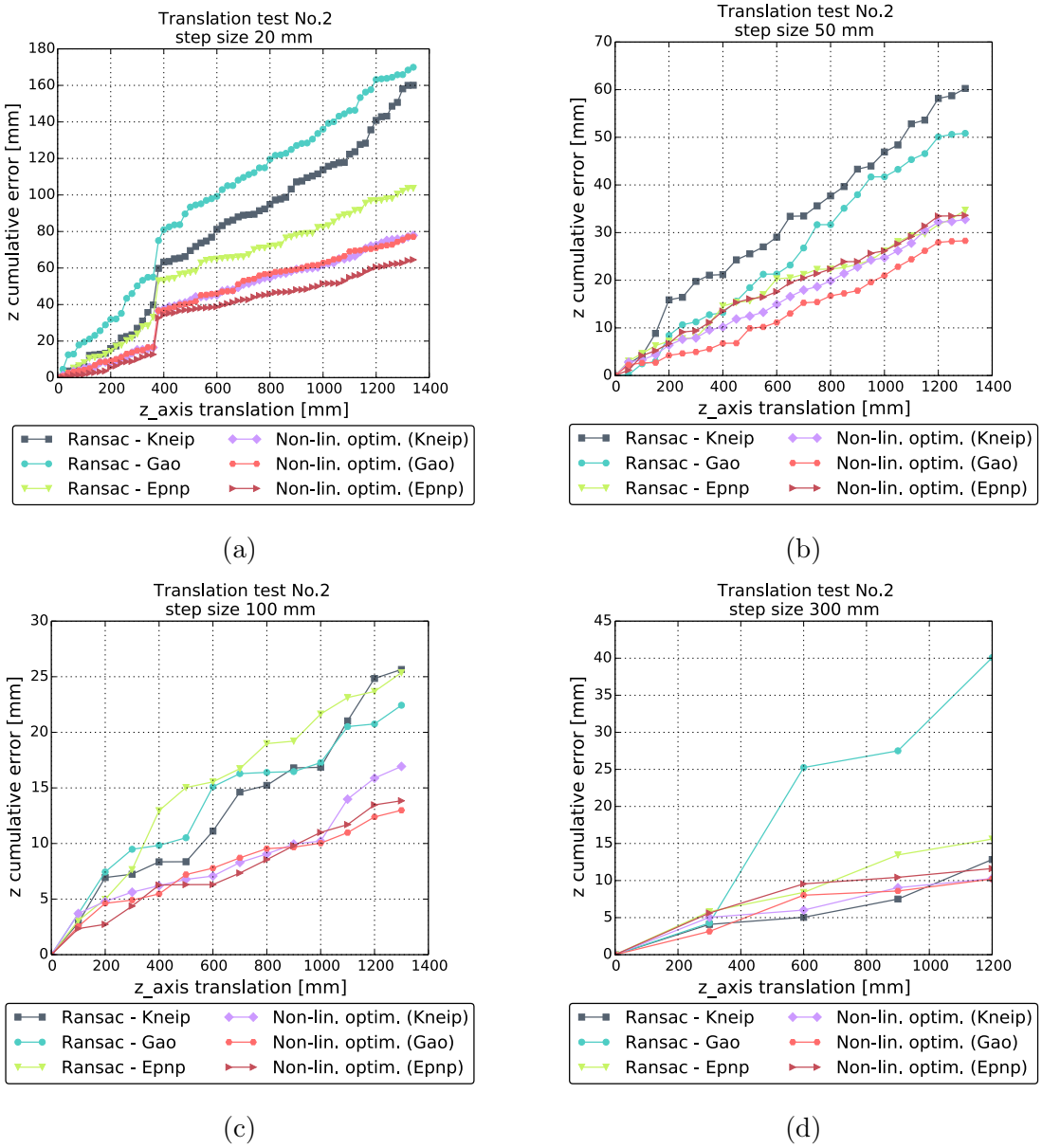
(b)





(c)

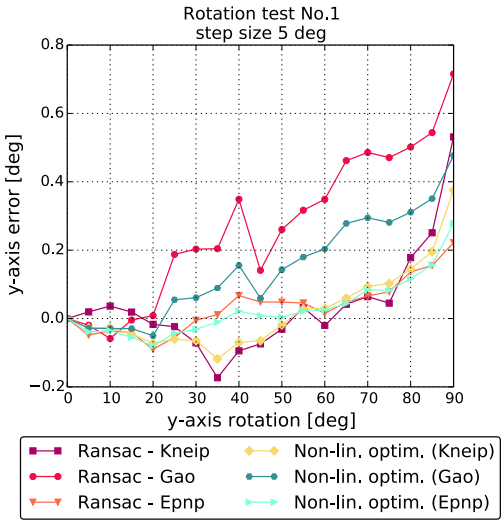
Figure 6.13: Cumulative errors calculated by adding the absolute errors for some different step sizes on the translation sequence No.1.



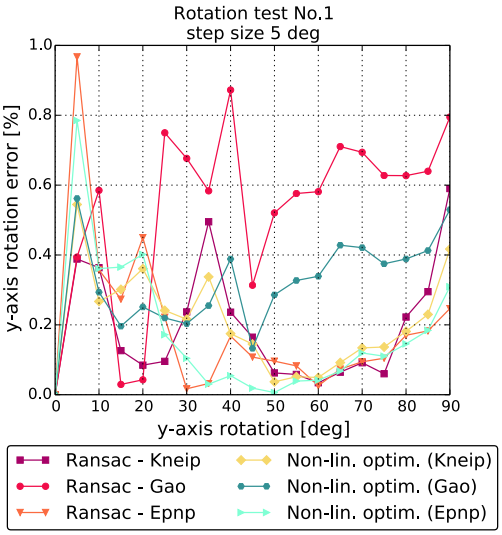
(a) (b)

(c) (d)

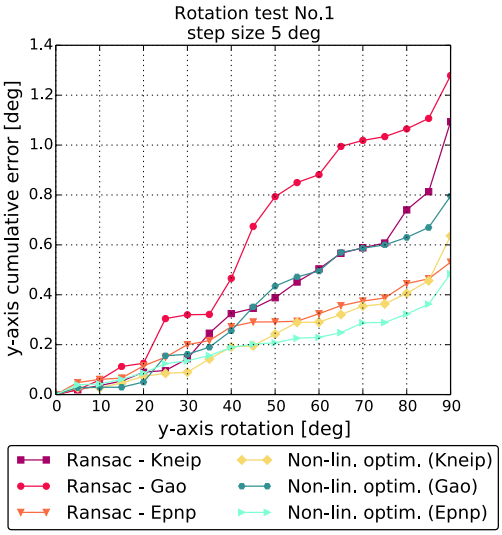
Figure 6.14: Cumulative errors calculated by adding the absolute errors for some different step sizes on the translation sequence No.2.



(a) Error relative to the ground truth.

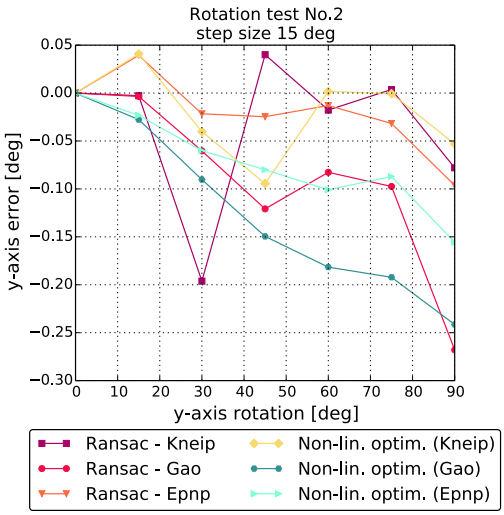


(b) Percentage error.

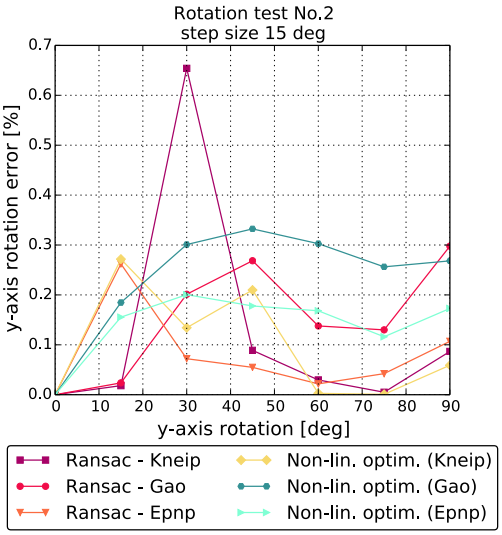


(c) Cumulative error.

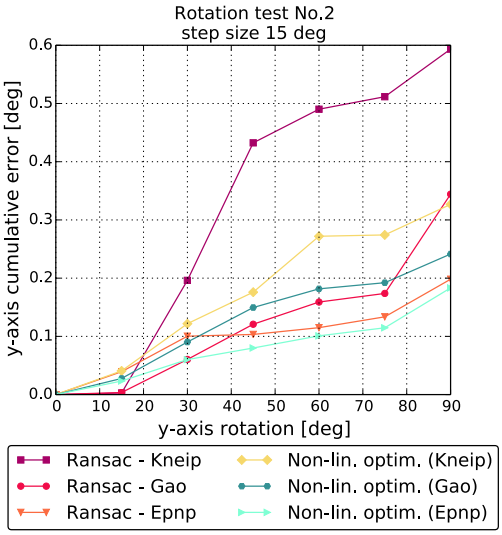
Figure 6.15: A comparison between different algorithm accuracies on the rotation sequence No.1. Graph 6.15a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.15b displays this percentage error; finally 6.15c shows the error cumulated at each step.



(a) Error relative to the ground truth.

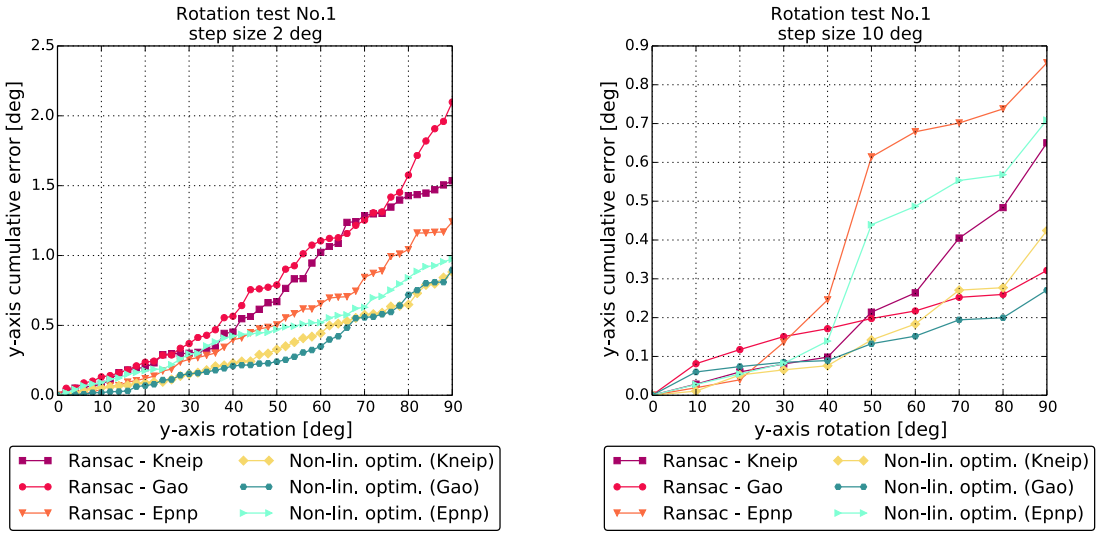


(b) Percentage error.



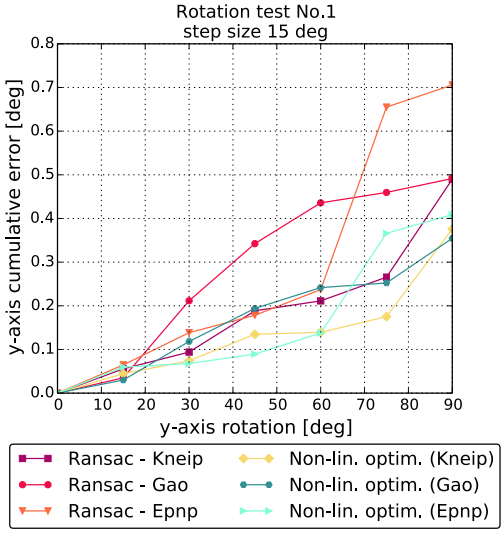
(c) Cumulative error.

Figure 6.16: A comparison between different algorithm accuracies on the rotation sequence No.2. Graph 6.16a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.16b displays this percentage error; finally 6.16c shows the error cumulated at each step.

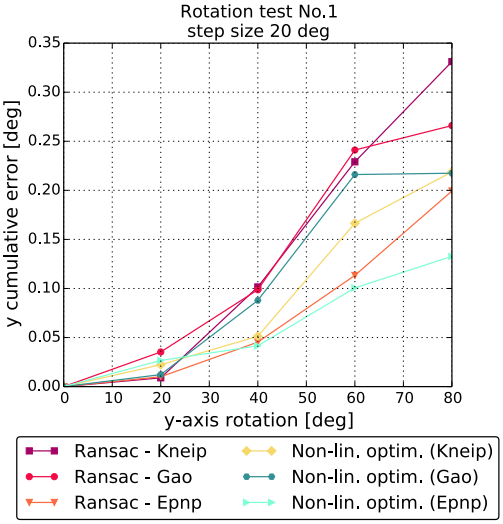


(a)

(b)



(c)



(d)

Figure 6.17: Cumulative errors calculated by adding the absolute errors for some different step sizes on the rotation sequence No.1.

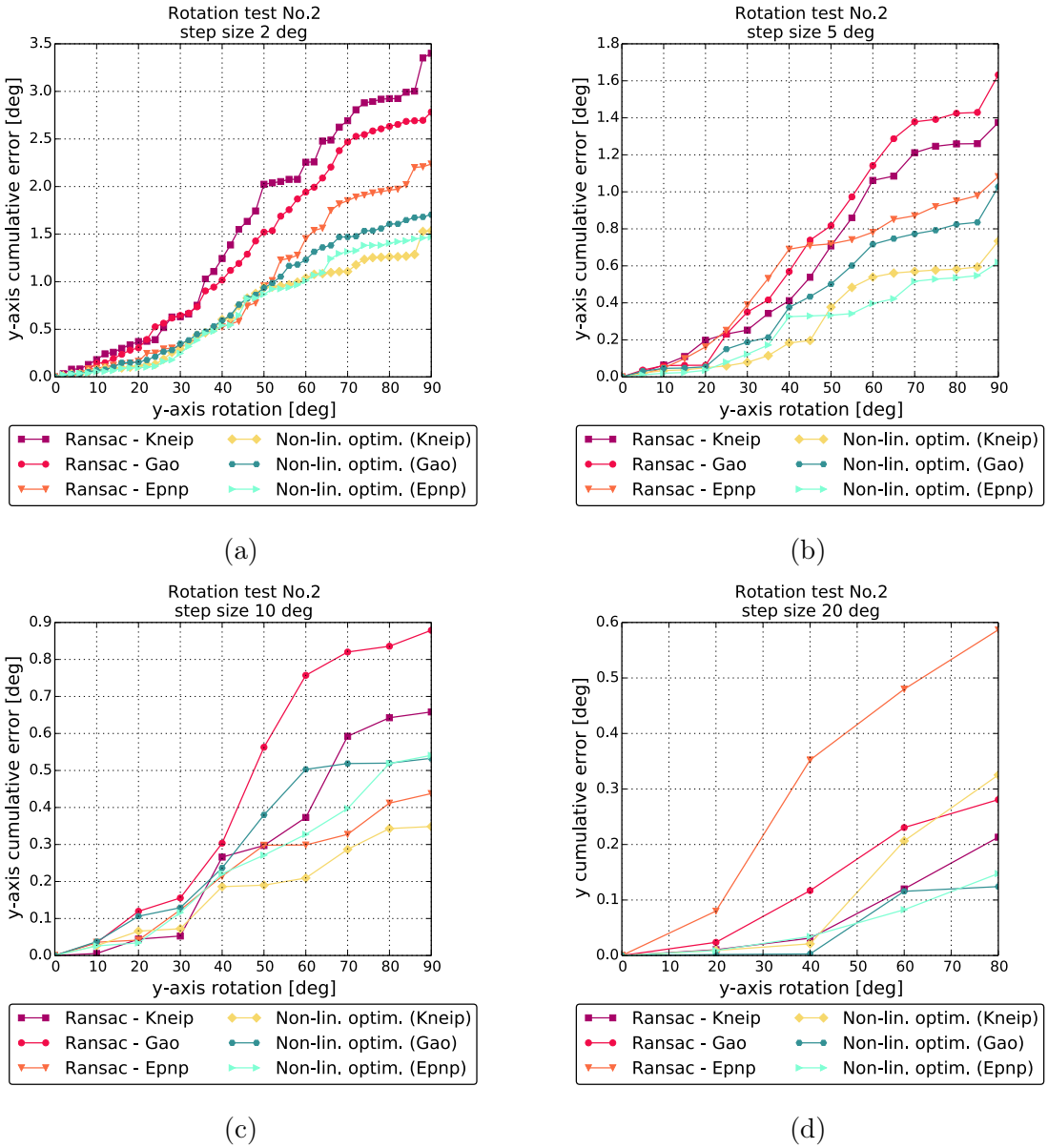
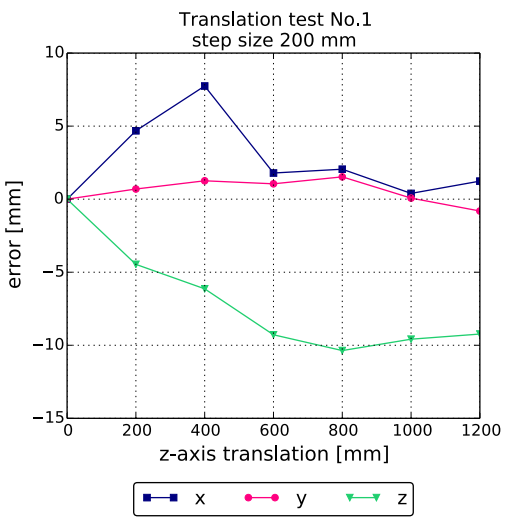
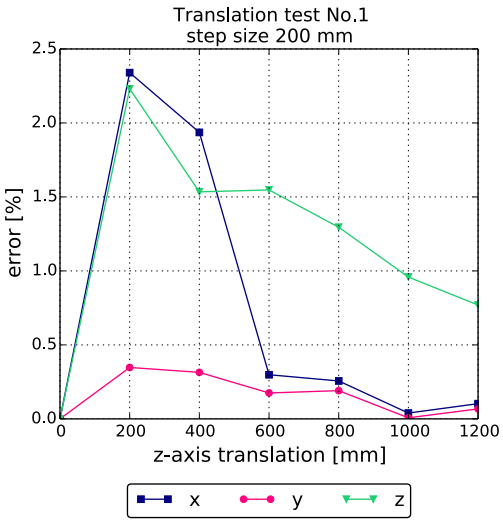


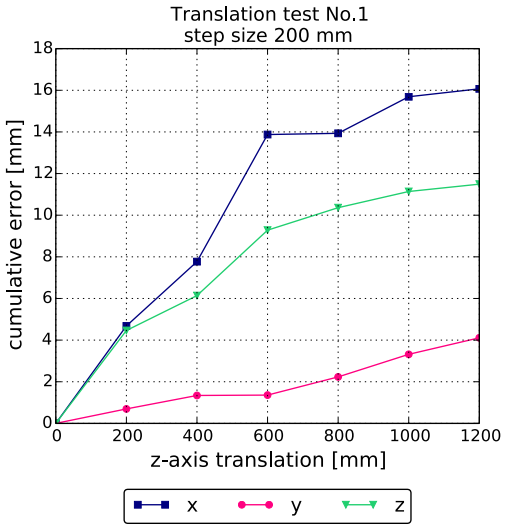
Figure 6.18: Cumulative errors calculated by adding the absolute errors for some different step sizes on the rotation sequence No.2.



(a) Error relative to the ground truth.

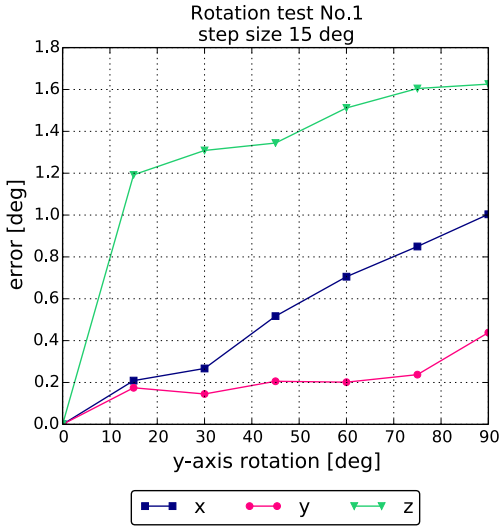


(b) Percentage error.

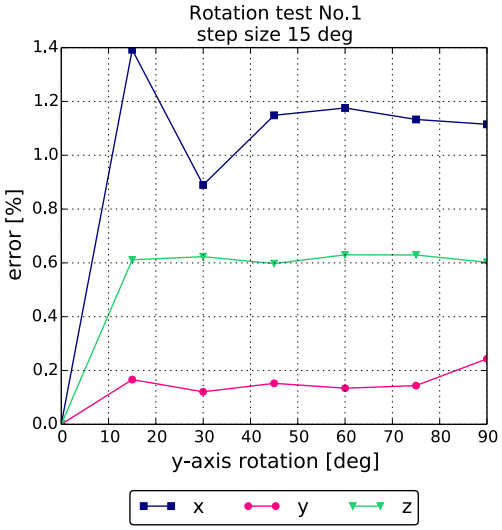


(c) Cumulative error.

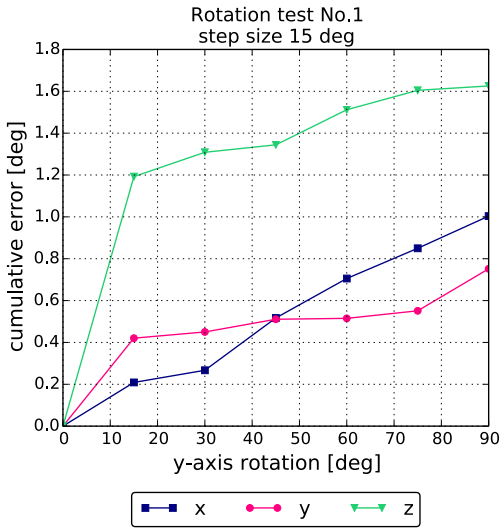
Figure 6.19: Illustration of the error on the three axis on the translation sequence No.1. Graph 6.19a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.19b displays this percentage error; finally 6.19c shows the error cumulated at each step.



(a) Error relative to the ground truth.



(b) Percentage error.



(c) Cumulative error.

Figure 6.20: Illustration of the error on the three axis on the rotation sequence No.1. Graph 6.20a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.20b displays this percentage error; finally 6.20c shows the error cumulated at each step.

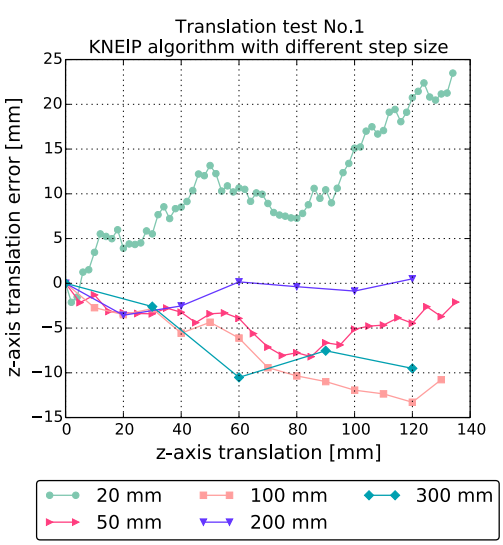
6.3 Acquisition frequency

The employed acquisition frequencies (i.e., $10\,mm$ for the translation tests and $0.5\,deg$ for the rotation ones) are relatively high for the slow motion of the vehicle. The visual odometry algorithm can use every acquired image or it can skip one or more frames every motion evaluation. In this way different motion step sizes were compared. Figures 6.21 - 6.24 shows the Kneip and the Gao algorithms on the translation and rotation sequences with different step sizes.

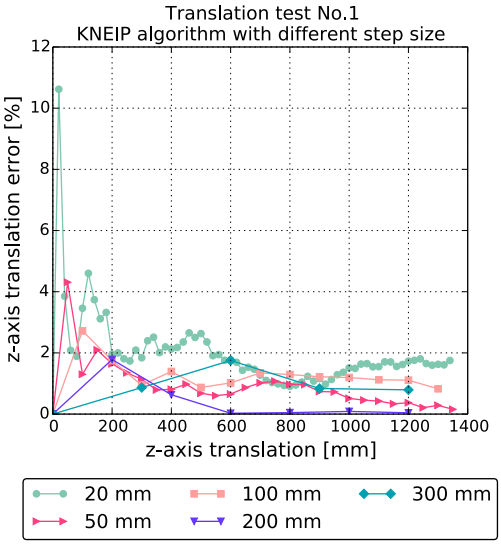
We can note that the accuracy improves as the step size between two image pairs increases despite the visual odometry algorithm estimates and integrates sequential frame-to-frame motions. Regarding the translation sequence, the precision improves increasing the step size from 20 to 200 mm and then slightly worsen with the 300 mm step, while for the rotation sequence the accuracy gets considerably better when moving from 3° step size to 15°.

We can clearly deduce that the algorithm introduces a constant error at each step and thus the cumulative error increases for small step sizes. Moreover, the visual odometry algorithm must have a sufficient image overlap in order to match features and then the step size shouldn't become too large.

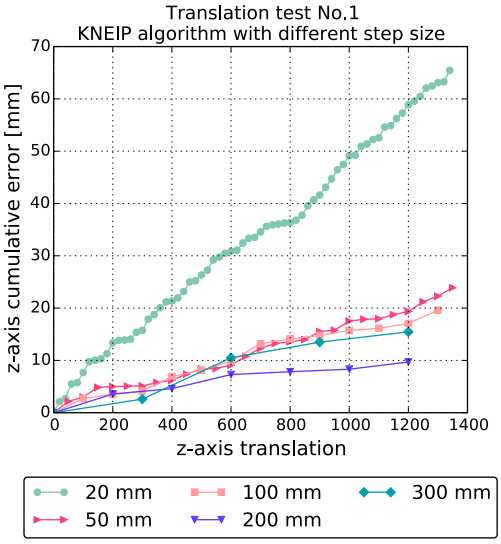
As mentioned in section 6.2 we also noted that if the step size is too small, e.g. $10\,mm$ or $20\,mm$, RANSAC iterations may exceed the limit and the result is not reliable.



(a) Error relative to the ground truth.

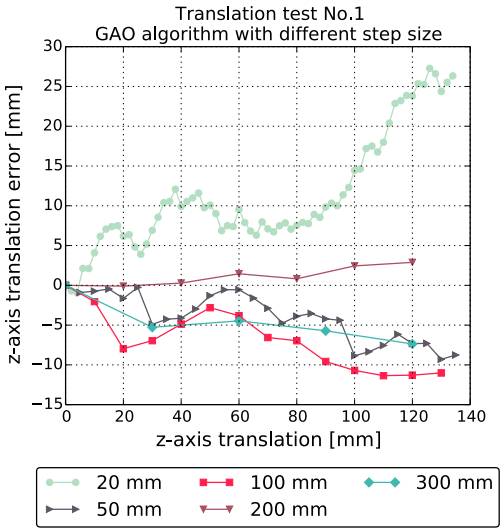


(b) Percentage error.

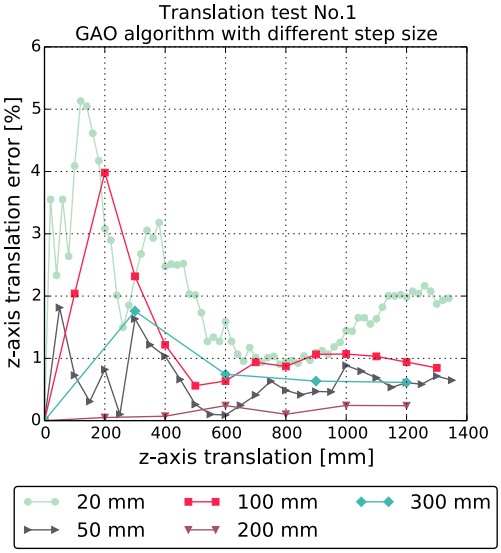


(c) Cumulative error.

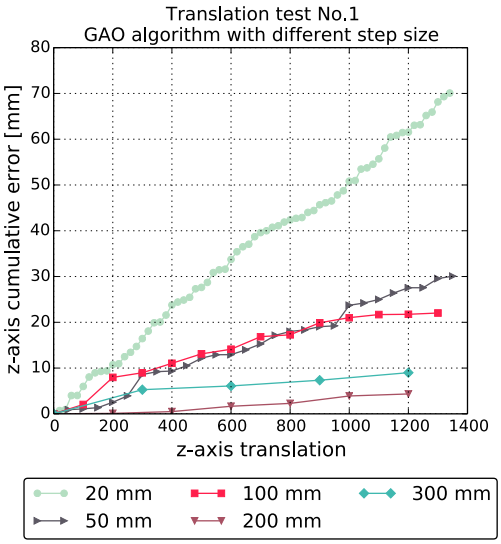
Figure 6.21: Illustration of Kneip algorithm with different step sizes on translation sequence No.1. Graph 6.21a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.21b displays this percentage error; finally 6.21c shows the error cumulated at each step.



(a) Error relative to the ground truth.

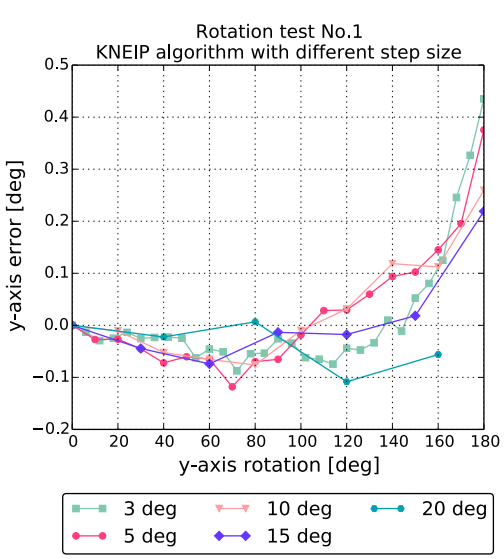


(b) Percentage error.

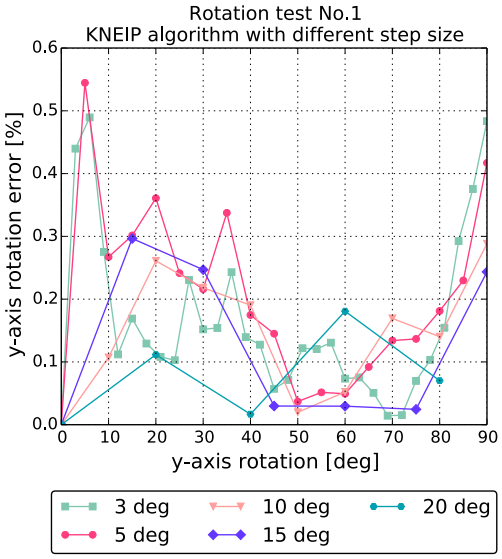


(c) Cumulative error.

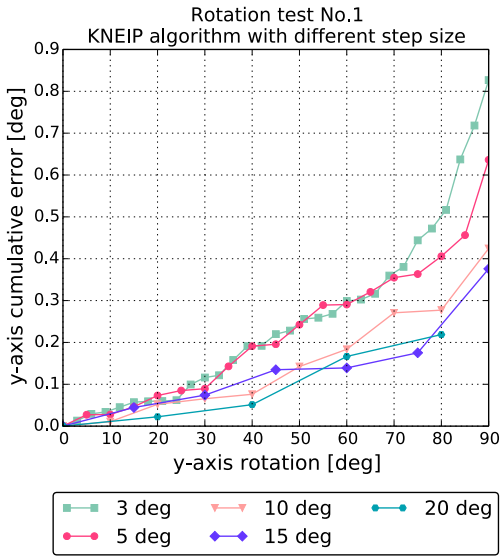
Figure 6.22: Illustration of Gao algorithm with different step sizes on translation sequence No.1. Graph 6.22a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.22b displays this percentage error; finally 6.22c shows the error cumulated at each step.



(a) Error relative to the ground truth.

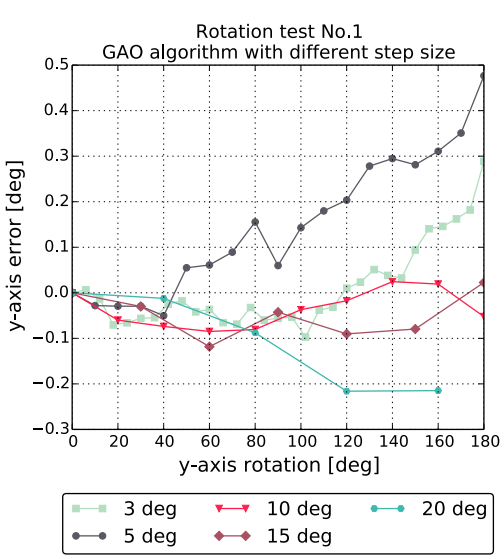


(b) Percentage error.

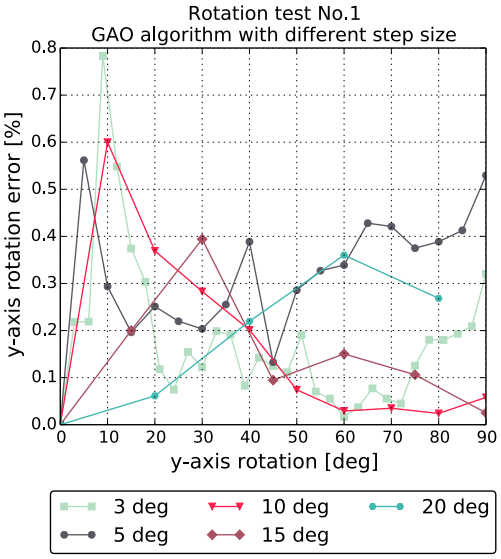


(c) Cumulative error.

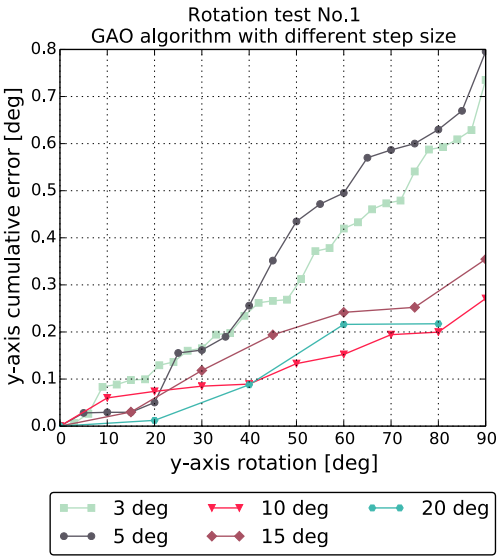
Figure 6.23: Illustration of Kneip algorithm with different step sizes on rotation sequence No.1. Graph 6.21a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.21b displays this percentage error; finally 6.21c shows the error cumulated at each step.



(a) Error relative to the ground truth.

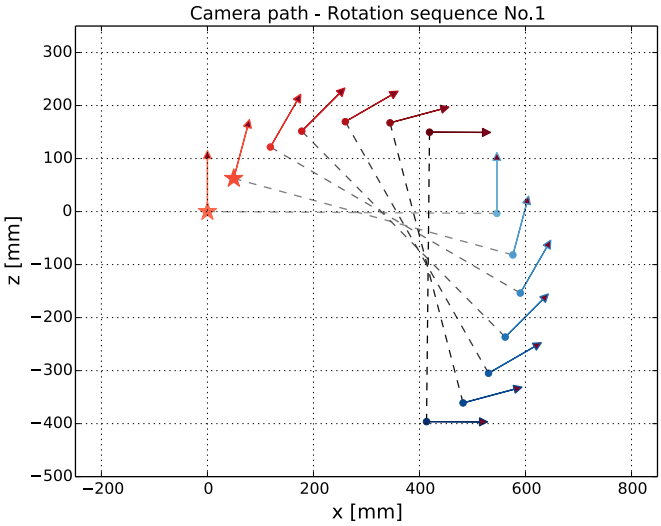


(b) Percentage error.



(c) Cumulative error.

Figure 6.24: Illustration of Gao algorithm with different step sizes on rotation sequence No.1. Graph 6.22a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.22b displays this percentage error; finally 6.22c shows the error cumulated at each step.



(a) Stereo-camera path.

(b)

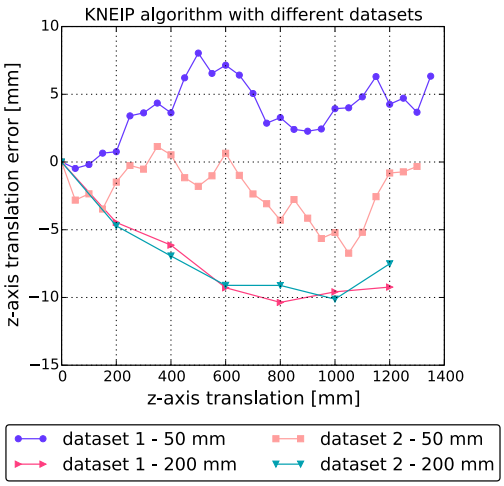
(c)

Figure 6.25: Example of images of the rotation sequence and computed path: the two locations marked in figure 6.25a indicate the position of the left camera which capture the images 6.25b and 6.25c.

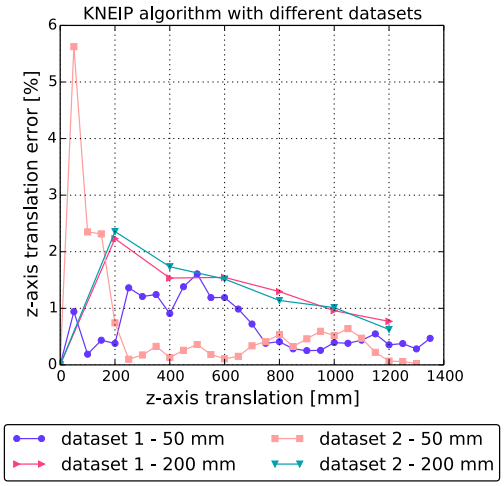
6.4 Datasets

Let us finally observe how the motion estimation accuracy change according to the sequence considered. In our evaluation, four different datasets of stereo frames are adopted. As described in chapter 5, we tested two translation and two rotation sequences. These datasets are collected in different laboratory positions, with changes on scale, viewpoints and illumination conditions.

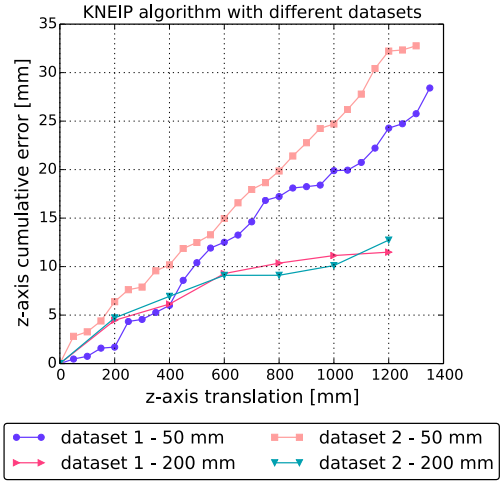
Figures 6.26 - 6.29 display the obtained precision. As can be seen the results are comparable and stable, demonstrating the robustness of the algorithm.



(a) Error relative to the ground truth.

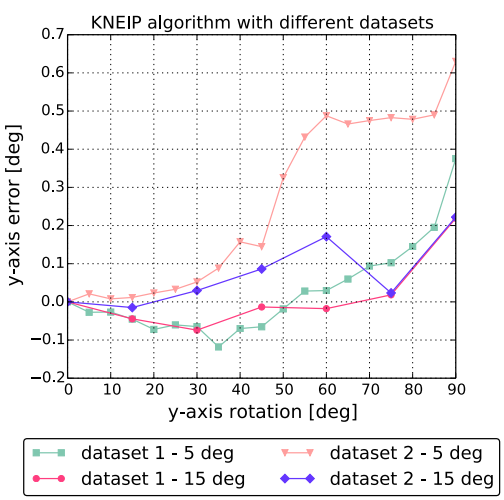


(b) Percentage error.

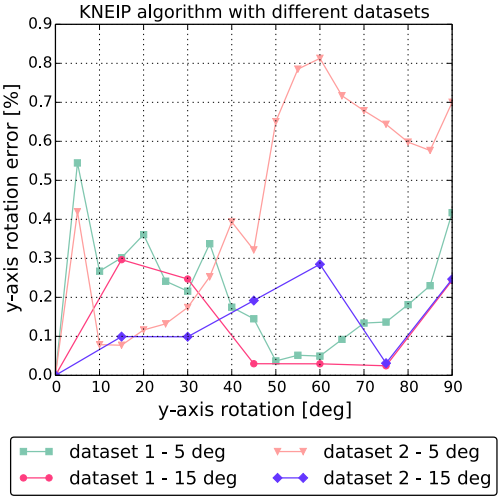


(c) Cumulative error.

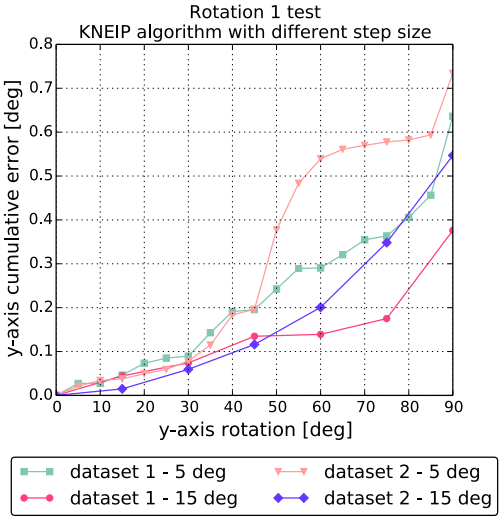
Figure 6.26: Illustration of Kneip algorithm with different translation sequences. Graph 6.26a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.26b displays this percentage error; finally 6.26c shows the error cumulated at each step.



(a) Error relative to the ground truth.

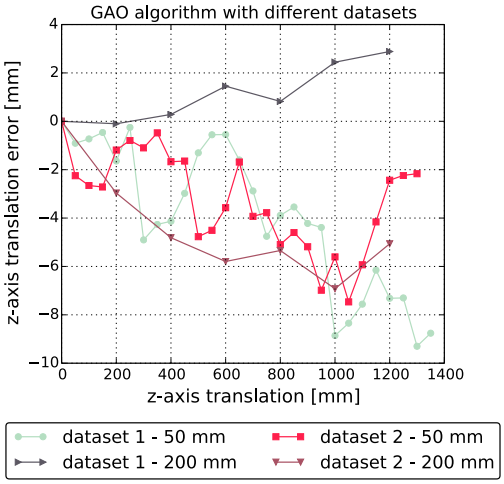


(b) Percentage error.

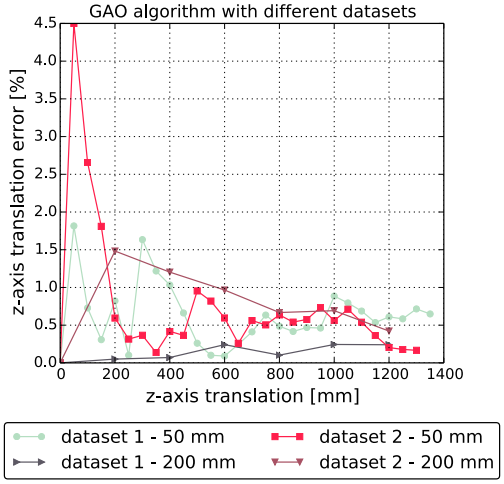


(c) Cumulative error.

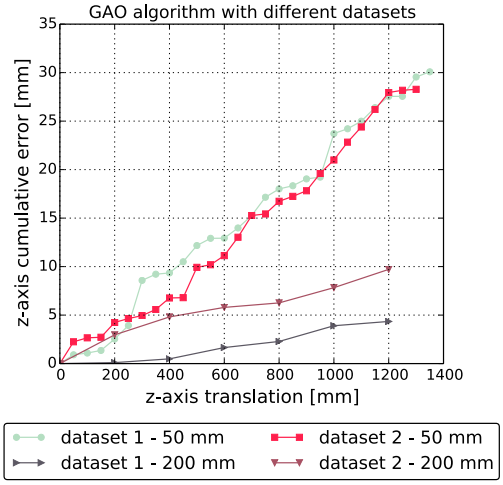
Figure 6.27: Illustration of Kneip algorithm with different rotation sequences. Graph 6.27a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.27b displays this percentage error; finally 6.27c shows the error cumulated at each step.



(a) Error relative to the ground truth.

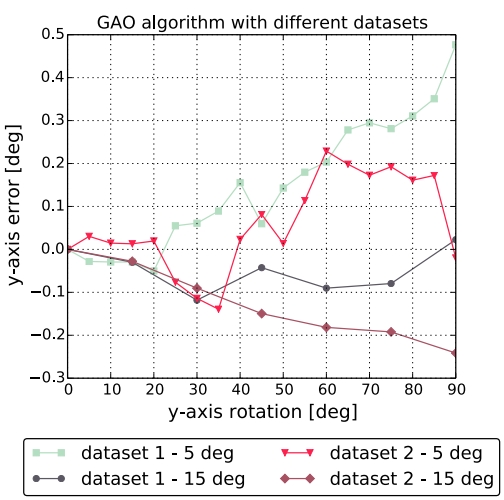


(b) Percentage error.

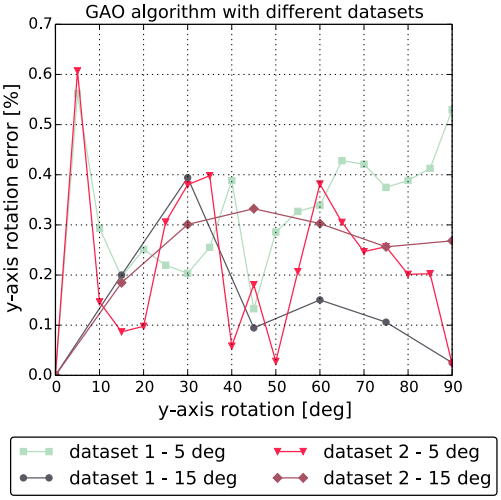


(c) Cumulative error.

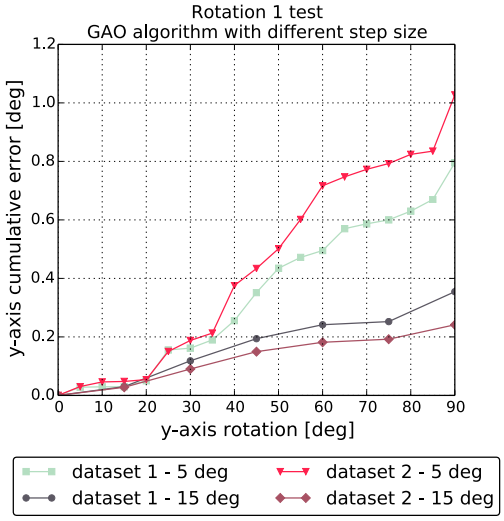
Figure 6.28: Illustration of Gao algorithm with different translation sequences. Graph 6.28a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.28b displays this percentage error; finally 6.28c shows the error cumulated at each step.



(a) Error relative to the ground truth.



(b) Percentage error.



(c) Cumulative error.

Figure 6.29: Illustration of Gao algorithm with different rotation sequences. Graph 6.29a shows the error computed as the difference between the estimated position and the correct one in the ground truth, while 6.29b displays this percentage error; finally 6.29c shows the error cumulated at each step.

Chapter 7

Conclusions

In the present work the attention is focused on visual odometry, that is an important technique in the robotic field. The low cost, small size, lower power needs, and high information content of modern cameras make them attractive sensors to address the problem of motion estimation.

We presented a software that can determine three-dimensional motion of a vehicle incrementally using the data collected by a stereo-camera.

The developed class implements a visual odometry algorithm using a 2D-to-3D method. Different motion estimation algorithms have been considered (i.e., Kneip, Gao and EPnP) inside the RANSAC outlier rejection scheme that allows to ensure accurate motion estimation at each step in the presence of outliers. A non-linear optimization has also been applied. We also reviewed a variety of detector and descriptor formulations and sensitivity of solutions to environment conditions and frequency acquisition.

The algorithm presented has been thoroughly tested and has shown performance that is promising for real-time applications. For example, the NVIDIA Jetson TK1 can process two 1020×543 px images and obtain the 3D vehicle pose in about $0.6\,s$. Experiments on hundreds of real stereo pairs have demonstrate an high degree of reliability and an accuracy of better than 1% over $1350\,mm$ of travel and better than 0.5% over $90°$.

We presented the capabilities and limitations of pure visual odometry. One of the disadvantages is related to the fact that visual odometry accumulates error as

the robot moves, so that some periodic update is beneficial. Stereo visual odometry algorithm described is therefore intended to augment rather than replace the sensors, and to work with higher-level pose estimators that fuse data from multiple sources. However this integration is beyond the purpose of this work and is not taken into account.

As described the experimental tests were performed inside our laboratory which is a closed environment. We should evaluate the robustness of these techniques using real rover images captured in rocky terrain, similar to the surface that a rover would encounter on Mars. A future development has been described in 5.1 and involves the application of the software on a fully functional terrain rover. In this way we could appraise the software performance using real rover images captured in outdoor environment with the rover undergoing six degree-of-freedom motion.

Bibliography

[1] D.Nister, O.Naroditsky, J.Bergen. Visual odometry. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 652–659, 2004.

[2] D.Scaramuzza, F.Fraundorfer. Visual odometry. Part I: The First 30 Years and Fundamentals. *IEEE Robotics & Automation Magazine*, pages 80–92, December 2011.

[3] H.Moravec. *Obstacles avoidance and navigation in the real world by a seeing robot rover.* PhD thesis, Stanford Univ., Stanford, CA, 1980.

[4] Y.Cheng, M.W.Maimone, L.Matthies. Visual odometry on the Mars Exploration Rovers. In *SMC*, pages 903–910. IEEE, 2005.

[5] D. Nistér, O.Naroditsky, J.R.Bergen. Visual odometry for ground vehicle applications. *J. Field Robotics*, 23(1):3–20, 2006.

[6] M.Agrawal, K.G.Konolige. Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS. In *ICPR*, pages III: 1063–1068, 2006.

[7] C.F.Olson, L.Matthies, M.Schoppers, M.W.Maimone. Rover navigation using stereo ego-motion. *Robotics and Autonomous Systems*, 43(4):215–229, 2003.

[8] N.Sünderhauf, K.Konolige, S.Lacroix, P.Protzel. Visual Odometry Using Sparse Bundle Adjustment on an Autonomous Outdoor Vehicle. In Paul Levi, Michael Schanz, Reinhard Lafrenz, and Viktor Avrutin, editors, *AMS*, Informatik Aktuell, pages 157–163. Springer, 2005.

[9] A.Howard. Real-time stereo visual odometry for autonomous ground vehicles. pages 3946–3952. IEEE, 2008.

[10] T.S.Huang, A.N.Netravali. Motion and Structure from Feature Correspondences: A review. *Proceedings of IEEE*, 82(2):252–268, feb 1994.

[11] Y.Ma, S.Soatto, J.Kosecka, S.S.Sastry. *An Invitation to 3-D Vision: From Images to Geometric Models.* Springer-Verlag, 2005.

[12] OpenCV: Open-source computer vision library, available at `http://opencv.org`.

[13] D.Scaramuzza, F.Fraundorfer. Visual odometry. Part II: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, pages 78–90, June 2012.

[14] N.Govender. Evaluation of feature detection algorithms for structure from motion. In *3rd Robotics and Mechatronics Symposium (ROBMECH 2009)*, 2009.

[15] K.Mikolajczyk, C.Schmid. Scale & affine invariant interest point detectors. 2004.

[16] Y.Jiang, Y.Xu, Y.Liu. Performance evaluation of feature detection and matching in stereo visual odometry. *Neurocomputing*, 120:380–390, 2013.

[17] D.G.Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[18] H.Bay, T.Tuytelaars, L.J.Van Gool. SURF: Speeded Up Robust Features. In *ECCV*, pages I: 404–417, 2006.

[19] E.Rosten, T.W.Drummond. Machine Learning for High-Speed Corner Detection. In *ECCV*, pages I: 430–443, 2006.

[20] M.Calonder, V.Lepetit, C.Strecha, P.Fua. BRIEF: Binary Robust Independent Elementary Features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *ECCV (4)*, volume 6314 of *Lecture Notes in Computer Science*, pages 778–792. Springer, 2010.

[21] M.Pertile, M.Magnabosco, S.Debei. Calibration of a vision-based system for displacement measurement in planetary exploration space mission. *Journal of Physics: Conference Series*, 238(1), 2010.

[22] M.A.Fischler, R.C.Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[23] D.Nister. A minimal solution to the generalised 3-point pose problem. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 560–567, 2004.

[24] R.M.Haralick, C.N.Lee, K.Ottenberg, M.Nolle. Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem. *International Journal of Computer Vision*, 13(3):331–356, December 1994.

[25] L.Kneip, P.T. Furgale, R.Siegwart. Using multi-camera systems in robotics: Efficient solutions to the NPnP problem. In *ICRA*, pages 3770–3776. IEEE, 2013.

[26] R.M.Haralick, C.N.Lee, K.Ottenberg, M.Nolle. Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem. *International Journal of Computer Vision*, 13(3):331–356, dec 1994.

[27] J.A.Hesch, S.I.Roumeliotis. A direct least-squares (DLS) method for pnP. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 383–390. IEEE, 2011.

[28] Laurent Kneip, Paul Timothy Furgale. OpenGV: A unified and generalized approach to real-time calibrated geometric vision. pages 1–8. IEEE, 2014.

[29] Gao, Hou, Tang, Cheng. Complete solution classification for the perspective-three-point problem. *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25, 2003.

[30] L.Kneip, D.Scaramuzza, R.Siegwart. A novel parameterization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 2969–2976, 2011.

[31] V.Lepetit, F.Moreno-Noguer, P.Fua. EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision (IJCV)*, 81:155–166, February 2009.

[32] P.H.S.Torr, A.Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[33] OpenGV: A library for solving calibrated central and non-central geometric vision problems, available at `http://laurentkneip.github.io/opengv/index.html`.

[34] M.Pertile, S.Chiodini, S.Debei, E.Lorenzini. Laboratory calibration and comparison of three visual odometry systems. *IX Congresso MMT*, pages 48–55, 2014.

[35] M.Pertile, S.Chiodini, S.Debei. Comparison of visual odometry systems suitable for planetary exploration. *Journal of Physics: Conference Series*, pages 232–237, 2014.