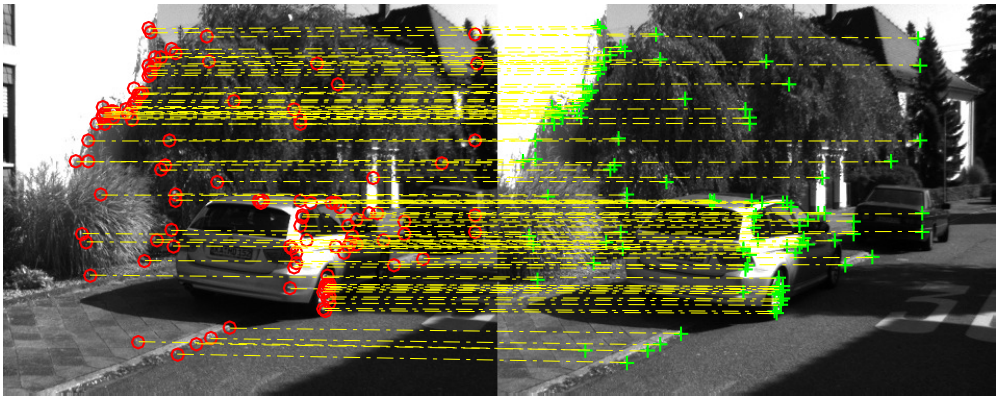

Visual Odometry for Road Vehicles Using a Monocular Camera

A comparison of Feature Matching and Feature Tracking using FAST, SURF, and SIFT detectors

Master's thesis in Systems, Control and Mechatronics

HENRIK BERG & RAMAN HADDAD

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:EX105

Visual Odometry for Road Vehicles Using a Monocular Camera

A comparison of Feature Matching and Feature Tracking using FAST, SURF, and SIFT detectors

HENRIK BERG & RAMAN HADDAD

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2016

Visual Odometry for Road Vehicles Using a Monocular Camera
A comparison of Feature Matching and Feature Tracking using FAST, SURF, and SIFT detectors
HENRIK BERG, RAMAN HADDAD

Supervisor: Artur Chodorowski, Signals and Systems & Robert Björkman, Semcon
Examiner: Fredrik Kahl, Signals and Systems

Master's Thesis 2016:EX105
Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: FAST features matched in two images from KITTI dataset using MATLAB.

Typeset in LaTeX
Gothenburg, Sweden 2016

Visual Odometry for Road Vehicles Using a Monocular Camera
A comparison of Feature Matching and Feature Tracking using FAST, SURF, and SIFT detectors
HENRIK BERG, RAMAN HADDAD
Department of Signals and Systems
Chalmers University of Technology

Abstract

As technology keeps advancing, autonomous vehicles are more than a possibility, they are inevitable. An inevitability offering both security and comfort for passengers as well as for others in traffic. However there are still certain predicaments to be investigated, one of those challenges involves an accurate positioning system of the vehicle, especially when GPS is not available. One method dealing with this issue is to incrementally estimate motion using images taken by a digital camera, an area know as Visual Odometry. However Visual Odometry algorithms can be implemented in many different ways, hence there is a need to evaluate the performance on real data.

The aim of the thesis is to compare the different single camera Visual Odometry algorithms with respect to vehicle trajectory (the rotational and translational error) and the execution time. The algorithms differed with respect to the used feature detectors and descriptors and the feature matching/tracking method. The investigated feature detectors, descriptors and tracking were based on FAST/ORB, SIFT and SURF methods and compared with Kanade-Lucas-Tomasi (KLT) tracking. The relative motion between two consecutive images was estimated from 2D–*to*–2D feature correspondence and Nister's five-point algorithm.

The algorithms are implemented in C++/OpenCV and tested on three image sequences in different environments from the public KITTI dataset. The obtained results are compared to ground truth data from a highly accurate GPS. The results show that the investigated methods are able to estimate the ego-motion with an average translation error of <7 % and a rotation error of <0.02 deg/m. The best results, with respect to rotational and translational error, are obtained using feature matching of SIFT features along with the corresponding descriptor. The results also show that the feature tracking using KLT provides a faster algorithm than feature matching. However this comes at the cost of reduced accuracy, which is something that also holds for the choice of detectors and descriptors.

Keywords: Visual Odometry, FAST, SURF, SIFT, RANSAC, Monocular, KLT.

Acknowledgements

We would like to thank our supervisor at Chalmers University of Technology, Artur Chodorowski, for the interesting discussions and feedback during the thesis. We would also like to thank our examiner Fredrik Kahl for the support despite the late time frame of this thesis. And at Semcon, we would like to thank our supervisor Robert Björkman for involving us in the Born to Drive project, along with Sofia Hjalmarsson for her involvement.

Henrik Berg and Raman Haddad, Gothenburg, August 2016

Contents

List of Figures

List of Tables

List of Acronyms

AGV Automatic Guided Vehicles.

BA Bundle Adjustment.
BRIEF Binary Robust Independent Elementary Features.

DoF Degrees of Freedom.
DoG Difference of Gaussians.

EKF Extended Kalman Filter.

FAST Feature from Accelerated Segment Test.
FLANN Fast Library for Approximate Nearest Neighbours.
FPGA Field-programmable gate array.

GPS Global Positioning System.
GPU Graphics Processing Unit.

KITTI Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago.
KLT Kanade-Lucas-Tomasi.

LIDAR Light Detection And Ranging.

NN Nearest Neighbour.

ORB Oriented FAST and Rotation-Aware BRIEF.

PF Particle Filter.

RANSAC Random Sample Consensus.
RGB-D Red Green Blue - Depth.

SFM Structure From Motion.
SIFT Scale Invariant Feature Transform.
SLAM Simultaneous Localization and Mapping.
SURF Speeded Up Robust Features.
SVD Singular Value Decomposition.

VO Visual Odometry.

1
Introduction

1.1 Background

Autonomous vehicles are of substantial focus today with vehicle manufacturers competing for headlines, competence and technological capabilities. The Gothenburg region have interest in developing both competence within the area of autonomous driving and also showcase that the region is ready for the next generation of technological challenges of autonomous driving.

In order to meet these goals, the project "Born to drive" [1] was started. It will operate as a consortium of local companies to work together in order to achieve the task of developing a system to control a vehicle. The vehicle shall operate autonomously within certain boundaries to showcase the potential of the technology and know-how.

Today someone drives the vehicles from factory to the designated parking area. The project aims to use autonomous driving within the factory area, from the drop-plan of factory completed vehicles to a designated parking area in order to improve efficiency of vehicle logistics.

To move the vehicle autonomously and safe it is required that the vehicle is aware of the ego-motion, its 3D-movement in a static environment, along with potential obstacles in the surrounding. To comply with these requirements the vehicle is equipped with the sensors shown in Figure 1.1

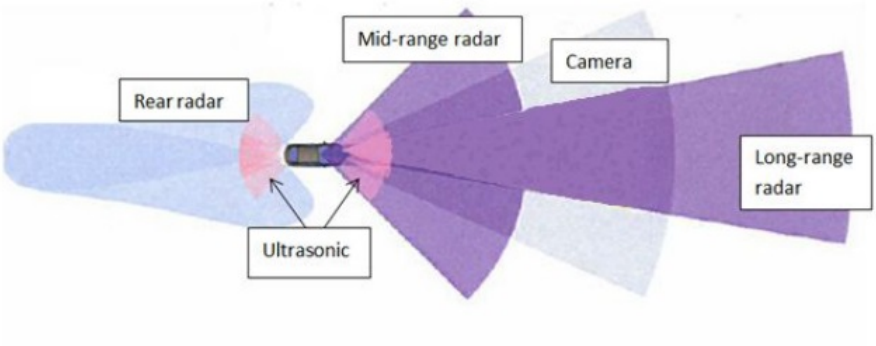

Figure 1.1: Sensors available for the target vehicle, note that this is only an illustrative image

The project is divided into eight different packages which focuses on different tasks. The thesis is done in the second package “Research, development and implementation of new functionality”.

1.2 Purpose

The purpose of this thesis is to examine and evaluate how the images taken by a single digital camera, mounted on a vehicle can establish the ego motion using Visual Odometry. The main focus will be on how the usage of different feature detectors and descriptors will affect the accuracy of the position in varying environments.

1.3 Delimitations

The set of existing hardware is the hardware at disposal, thus there is one camera available to work with along with the velocity of the vehicle. Considering the fact that only one camera is available, the scale ambiguity (the translation vector is incorrect by an unknown factor) of the motion is resolved using the speed of the vehicle.

Even though the intention of Visual Odometry is to be run in real-time, no code optimization is done in this project since it is outside the scope of this thesis. The execution time of the different methods are only to be interpreted relative to each other.

As we will see further down in this thesis, two different methods for motion estimation is examined 2D–*to*–2D and 3D–*to*–2D. However only the former is implemented, which means that no triangulation is required.

The implemented algorithm requires that the input images are all in grayscale. The images must also be rectified which means that they are compensated for lens distortion. However the KITTI dataset provide images that already fulfill these requirements, hence the algorithm does not manipulate the images in any way.

All the images must be in grayscale and compensated for lens distortion.

1.4 Structure of the thesis

Chapter 2, Theory, gives a thorough explanation about the problems with Visual Odometry and how the tools used to solve the problem function. It is followed by the chapter Implementation, which describes how the algorithms presented in chapter 2, are pieced together in the Visual Odometry chain. Chapter 4, Results, presents the achieved positioning accuracy with different combinations of algorithms in different environments. The work done in the thesis is summed up in Chapter 5, Discussion, where the authors reflect over the used method, achieved results and further improvements.

1.5 Different methods of positioning

Today there exists multiple ways of calculating the position of a moving vehicle. A popular method to position Automatic Guided Vehicles (AGV) is to triangulate the position with well placed laser beacons [2, 3]. A downside with this method is that it requires additional expensive hardware, beacons at the parking area plus an optional antenna on the vehicle. The benefit is that the calculated position is very accurate. Data received from testing in [3], gives a positioning error of $\pm 10\,\text{mm}$ and angle error of $\pm 0.3\,\text{deg}$.

A more basic method is to use wheel encoders combined with an Extended Kalman Filter (EKF) [4]. However, the method suffers from the need of an exact wheel radius and that the road conditions are good enough to have wheel rotations with no slip. This parameter and environmental dependant method makes it hard to design an algorithm which does not require daily maintenance.

Visual Odometry (VO) [5, 6] is a special case of Structure From Motion (SFM) [7, 8]. The method uses an image sequence to detect the movement of features points between each frame. The movements are used in order to calculate the pose and motion of the camera, thus giving the position. With the improvement of both algorithms and hardware, the method is realizable in real-time [9]. There exists two different approaches, with one camera (monocular) [9] and two cameras (stereo) [10, 11]. Stereo VO is the branch of VO in which most research has been done. By using two cameras the 3D structure of the environment is given by an image pair instead of sequential images, resulting in that the drift is less than compared to the monocular method. Additionally, 3D features are computed in absolute scale due to the embedded depth given by the image pair. In Monocular VO, the scale cannot be calculated with a single image. Additional information is required to calculate the relative scale and thus proper motion. In paper [12] the authors have evaluated several different detectors and descriptors such as FAST, SURF, SIFT, and BRIEF. The have compared these different methods in terms of speed, precision, and repeatability. They have also investigated how the combination of detectors and descriptors affect the accuracy and speed.

Another existing method is point cloud based VO, which is a mixture of mono or stereo VO combined with point cloud data. A simple implementation is described in [13]. This method needs a Light Detection And Ranging (LIDAR) sensor in order to generate the point cloud.

SLAM, Simultaneous Localization and Mapping [14, 15], positions the vehicle by mapping the surrounding environment at the same time as the method calculates the location of the sensors mounted on the vehicle. Two popular solutions are Particle Filter (PF) [16] and the Extended Kalman Filter [17]. One problem with this method is how to handle parts where the vehicle has already been at. This is called *Loop closure*, and this problem is usually solved by having an additional algorithm which checks for similar sensor inputs and refines the map accordingly. By

using Bundle Adjustment (BA), the camera poses can be optimized by minimizing the reprojection error that occurs when triangulating the same 3D-point in different images. An evaluation of feature detection algorithms for SFM can be read about here [18]. The authors build up a SFM framework using different feature detectors like KLT tracker, SURF, and SIFT, and tries to determine the accuracy and speed of these different algorithms.

2
Theory

This chapter provides the theory needed to understand the Visual Odometry problem. It explains how the algorithms function and how they provide the outputs that are used in steps further down the Visual Odometry chain.

2.1 Problem formulation

For a camera in an environment taking pictures, let the set of images taken at the discrete time instants k be denoted by $\mathbf{I}_{0:n} = \{I_0, \ldots, I_n\}$. Two camera positions at adjacent time instants $k-1$ and k are then related, see Figure 2.1, by the rigid body transformation $\mathbf{T}_{k,k-1} \in \mathbb{R}^{4\times4}$, defined as

$$\mathbf{T}_{k,k-1} = \begin{bmatrix} \mathbf{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{R}_{k,k-1} \in \mathbb{R}^{3\times3}$ is the rotation matrix, and $\mathbf{t}_{k,k-1} \in \mathbb{R}^{3\times1}$ is the translation

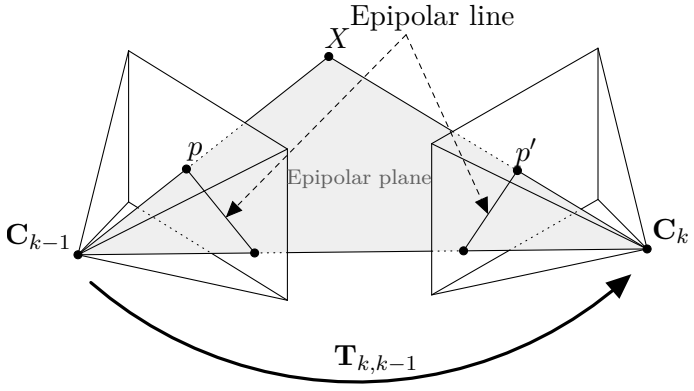

Figure 2.1: Epipolar constraints.

vector. For the sake of simplicity the notation $\mathbf{T}_k$ will be used instead of $\mathbf{T}_{k,k-1}$. Let then the set of all subsequent movements be denoted by $\mathbf{T}_{1:n} = \{\mathbf{T}_1, \ldots, \mathbf{T}_n\}$ and the camera poses denoted by $\mathbf{C}_{0:n} = \{\mathbf{C}_0, \ldots, \mathbf{C}_n\}$, the initial pose is arbitrary and can be set to $\mathbf{C}_0 = \mathbf{I}_{4\times4}$. The pose $\mathbf{C}$ has the same structure as $\mathbf{T}$, see (2.1), however it is relative to the initial pose $\mathbf{C}_0$. By concatenating all the transformations the camera pose can be calculated, hence $\mathbf{C}_n = \mathbf{C}_{n-1}\mathbf{T}_n^{-1}$. The position of the camera at time k, relative the initial position, is given by the last column of $\mathbf{C}_k$. The values in that column represent the x, y and z coordinates.

2.2 Pinhole camera model

There are several different camera models to choose from when working with visual odometry, e.g. catadioptric projection, spherical model for perspective, omnidirectional and perspective projection. However the most used model is the perspective projection [5]. The model assumes a *pinhole projection* system where the intersection of all light rays through the camera lens forms the image.

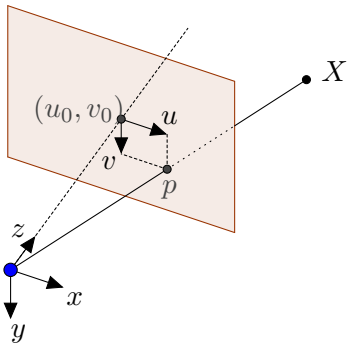

Figure 2.2: Pinhole camera projection.

Let the projection of a 3D-point $\mathbf{X} = (x, y, z)^\top$, on the xy-plane going through the point $(0, 0, f)^\top$, be denoted by the image coordinates $\mathbf{p} = (u, v)^\top$ as seen in Figure 2.2. The image coordinates are then related to the objects world coordinates by following relation

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{K}} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{2.2}$$

where λ is the depth factor, f is the focal length and u_0, v_0 are the coordinates of the projection center. These parameters are known as *intrinsic parameters* or *calibration matrix* and are given by the matrix $\mathbf{K}$. The calibration matrix can be found using a camera calibration tool from MATLAB for instance, in our case the matrix are obtained from the KITTI website. The camera matrix is required for the *five point solution* described in section 2.7.1. Important to note is that the intrinsic matrix in (2.2) assumes an ideal camera where the pixels are perfect squares. This is however not always true and in order to compensate for that, three other factors, s_x, s_y and s_φ, are introduced. The first two terms are called *scaling factors* and they define the size of a pixel. The last term is know as the *skew factor* and it is proportional to $\cot(\varphi)$, where φ is the angle between the axes u and v. Given these parameters the intrinsic matrix is rewritten as,

$$\mathbf{K} = \begin{pmatrix} fs_x & fs_\varphi & u_0 \\ 0 & fs_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.3}$$

2.3 Feature detection

The first step of the Visual Odometry algorithm, is to detect features within an image. A feature is a small part or region within the image, which differs from the pixels in its proximity. This difference is often described by colour and intensity, and the difference is noticed in edges and blobs. Pixels in an edge have a high gradient magnitude and stands out from pixels in the immediate neighbourhood. Two or more edges that coincide forms a corner and can easily be detected. Blobs on the other hand consists of many pixels with the same colour and intensity, grouped together that stands out from pixels surrounding the blob. A good feature detector is determined by the following properties [6]

1. Computational efficiency
2. Geometric invariance (rotation and scale)
3. Localization accuracy
4. Photometric invariance (illumination)
5. Repeatability (detect same features in consequent images)
6. Robustness (against blur/noise)

2.3.1 Harris corner detection

Harris corner detection is one of the earliest detectors made (1988), which is still used widely today. Although the algorithm is not implemented in this thesis, it gives a good insight in how many of the existing algorithms work. The basic idea behind Harris corner detection is to examine a point by looking at the point and the nearby pixels through a small window. By shifting the window in any direction from a given point, the average intensity of the window should change significantly if the point is a corner. Three possible scenarios are shown in Figure 2.3

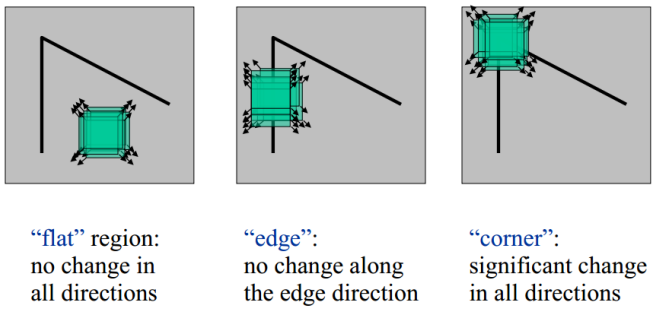

Figure 2.3: Harris corner detection.

Since the the window is shifted in all directions, the algorithm gives the same results if the image is rotated (rotation invariant). If the image is "zoomed out", then the algorithm might not give the same result as the original image. This is because the algorithm might interpret a curved edge in the original image as a corner when zoomed out. Thus the algorithm is non-invariant to scale. It is also computationally demanding due to many calculations of the window intensity average.

2.3.2 FAST

Feature from Accelerated Segment Test (FAST) is a corner detection algorithm which as the acronym suggests, is a quick detector. The algorithm uses a 16 pixel circle with a radius of 3 pixels around a pixel p in order to determine if p is a corner, see Figure 2.4.

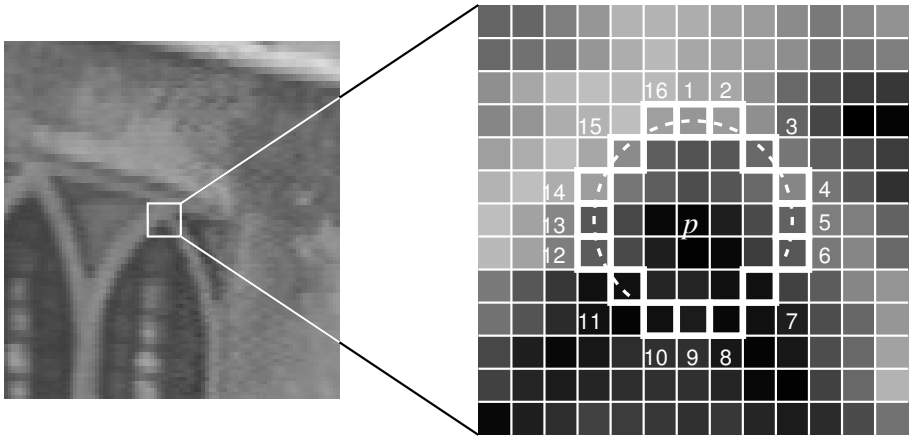

Figure 2.4: FAST corner detection, the arc shows 12 pixels which have a brighter intensity than p (from Rosten and Drummond, 2006).

As seen in the figure, pixels 11–16 and 1–6 have a higher intensity than the examined pixel p. This means that p is a corner and the location of the feature is saved. The segment test can be summarized in two conditions:

1. If there are N adjacent pixels in the ring that all have a brighter intensity than p plus a threshold, then p is a corner
2. If there are N adjacent pixels in the ring that all have a darker intensity than p minus a threshold, then p is a corner

Thus if either of the conditions is true, then p is classified as a corner. The parameter N is usually set to 12 in order to get a high quality feature detection. N could be set to 9 in order to achieve a faster detection, the trade-off is however accuracy. The algorithm is dependent on the intensity threshold, thus different environments might give different results and the threshold then needs to be tweaked for best performance. Since the algorithm uses a circle of pixels to determine if the examined pixel is in a corner, the algorithm is both scale and rotation invariant.

2.3.3 SIFT detector

The first step in the Scale Invariant Feature Transform (SIFT) algorithm is to smooth an input image by convolution with a Gaussian. This scale space function of an image is defined as

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{2.4}$$

where $I(x, y)$ is the input image and the Gaussian kernel $G(x, y, \sigma)$ is equal to

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{2.5}$$

The step is repeated with different σ, scaled by a factor k. This is done in order to compute the Difference of Gaussians (DoG)

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma). \tag{2.6}$$

The DoG has two important properties, it is an efficient computation and it "provides a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$" [19]. This means that features found by a DoG are more stable in comparison to other types of feature detection [20]. Once all the DoGs are computed for the current scale, called octave, the smoothed images are down-sampled by a factor of 2 and the process is repeated. The procedure is illustrated in Figure 2.5.

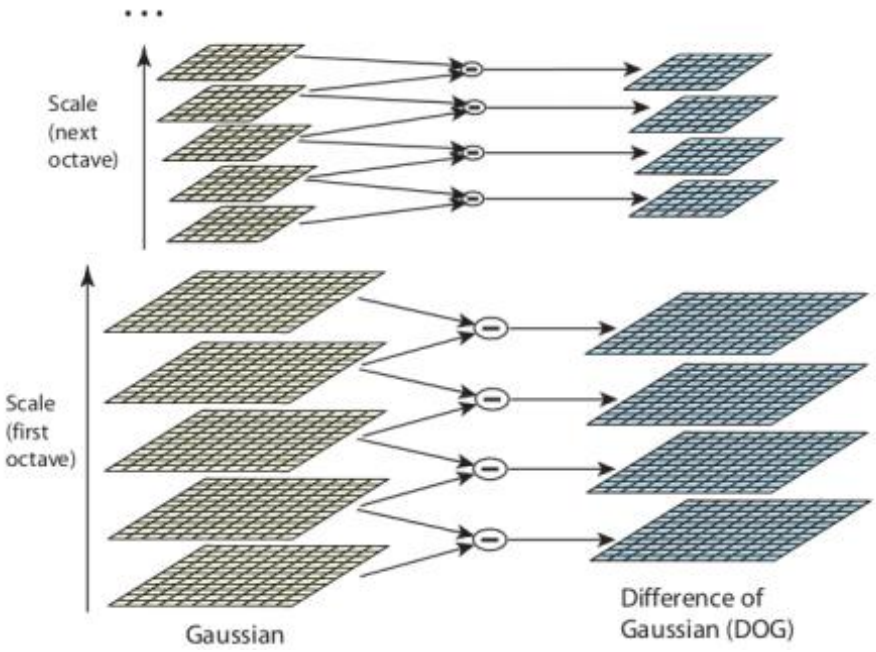

Figure 2.5: Computation of DoG (from Lowe, 2006).

When the DoGs are found for each octave, the DoGs are searched for local extrema. Each pixel is compared to the surrounding eight pixels and then compared to the pixels in the scale directly above and below, which is shown in Figure 2.6

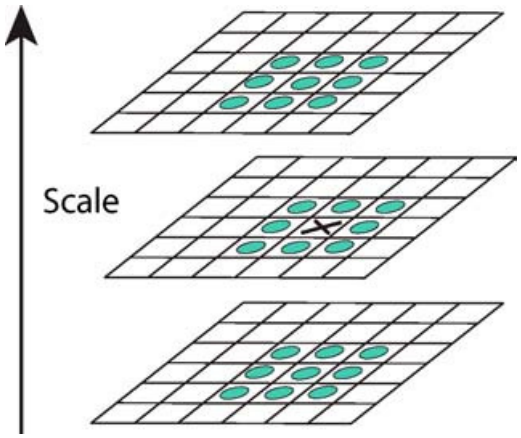

Figure 2.6: Detection of extrema in the DoGs (from Lowe, 2006).

If the pixel is either larger or smaller than the other 26 pixels, the pixel is saved for further refinement and testing. To determine if the pixel has proper characteristics, a 3D quadratic fitting function is used. This is done by shifting the origin of the DoG to the pixel location, and by using Taylor expansion the following expression is formed

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}, \tag{2.7}$$

here $\mathbf{x} = (x, y, \sigma)^T$ describes the offset from the location of the shifted pixel. The extremum is then located by taking the derivative of (2.7) w.r.t $\mathbf{x}$ and solving for zero, resulting in the following equation

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1}\frac{\partial D}{\partial \mathbf{x}}. \tag{2.8}$$

If $\hat{\mathbf{x}}$ is larger than 0.5 in any direction it indicates that the extrema of the point which is investigated, lies closer to an other point. The point is then changed, and the test is redone. When below 0.5 in any direction, the offset is added to the pixel, which gives "the interpolated estimate for the location of the extremum"[19]. For sorting out points with low contrast, (2.7) combined with (2.8) gives

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}, \tag{2.9}$$

which returns the function value at the extrema point. Then the keypoint is either kept or discarded by comparing the function value to a threshold. In order to further increase stability, keypoints located along edges are removed. These keypoints have a low principal curvature perpendicular to the edge, and large along the edge. To find these keypoints, the Hessian matrix is used to calculate the principal curvatures. It is calculated at the scale and location of the detected keypoint, as seen in (2.10) below

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}. \tag{2.10}$$

The ratio of the Hessian's eigenvalues are of importance, the larger eigenvalue in terms of magnitude is denoted α and the smaller is denoted β. Then the trace and

determinant can be calculated as

$$\begin{aligned} Tr(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta, \\ Det(\mathbf{H}) &= D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta. \end{aligned} \tag{2.11}$$

If the determinant becomes negative, the point is discarded due to curvatures with different signs. By introducing r, the ratio between the two eigenvalues, we have that $\alpha = r\beta$. Combining this with (2.11) the following inequality can be formed

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} < \frac{(r+1)^2}{r}. \tag{2.12}$$

This makes it easy to see if the principal curvature ratio is below the threshold r. The RHS of (2.12) is minimized when the eigenvalues of the Hessian is equal. When the ratio r is greater than 10 [19], the keypoint is discarded and the edge responses have been sorted out. In order to make the keypoints rotational invariant, the keypoints are assigned an orientation. This is done by computing magnitude and orientation in the closest smoothed image based on the scale, for every detected keypoint as follows

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \\ \theta(x,y) &= tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y))) \end{aligned} \tag{2.13}$$

Once this is done, a histogram of the gradient orientation is computed. This is done over 36 bins around the keypoint, which covers the 360° of possible orientations. Each point in the histogram is also weighted in terms of distance to the keypoint. The peak given by the histogram determines the orientation of the region.

To summarize, each keypoint given by the SIFT algorithm has the following properties:

1. Location, x- and y-coordinate of the keypoint centre
2. Scale, radius of the calculated circle (scale invariance)
3. Orientation, angle based on direction of the region's gradient magnitude (rotation invariance)

An important note about the SIFT detector is that the algorithm is patented, see [21] for a full commercial license.

2.3.4 SURF detector

Speeded Up Robust Features (SURF) is similar to SIFT (section 2.3.3) in the sense that the steps are the same, but they are done differently. Instead of smoothing the image with a Gaussian, as in SIFT detection, the SURF algorithm filters the image with windows based on the integral image method [22] as an approximation of a Gaussian

$$I_\Sigma(x,y) = \sum_{i}^{x}\sum_{j}^{y} I(i,j). \tag{2.14}$$

The benefit of using this method is that the number of calculations are low and that different window sizes have no impact on the calculation time [23]. One addition

and two subtractions using the corner points of the window are required to calculate the sum, see Figure 2.7.

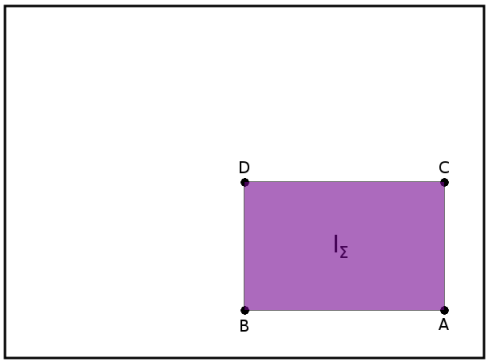

Figure 2.7: Illustration of the integral image method.

The sum of all pixels as seen in (2.14) is thus equal to $I_\Sigma = A - B - C + D$, given that the origin of the original image is in the upper left corner. Blob-detection is used and it is done using the Hessian matrix. Given an image I and a point $\mathbf{x} = (x, y)$ in that image, the Hessian matrix $H(\mathbf{x}, \sigma)$ is calculated. Here σ is the current scale. The matrix has the following form

$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}. \tag{2.15}$$

Where $L_{xx,xy,yy}(\mathbf{x}, \sigma)$ are the second order partial derivatives convoluted with I in point $\mathbf{x}$. These partial derivatives are approximated using filters for different scales and octaves. Instead of smoothing and down-sampling the image for each σ as in SIFT, the SURF algorithm up-scales the filter size. The use of integral images assures a constant computational cost. Once the Hessian is formed for a point $\mathbf{x}$ at a certain scale σ, the determinant of the Hessian is calculated and weighted in order to get a good approximation. The approximated determinant is saved in a response map for the current scale. Once the Hessian matrix are calculated for all octaves, the algorithm searches for local maximum and the detected maximum is saved and "interpolated in scale and image space"[23]. This procedure assures scale invariance.

Furthermore, Haar wavelets [24] are used to add rotational invariance. A circular region with radius $6s$, where s is the scale in which the point was detected, is centered at the feature point. Haar wavelet responses are calculated and weighted in both x and y direction. The orientation of the feature point is determined by summing the responses within a sliding orientation window of the size $\frac{\pi}{3}$. This gives an orientation vector, and the vector of highest magnitude is chosen to describe the orientation, see Figure 2.8.

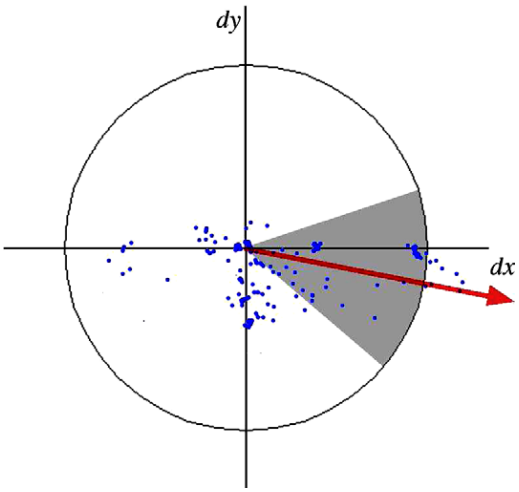

Figure 2.8: Orientation assignment using sum of Haar wavelet responses in a sliding orientation window (from Bay et al, 2008).

To conclude, the SURF detected keypoints has

1. Location, x- and y-coordinate
2. Scale
3. Orientation

2.4 Feature descriptors

Once a set of keypoints has been extracted from an image, they need to be processed for further use. This is done by encoding a numerical description of the image area around the keypoint, for each individual keypoint. By doing this, the keypoints between two images are comparable to each other.

2.4.1 ORB

ORB is an acronym for Oriented FAST and Rotation-Aware BRIEF. This means that the ORB descriptor needs the keypoints to be detected by the oFAST detector. The detection is done as described in section 2.3.2 with the addition of orientation to the feature points. This is done by calculating the intensity centroid [25]. First, the moment of a patch is calculated as [26]

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y), \tag{2.16}$$

where p and q are parameters for a pixels order of moment. And the centroid of the patch is found with the following equation

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}). \tag{2.17}$$

Then a vector $(\vec{OC})$ is constructed from the center of the detected feature, O, to the centroid given by (2.17). The angle of $\vec{OC}$ gives the orientation of the patch. It

is calculated by the following equation

$$\theta = \text{atan}^2(m_{01}, m_{10}). \tag{2.18}$$

Once the features has been assigned an orientation, the BRIEF descriptor does a binary test, τ, of intensity between points in a smoothed image patch. The smoothing is done using integral image as explained in section 2.5.2, "each test point is a 5 x 5 subwindow of a 31 x 31 pixel patch" [26]. The test is defined as

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{if } \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases}, \tag{2.19}$$

here $\mathbf{p}$ is the image patch which has been smoothed and $\mathbf{x}$ is the point that is tested. The resulting feature is a vector of 256 binary tests

$$f_{256}(\mathbf{p}) = \sum_{1 \leq i \leq 256} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y}_i). \tag{2.20}$$

In order to make the Binary Robust Independent Elementary Features descriptor invariant to rotation, is to steer the descriptors to the keypoint orientation θ. This is done by constructing a steered version of the binary tests at the feature location

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S}, \tag{2.21}$$

where $\mathbf{R}_\theta$ is a rotation matrix and $\mathbf{S}$ is equal to

$$\mathbf{S} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_{256} \\ \mathbf{y}_1 & \cdots & \mathbf{y}_{256} \end{bmatrix}. \tag{2.22}$$

The steered BRIEF then is equal to

$$g_n(\mathbf{p}, \theta) = f_n(\mathbf{p}) | (\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{S}_\theta. \tag{2.23}$$

A downside with the steered BRIEF is the low variance and high correlation among the binary tests. This is reduced by searching among the binary tests for tests with high variance and uncorrelation. Further details of the algorithm can be seen in [26]. The resulting descriptor is called rBRIEF and the method is considerable faster than SURF and SIFT [26].

2.4.2 SIFT descriptor

First the gradient and orientation are computed for points in a window which is centered by the keypoint of interest. The size of the window is constructed by 16×16 bins in an array. This array is divided into 4×4 sub-regions, where a histogram based on orientation is computed for each sub-region. Then the coordinates of these points and the orientation of the gradients are rotated w.r.t the keypoint orientation, to incorporate rotational invariance. The points are further refined by using a Gaussian weighting function where $\sigma = \frac{w}{2}$, w is the width of the descriptor window. The weighting is aimed at the magnitude of each point. One important

property by doing the weighting is that the gradients which are far away from the keypoint are taken less into account, "as these are more affected by misregistration errors" [19]. The procedure is illustrated in Figure 2.9

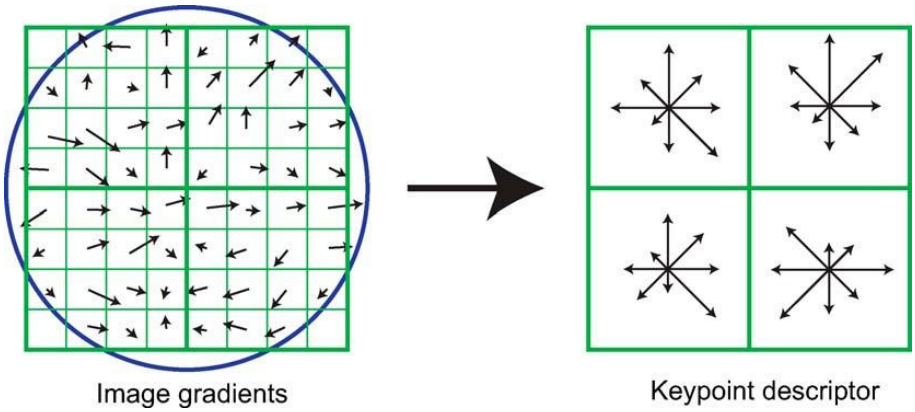

Figure 2.9: SIFT feature description procedure. The arrows depict the sum of gradients in the individual bins. The blue ring in the left image shows the region which is considered after the Gaussian weighting function is applied. This example shows a 8×8 array and the resulting 2×2 descriptor (from Lowe, 2004).

2.4.3 SURF descriptors

A square window is centered at the keypoint, with the same orientation as the keypoint. The size of the window is $20s \times 20s$, where s is the scale in which the keypoint was detected. The window is then evenly split up into 4×4 sub-regions. Haar wavelet responses and their absolute value are summed up for horizontal and vertical direction in each sub-region. The resulting descriptor consists of a 4D vector which contains the four sums for each sub-region.

2.5 Feature matching

When the features and descriptors are extracted from a sequence of images, the next step is to match the features from one image to the next image in the sequence. This is done by the matcher and two different methods are briefly explained below. To further refine the matches, if the second closest match has a distance ratio greater than 0.8 of the best match then the match is rejected. "Which eliminates 90% of the false matches while discarding less than 5% of the correct matches"[19].

2.5.1 Brute force

The brute force matcher compares a descriptor from a set of keypoints in the first image to all the descriptors of the keypoints in the second image. Generally, the descriptor with the shortest Euclidian distance is then matched to the descriptor in the first image. The distance of the shortest descriptor also needs to be below some defined threshold in order to make the matching valid. The time between two images must also be low enough in order to correctly match the descriptors.

2.5.2 FLANN

Fast Library for Approximate Nearest Neighbours (FLANN), consists of algorithms that searches a dataset for the closest neighbour. It is functional in high dimensional spaces but also well suited for feature matching in two dimensions. OpenCV uses a FLANN based matcher which utilizes the k-d tree data structure and nearest neighbour search. K-d stands for k-dimensional, where k is a positive integer. The basic idea behind k-d trees is, given a dataset with k dimensions:

1. Pick a dimension from k
2. Find the median value of that dimension
3. Split that dimension in two halves based on the median value
4. Repeat, k-1 until it reaches 0 then start over with the original k until all elements in the dataset is examined

The procedure is illustrated with a dataset in Fig 2.10. In this example the starting point is (7,2).

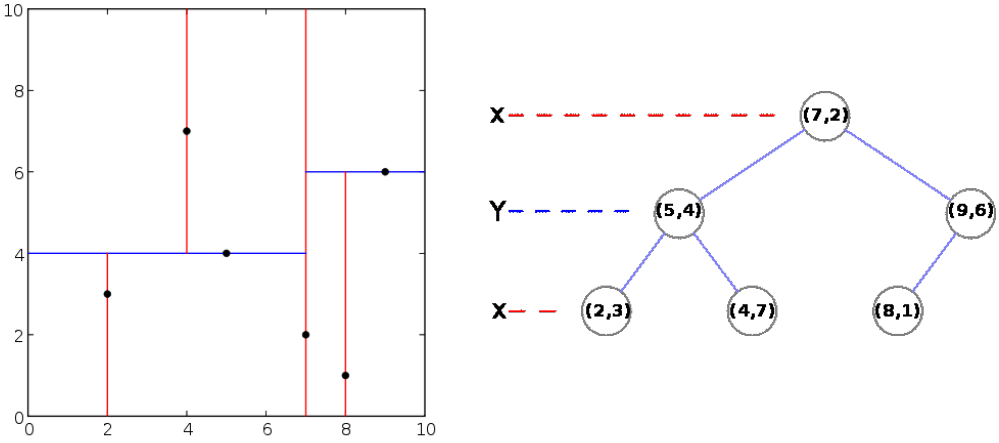

Figure 2.10: The left image shows a 2-d decomposition, note that the red lines are for x and blue for y. Right image shows corresponding 2-d tree.

Once the k-d tree is formed, the next step is to use points from the second dataset and see which node from the k-d tree that the examined point is closest to. The procedure is called Nearest Neighbour (NN). By starting at the root node (7,2) in the example, the algorithm moves down the constructed 2-d tree recursively. It chooses the left or right branch if the examined point has a greater or smaller value than the current node in current dimension. Once the algorithm has reached an ending node (leaf node), the algorithm saves that node as the current best. The recursion is then finished and the algorithm start to traverse back to the root node. The distance to the examined point is checked at every node, if the distance is smaller then the current best gets updated. The algorithm also checks if the hyperplane on the other side of the tree is inside the radius of the current shortest distance, if not, that hyperplane is discarded. If it is inside, then the algorithm runs through that branch of the k-d tree as well. The procedure can be seen in the example shown in Figure 2.11, and an explanation can be read after the image.

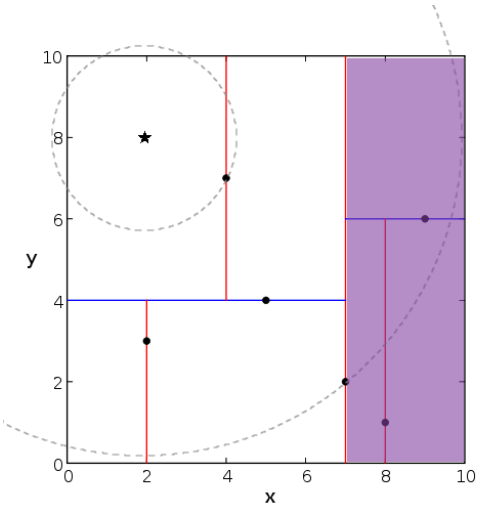

Figure 2.11: NN search algorithm based on a 2-d tree.

Point (2,8) is examined for its nearest neighbour, marked with a star in the figure. The algorithm starts at the root node (7,2) and the distance e.g. the radius of the large circle centered at the star, covers all hyperplanes. This means that we cannot discard any hyperplane yet. The algorithm runs through the tree, ignoring the purple hyperplanes in the figure. The leaf node (4,7) is set as the current best match and the algorithm starts to traverse back to the root node, checking the distance at every node. Since the radius of the small circle centered around the star is the current best match and it is not intersecting with the purple hyperplanes, we can now discard those hyperplanes and the algorithm does not search through them. Once at the root node, the algorithm terminates and (4,7) was indeed the closest neighbour.

2.6 KLT tracker

The Kanade-Lucas-Tomasi (KLT) tracker is an intensity based tracker, this method tracks points in one image to another using optical flow which is based on changes in light intensity. This means that the tracker is not comparing nor matching any descriptors as described in section 2.5. Given I and J, two images in sequence, the idea is to find a point $\mathbf{v} = \begin{bmatrix} v_x & v_y \end{bmatrix}^T$ on image J where the intensity of $I(\mathbf{u})$ and $J(\mathbf{v})$ are a match. This can be summarized in the following equation

$$\mathbf{v} = \mathbf{u} + \mathbf{d} = \begin{bmatrix} u_x + d_x & u_y + d_y \end{bmatrix}^T \tag{2.24}$$

where $\mathbf{d} = \begin{bmatrix} d_x & d_y \end{bmatrix}^T$ is the optical flow at point $\mathbf{u}$. Since there are multiple disadvantages with tracking only a pixel using optical flow[27], the algorithm uses a window which is centered at the feature point. The vector $\mathbf{d}$, also called the displacement vector, contains as mentioned two parameters. These parameters are chosen in order to minimize the residual error. Which can be seen in the following

equation [28]

$$\epsilon(\mathbf{d}) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x+d_x, y+d_y))^2 \tag{2.25}$$

Here w_x and w_y are parameters for the window size. If the residual error is below some threshold, the points are a match. The time between two sequential images must be sufficiently low, since images taken at near same time instances are strongly related and the displacement vector will thus be small[27]. Further details about the KLT tracker can be read about here [27, 28, 29, 30]

2.7 Motion estimation

Once features have been found and matched in previous and current image, it's time to estimate the motion. Motion estimation is the fundamental step performed in every step in a VO system. The basis of this algorithm involves computation of the relative camera motion between previous and current image. By concatenating all these relative movements, the trajectory of the camera is recovered.

This section will give a detailed explanation of two methods used for estimating the relative motion. The two methods are

- 2D–*to*–2D: Features in both previous and current image are given in 2D image coordinates.
- 3D–*to*–2D: In this case the features in previous image are triangulated and given in 3D-coordinates. Corresponding features in current image are given in 2D.

2.7.1 2D–*to*–2D

The underlying idea of 2D–*to*–2D is to utilize some geometric relations to which the image correspondences are restricted to. These relations are also known as *epipolar geometry* and can be seen in Figure 2.1 on p. 5.

Consider the case of monocular VO where two images I_0 and I_1 are taken at time $k = \{0, 1\}$. Let $\mathbf{c}_0$ and $\mathbf{c}_1$ denote the camera centers at $k = 0$ and $k = 1$ respectively, where $\mathbf{c}_1$ has undergone the rotation $\mathbf{R}$ and translation $\mathbf{t}$ relative $\mathbf{c}_0$. Also let the projection of the 3D-point $\mathbf{X}$ on the camera planes, i.e. images, be $\mathbf{p}$ and $\mathbf{p}'$. Looking at Figure 2.1 on p. 5 we can see that the vectors, between $\mathbf{X}$ and the camera centers, $(\mathbf{c}_0 - \mathbf{X})$ and $(\mathbf{c}_1 - \mathbf{X})$ spans a plane known as the *epipolar plane*. Since $\mathbf{t}$ also lie in this plane, we know that the cross product between $\mathbf{t}$ and $(\mathbf{c}_1 - \mathbf{X})$ is the normal vector of the epipolar plane and it is orthogonal to $(\mathbf{c}_0 - \mathbf{X})$. This means that

$$(\mathbf{c}_0 - \mathbf{X})^\top (\mathbf{t} \times (\mathbf{c}_1 - \mathbf{X})) = 0. \tag{2.26}$$

However since we know that $\mathbf{p}$ is parallel to $(\mathbf{c}_0 - \mathbf{X})$ and $\mathbf{R}\mathbf{p}'$ is parallel to $(\mathbf{c}_1 - \mathbf{X})$, (2.26) can be rewritten as

$$\mathbf{p}^\top (\mathbf{t} \times \mathbf{R}\mathbf{p}') = 0. \tag{2.27}$$

By using the property $\mathbf{t} \times \mathbf{R} = [\mathbf{t}]_\times \mathbf{R}$, where

$$[\mathbf{t}]_\times = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix}, \tag{2.28}$$

is a skew symmetric matrix, (2.27) becomes,

$$\mathbf{p}^\top [\mathbf{t}]_\times \mathbf{R}\mathbf{p}' = \mathbf{p}^\top \mathbf{E}\mathbf{p}' = 0. \tag{2.29}$$

The matrix $\mathbf{E}$ is called the *essential matrix*. Hence the 2D–to–2D correspondences problem boils down to estimating the essential matrix satisfying (2.29) for all of the feature correspondences. In practice there is no matrix that is able to solve it due to noise, there will always some residual left. We can deal with this problem by incorporating it with a RANSAC-framework in order to minimize the residual, see section 2.9.

There are several different methods solving for the essential matrix, however since $\mathbf{R}$ has three Degrees of Freedom (DoF) and $\mathbf{t}$ has two DoF (correct up to scale), we have chosen the *five point solution* proposed by Nistér in [31]. This means that only five correspondence points are needed to estimate the essential matrix. Generally we want to choose a method based on as few correspondences as possible for efficiency, see section 2.9. Once the essential matrix is computed we need to extract $\mathbf{R}$ and $\mathbf{t}$ where

$$\begin{aligned} \mathbf{R} &= \mathbf{U}\left(\pm\mathbf{W}^\top\right)\mathbf{V}^\top, \\ \mathbf{t} &= \mathbf{U}\left(\pm\mathbf{W}\right)\mathbf{S}\mathbf{U}^\top, \end{aligned} \tag{2.30}$$

where

$$\mathbf{W}^\top = \begin{pmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.31}$$

and the matrices $\mathbf{U}, \mathbf{S}$ and $\mathbf{V}$ are computed from the Singular Value Decomposition (SVD),

$$\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^\top. \tag{2.32}$$

2.7.2 3D–*to*–2D

As mentioned in previous section, the five point method can be used to calculate the essential matrix from which the pose ($\mathbf{R}$ and $\mathbf{t}$) can be extracted. The method requires image correspondences in two different images (feature points) as indicated by the name, 2D–*to*–2D. However there are methods utilizing feature points and their corresponding 3D-points. The problem of estimating the pose using this approach is know as *Perspective–n–Point* (PnP), where n is the number of feature points and their corresponding 3D-points. Although this method is not used in this thesis, it is interesting to know how it works compared to 2D–*to*–2D.

In section 2.9 we learn how the number of correspondence points affects the number of iterations required by the RANSAC algorithm in order to filter out outliers. This means that we can keep the number of iterations down by estimating the pose using a method requiring fewer correspondence points. And as show in [32], the minimum number of correspondence points required is three and the method is denoted as P3P. The output of the algorithm will in fact be four possible poses, however by using a fourth correspondence point three of the solutions will be eliminated.

One downside with this method is that the 3D-coordinates of the feature points are required as indicated by the name 3D–*to*–2D. This is not a problem if the images are given in Red Green Blue - Depth (RGB-D) format (the format uses one extra channel containing information about the image depth). However in our case, where we only use one camera without any information about depth, some workaround is required concerning triangulation of the feature points as described in section 2.8.

The 3D–*to*–2D method requires image correspondences across three images. The outline of this method is as follows:

1. Compute feature correspondences $\mathbf{p}_{k-2}, \mathbf{p}_{k-1}$ and $\mathbf{p}_k$ across the images I_{k-2}, I_{k-1} and I_k.
2. Use 2D–*to*–2D method to extract relative transformation $\mathbf{T}_{k-2,k-1}$ between images I_{k-2} and I_{k-1}.
3. Use $\mathbf{T}_{k-2,k-1}$ to triangulate corresponding 3D-points $\mathbf{X}$.
4. Use $\mathbf{X}$ and $\mathbf{p}_k$ in the P3P-algorithm to compute the transformation $\mathbf{T}_{k-1,k}$ between images I_{k-1} and I_k.
5. Compute feature correspondences $\mathbf{p}_{k-1}, \mathbf{p}_k$ and $\mathbf{p}_{k+1}$ across I_{k-1}, I_k and a new image I_{k+1}.
6. $k = k + 1$.
7. Go to step 3.

It should be noted that $\mathbf{X}$ and $\mathbf{p}$ are vectors of equal length containing 3D-coordinates and 2D-coordinates respectively for the features extracted and matched across the images. The length $n \geq 4$ is required since the P3P-algorithm needs at least three features and the fourth feature is used to get the unique solution. We can also see that the 2D–*to*–2D method is only used to initialize the algorithm.

2.8 Triangulation

As discussed in section 2.7.2 the P3P-algorithm requires 3D correspondence which in this case (working with only one camera with no depth information) must be computed using triangulation from two different views. The process of triangulation concerns the task of computing the position of a point in space given its position in two images by finding the intersection of two known rays in space. Acknowledging the fact that feature matching across different views is not always perfect due to noise, makes the task of triangulation slightly more difficult than expected. This complication arises since the two rays will generally not meet, hence it is necessary

to find the best point of intersection. A lot of research has been done in this area and as a result of that several different methods have been developed. For more information refer to [33, 34] where the authors have developed an optimal way of triangulation, they also compare several different methods.

The simplest methods is the *linear triangulation method* which is the only method implemented in the OpenCV library. The method works by using the property of the pinhole camera model described in (2.2) with some additional parts since we are dealing with two cameras. We can see that (2.2) is in fact equal to

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{pmatrix}}_{\mathbf{M}=(\mathbf{R}|\mathbf{t})} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.33}$$

when the matrix $\mathbf{M}$ (extrinsic matrix) has an identity matrix as rotation and no translation, $(\mathbf{I}_3|\mathbf{0})$. The extrinsic matrix $\mathbf{M}'$ for the second camera has however undergone a transformation relative the first camera. The general form of the projection is expressed as

$$\lambda \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{M} \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix} \tag{2.34}$$

for the first camera and

$$\lambda \begin{pmatrix} \mathbf{p}' \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{M}' \begin{pmatrix} \mathbf{X} \\ 1 \end{pmatrix} \tag{2.35}$$

for the second camera. Hence the objective of the linear triangulation method is to find a vector $\mathbf{X}$ satisfying (2.34) and (2.35). We can combine these system of equations and form $\mathbf{A}\mathbf{X} = \mathbf{0}$ [33], where

$$\mathbf{A} = \begin{pmatrix} u\mathbf{f}^{3\top} - \mathbf{f}^{1\top} \\ v\mathbf{f}^{3\top} - \mathbf{f}^{2\top} \\ u'\mathbf{f}'^{3\top} - \mathbf{f}'^{1\top} \\ v'\mathbf{f}'^{3\top} - \mathbf{f}'^{2\top} \end{pmatrix}, \tag{2.36}$$

and $\mathbf{f}^{i\top}$ are the rows of $\mathbf{P} = \mathbf{K}\mathbf{M}$. The equations given by $\mathbf{A}\mathbf{X} = \mathbf{0}$ can then be solved using singular value decomposition.

2.9 Outlier removal using RANSAC

As it is evident from sections 2.7.1 and 2.7.2, the foundation on which VO rests is based upon the idea that the feature correspondences used in the process are perfect. However since this is not the case we must contemplate with how to deal with these outliers (false correspondences). Not accounting for this will yield wrong essential matrix which in the end results in false transformation matrix.

One way of solving this problem is by using a method know as Random Sample Consensus (RANSAC). The idea is to randomly select a set of data points which are

then used to calculate a model. Then try to fit the rest of the data points to this model and compute the error residuals. The set of points fitted to the model with a residual less than a predefined threshold is called the *consensus set*. Redo this iteratively a number of times and select the set yielding the largest consensus set. In our case we want to choose a set of *five* feature correspondences which are used to compute the essential matrix. Use all other feature correspondences to compute the residual given by (2.29). The largest consensus set are inliers and rest are outliers. A pseudocode for this method is given in Algorithm 1.

input : All features correspondences (w), iterations (K), threshold (t)
output: Five features (x) with largest consensus set (N), best essential matrix (E)

```
N ← 0;
for K iterations do
    l ← selectFiveFeatures(w);
    tempE ← computeEssentialMatrix(l);
    n ← fulfillConstraint(w - l, tempE, t);
    if N < n then
        x ← l;
        E ← tempE;
        N ← n;
    end
end
```

Algorithm 1: RANSAC.

The algorithm outputs five feature correspondences along with the essential matrix they produce. The function `fulfillConstraint()` is simply computing the error residual according to (2.29) and returns the number of features for which the residual is less than the given threshold, i.e. number of inliers. It is important to note that a sufficient number of iterations is required in order to ensure that the five feature correspondences are all inliers. If K is too small then some of the features might be outliers hence a false essential matrix. However if K is too large then the RANSAC algorithm will be time consuming.

Let N denote the total number of feature correspondences and s is the size of sample we randomly select (five in our case). Furthermore assume that actual fraction of inliers is ϵ and let γ be the probability that the RANSAC algorithm at least once selects s inliers. This means that probability of choosing a set with only inliers is

$$\mathrm{P}(\text{set of inliers}) = \epsilon^s, \tag{2.37}$$

which means that the probability of choosing a set with outliers is

$$1 - \mathrm{P}(\text{set of inliers}) = 1 - \epsilon^s, \tag{2.38}$$

hence selecting a set of outliers K times is

$$\mathrm{P}(K \text{ sets of outliers}) = (1 - \epsilon^s)^K. \tag{2.39}$$

This is equivalence to the RANSAC algorithm failing which means the probability of success at least once (γ) is

$$\mathrm{P}(\text{success}) = 1 - (1 - \epsilon^s)^K = \gamma. \tag{2.40}$$

By rewriting previous equation we conclude that the number of iterations K is

$$K = \frac{\log(1-\gamma)}{\log(1-\epsilon^s)}, \tag{2.41}$$

and in Table 2.1 we can see how the number of iterations are affected by the sample size and the percentage of inliers.

Number of iterations required for success

	Sample size (s)						
% inliers (ϵ)	2	3	4	5	6	7	8
90	3	4	5	6	7	8	9
80	5	7	9	12	16	20	26
70	7	11	17	26	37	54	78
60	11	19	34	57	97	163	272
50	17	35	72	146	293	588	1177
40	27	70	178	448	1123	2809	7025
30	49	169	567	1893	6315	21055	70188

Table 2.1: The table shows how the sample size (s) and the fraction of inliers (ϵ) affects the number of iterations (K) required to successfully pick a sample with only inliers when the desired probability of success (γ) is set to 99%.

3
Implementation

This chapter describes the how the Visual Odometry algorithm is implemented by piecing together the different parts from the previous chapter.

The implementation is done using C++ due to its superiority in speed, together with the third party library OpenCV.

As mentioned in previous chapter the motion estimation in monocular VO can be done using one of two methods, 2D–*to*–2D or 3D–*to*–2D. The algorithm in this project is using the former, hence no triangulation is needed.

3.1 Visual Odometry

The implemented algorithm is divided into *four* parts. **Part 1** acquires an image and extracts features using one of three methods described in section 2.3. **Part 2** involves finding corresponding features in another image which can be done by either using *feature matching* (see section 2.5) or *feature tracking* (see section 2.6). In **Part 3** the algorithm utilizes the *epipolar contraint* in order to compute the *essential matrix* using the *five points solution* within a *RANSAC* framework. This step also removes outliers as described in section 2.9. **Part 4**, which is the last part of the algorithm, computes and extracts the rotation matrix along with the translation vector according to equation (2.30).

Part 1, Feature detection: The first step is to acquire a rectified[1] image and send it to one of the feature detectors. The detector will find a set of features, the quality and quantity of the features depend on the detector parameters, see Appendix A.1 for a list of parameters used. Depending on the choice of method, matching or tracking, descriptors for the features must be extracted.

Part 2, Finding corresponding features: The feature matching algorithm requires features and descriptors for two consecutive images. Once those are sent to the FLANN matcher the algorithm finds corresponding features in the two images by comparing the descriptors. The features for which no match is found is deleted along with the corresponding descriptor. A list of parameters for the FLANN matcher can be seen in Appendix A.1.

[1]The images from the KITTI dataset used in this project are all grayscale and rectified.

The KLT tracker works in a different way, it does not need descriptors. The tracker, given a set of features in one image, tracks the features based on optical flow and finds the corresponding features in the consecutive image. The features for which no "match" is found, or the ones moving out of the image, are removed. Unlike the feature matcher, where new features are detected in every new image, the features detected in an image are tracked over several images. Once the number of features drop below a certain threshold new features are detected.

Part 3, Compute essential matrix: The feature correspondences acquired in previous part are sent to the *five point* algorithm in order to compute the essential matrix. Within this algorithm is the RANSAC-framework incorporated (see Algorithm 1) where an essential matrix best describing the feature correspondences is computed. The algorithm also ensures to remove any feature correspondences which do not satisfy equation 2.29 using the computed essential matrix. The set of features left are inliers. By tuning the RANSAC parameters described in section 2.9 the strictness of outlier rejection is achieved.

Part 4, Compute rotation and translation: The final part of the VO is to extract the rotation matrix and translation vector from the essential matrix, computed previously, using equation (2.30). The rotation and translation are used according to equation (2.1) in order to compute the relative transformation between two images. Once this is done the algorithm returns to **Part 1**.

A summary of the algorithm using feature matching and feature tracking is given in Algorithm 2 and 3.

input : Image (I), Velocity (v), minimum matches (ϵ)
output: Rotation (R), Translation (t)

for *image* 1 **do**
    $p' \leftarrow$ `extractFeatures(`I`);`
    $d' \leftarrow$ `computeDescription(`I`, `p`');`
end

for *images* $2 : N$ **do**
    $p \leftarrow$ `extractFeatures(`I`);`
    $d \leftarrow$ `computeDescription(`I`, `p`);`
    $m \leftarrow$ `matchFeatures(`d`', `d`);`

    if `size(`m`)` $< \epsilon$ **then**
        `/* Not enough matches */`
        $[p', d'] \leftarrow [p, d]$;
        return;
    end

    `/* Compute essential matrix using only inliers */`
    `/* oulier rejection using RANSAC is embedded */`
    $E \leftarrow$ `computeEssentialMatrix(`p`', `p`, `m`);`
    $[R, t] \leftarrow$ `computeRotationTranslation(`E`);`
    $t \leftarrow$ `correctScale(`t`, `v`);`

    `/* Save features and descriptors */`
    $[p', d'] \leftarrow [p, d]$;
end

Algorithm 2: Visual Odometry using feature matching.

input : Image (I), Velocity (v), minimum features tracked (ϵ)
output: Rotation (R), Translation (t)

for *image* 1 **do**
    $p' \leftarrow$ `extractFeatures(`I`)`;
    $I' \leftarrow I$;
end

for *images* $2 : N$ **do**
    $p \leftarrow$ `trackFeatures(`I`', `p`', `I`)`;
    if `size(`p`)` $< \epsilon$ **then**
        `/* Not enough features found, detect new features */`
        $p' \leftarrow$ `extractFeatures(`I`)`;
        $I' \leftarrow I$;
        return;
    end
    `/* Compute essential matrix using only inliers */`
    `/* oulier rejection using RANSAC is embedded */`
    $E \leftarrow$ `computeEssentialMatrix(`p`', `p`)`;
    $[R, t] \leftarrow$ `computeRotationTranslation(`E`)`;
    $t \leftarrow$ `correctScale(`t`, `v`)`;
    `/* Save features and descriptors */`
    $[p', d'] \leftarrow [p, d]$;
end

Algorithm 3: Visual Odometry using feature tracking.

4
Results

In order to evaluate and compare results from the different feature detectors, 3 different sets of images are used. The images which are from the KITTI dataset [35], have different characteristics such as environment, traffic, distance and speed, see Table 4.1. The results are divided into two parts. In the first part the algorithm is evaluated using feature matching with three different type of detectors and descriptors. In the second part however, the algorithm is using feature tracking instead with the same three detectors.

Data sets used in the simulations

Data set	Environment	Traffic	Speed	Frames	Distance	Name
00	Residential	+	+	4541	3.73 km	2011_10_03_drive_0027
01	Highway	+++	+++	1101	2.45 km	2011_10_03_drive_0042
02	Residential	+	+	4661	5.07 km	2011_09_29_drive_0071

Table 4.1: Overall information about the datasets used in the simulations.

The comparisons are made with respect to execution time per frame, rotation error measured in degrees per meter, and translation error measured in percent. The error in rotation and translation is calculated and compared with the GPS ground truth data (accurate <10 cm) using the KITTI evaluation code [35] where the results are divided into subsequences of length $(100, 200, \ldots, 800)$ meters. Note however that the execution time should be interpreted carefully since they may not be consistent for the different simulations.

The evaluation code computes the translation error as the distance between the endpoints of the ground truth subsequence and the corresponding estimated subsequence. The rotation error is the difference between the orientations of the final pose in the ground truth and VO algorithm for each subsequences.

4.1 Feature matching

This section will show the results obtained using feature matching. The results are computed as following; for each image sequence the algorithm is evaluated three times using the detectors along with their descriptors and matchers, i.e. FAST–ORB, SURF–SURF, and SIFT–SIFT. The results are then presented using three different

graphs showing the path of the vehicle, the translation error, and the rotation error for each method. In Table 4.2, at the end of the section, a summary of the obtained results are presented.

The result for dataset 00 are presented in Figure 4.1. We can see that errors for the three different detectors are quite similar, especially for SURF and SIFT. The FAST detector appears to handle the translation error better than the other however the rotation error is larger than the other detectors. Furthermore we note that the average execution time is 0.22274 s/frame for the FAST detector compared to 0.7667 s/frame and 0.6098 s/frame for SURF and SIFT respectively, see Table 4.2.

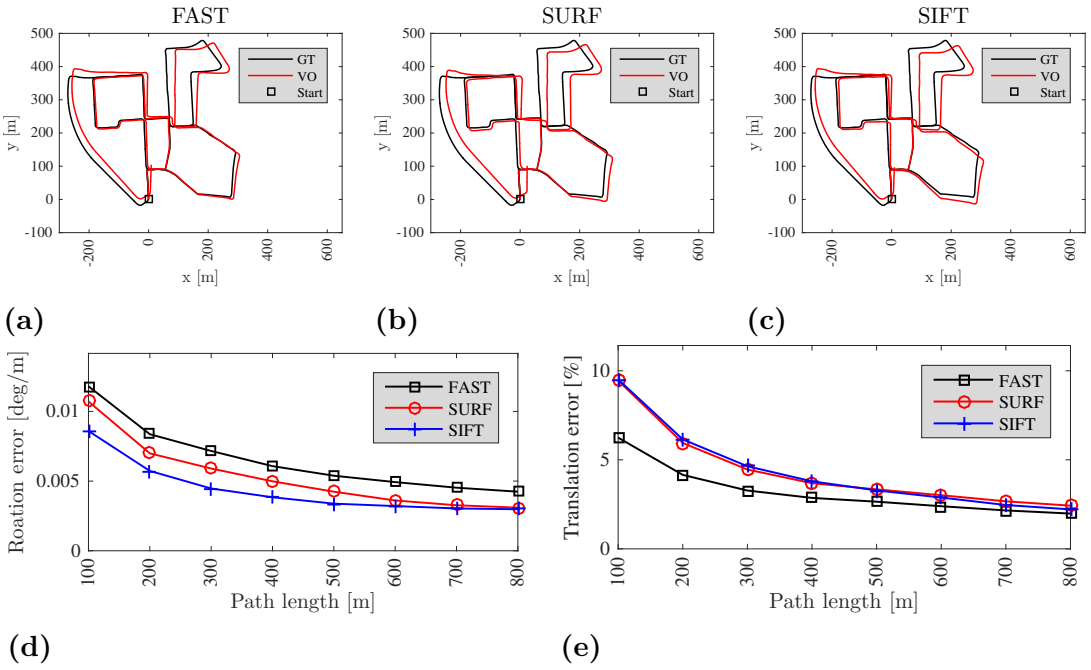

Figure 4.1: Evaluation of dataset 00 using feature matching.

Evaluation results for dataset 01 can be seen in Figure 4.2. This particular dataset represents driving on a highway where the velocity of the vehicle is high. The environment is also not ideal since it contains a lot of trees which makes the detection of "good" features poor. However the detectors performs relatively well considering the environment, especially SIFT with lowest error in both rotation and translation. And once again it is noted that the FAST detector exceeds in speed with an execution time of 0.1053 s/frame.

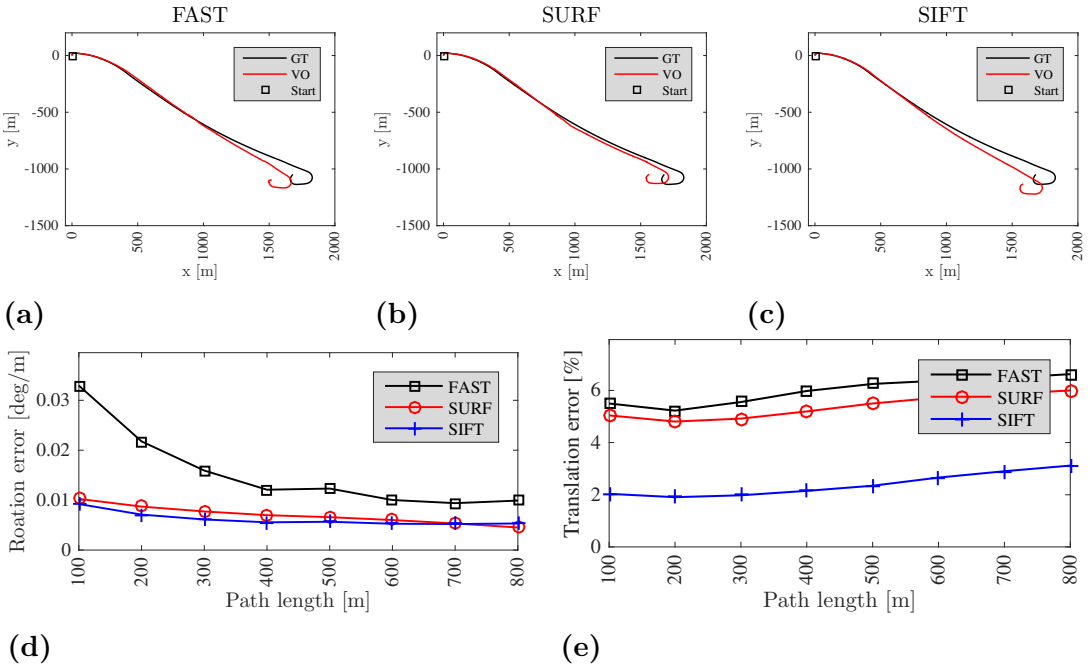

Figure 4.2: Evaluation of dataset 01 using feature matching.

Looking at the trajectory of the vehicle in Figure 4.3a – 4.3c we can immediately see that all three detectors performed quite well for dataset 02, which is also reflected in the error plots. Despite the large distance traveled by the vehicle, the error propagation is small in contrast to previous results. As predicted by the theory, SIFT achieves best result in terms of error while FAST outperforms in speed, see Table 4.2.

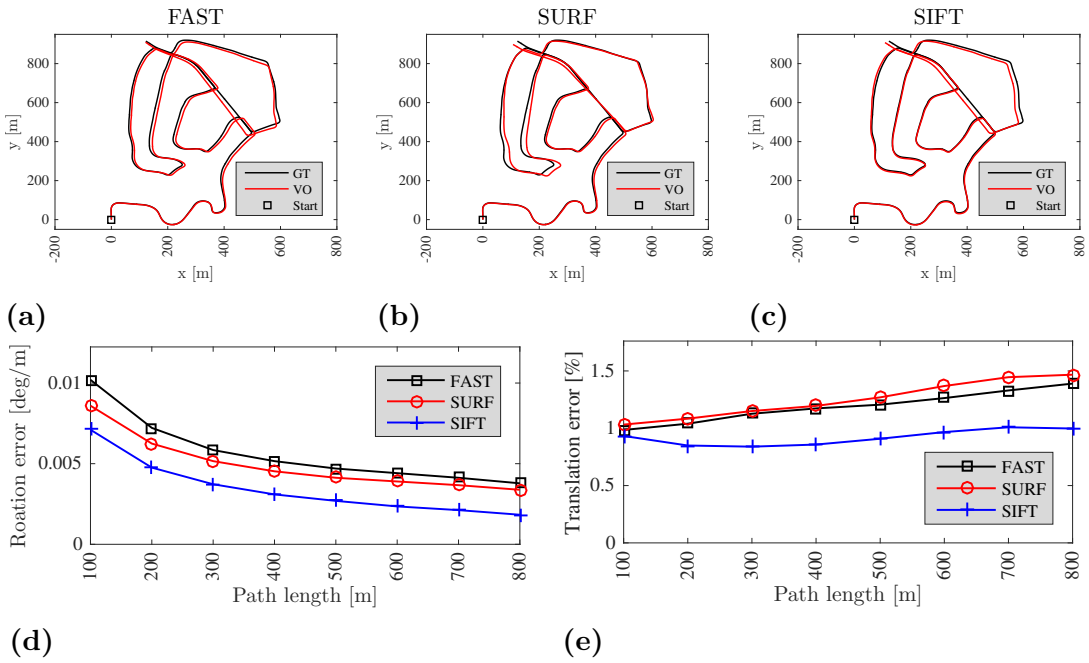

Figure 4.3: Evaluation of dataset 02 using feature matching.

Summary of results for matcher

	Data set	Rotation [deg/m]	Translation [%]	Time [s/frame]
FAST	00	0.0066	3.22	0.2274
	01	0.0155	5.99	0.1053
	02	0.0057	1.19	0.3567
mean		0.0093	3.47	0.2298
SURF	00	0.0057	4.37	0.7667
	01	0.0070	5.38	0.5421
	02	0.0049	1.25	0.8393
mean		0.0058	3.67	0.7161
SIFT	00	0.0044	4.37	0.6098
	01	0.0062	2.39	0.4876
	02	0.0035	0.92	0.7207
mean		0.0047	2.56	0.6061

Table 4.2: A summary of the results obtained by the detectors for all datasets using feature matching.

4.2 Feature tracking

Having evaluated all three datasets using feature matching in previous section, we proceed by reevaluating the datasets now using feature tracking. As described in section 2.6, the KLT does not depend on feature matching using descriptors instead it relies on optical flow. This means the VO algorithm will run faster but not necessarily better.

Starting with dataset 00 we can immaculately conclude that the results are slightly worse (as seen in Figures 4.4a – 4.4c) than the ones obtained by the matcher, especially for the FAST detector. However the execution time for all three methods have also changed as expected. All three methods are faster, noticeably for SURF and SIFT, when no feature matching is done.

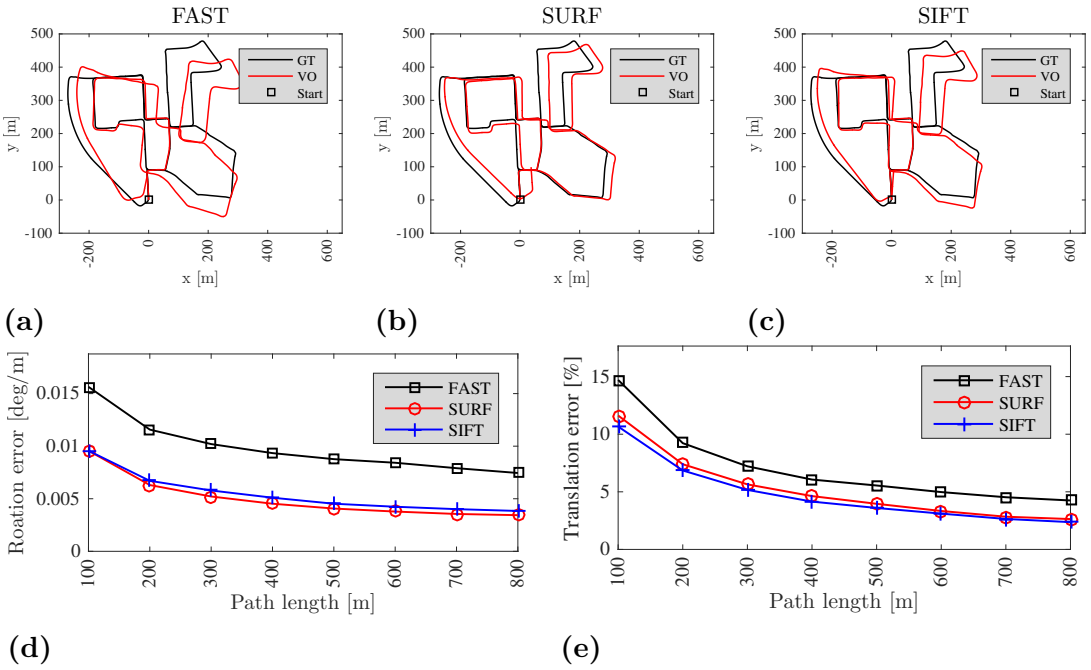

Figure 4.4: Evaluation of dataset 00 using feature tracking.

Continuing with dataset 01 we see that the impact on the results are even more noticeable than previous. As mentioned earlier, this dataset represent driving on a highway were no clear structures are visible which makes the feature detection more difficult. The only source for features are bypassing vehicles and trees. This combination is clearly reflected in the translation error Figure 4.5e where the error is increasing for all detectors. This difference is however expected considering the fact that we cannot distinguish the feature as good when no descriptors are available.

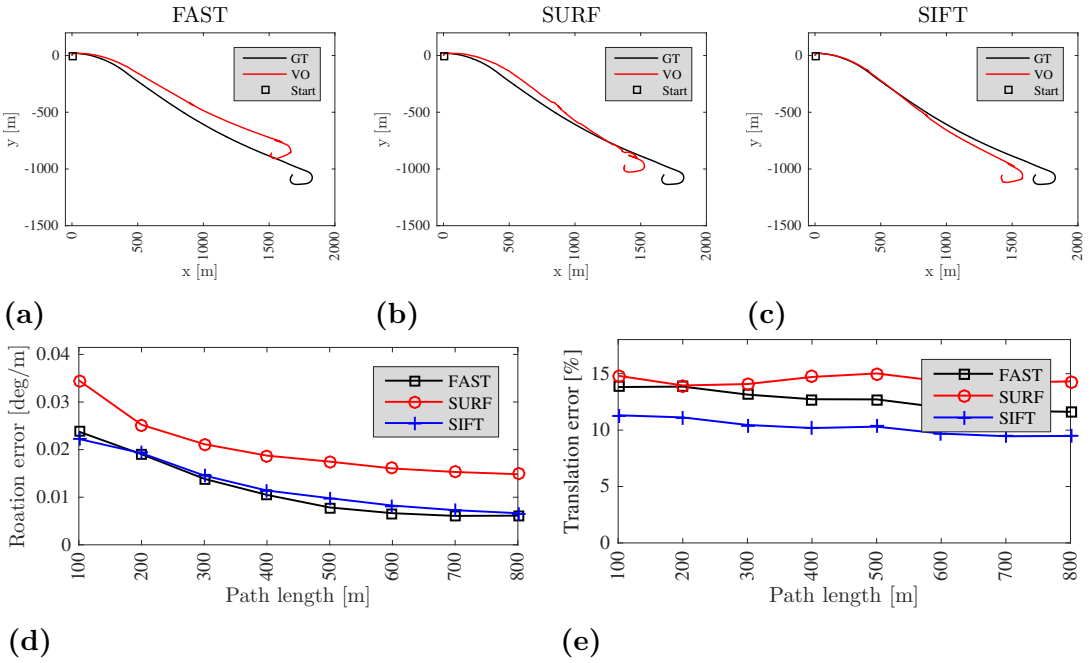

Figure 4.5: Evaluation of dataset 01 using feature tracking.

The last data evaluation, dataset 02, using the tracker has some interesting results. So far the trajectory of the path has been consistent for all three detectors however is not the case for this dataset. The method using FAST clearly deviates from the other two, Figure 4.6a. SURF and SIFT seems to handle the dataset quite well which might be due to the fact that the environment is full off structures providing "good enough" features to track for these methods. However looking at the images in the dataset we can also see that the vehicle is driving in an area with many trees which may be one reason to why the FAST detector performs so poorly. The error plots in Figure 4.6d and 4.6e show that SURF and SIFT perform very similar, which can also be seen in dataset 00 in Figure 4.4d – 4.4e. This raises the question to whether these two detector are so different or not, now that their respective descriptors are not used.

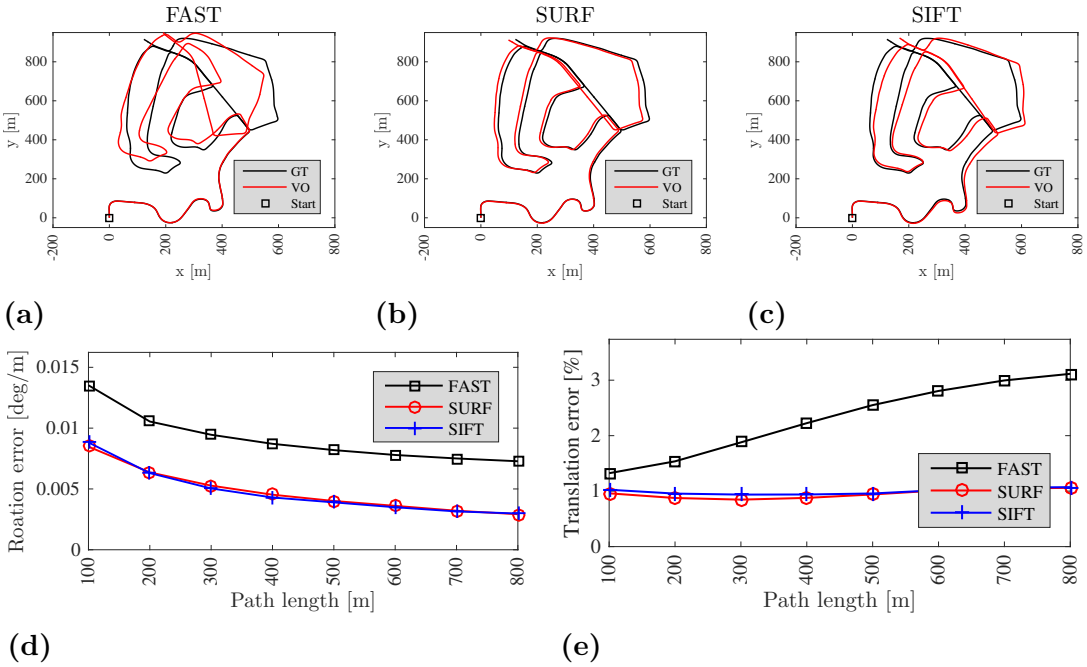

Figure 4.6: Evaluation of dataset 02 using feature tracking.

Summary of results for tracker

	Data set	Rotation [deg/m]	Translation [%]	Time [s/frame]
FAST	00	0.0099	7.06	0.1482
	01	0.0117	12.70	0.1441
	02	0.0091	2.30	0.2372
mean		0.0103	7.36	0.1765
SURF	00	0.0051	5.26	0.2332
	01	0.0204	14.43	0.2806
	02	0.0048	0.95	0.2459
mean		0.0101	6.88	0.2532
SIFT	00	0.0055	4.82	0.2086
	01	0.0124	10.25	0.2732
	02	0.0047	0.99	0.2438
mean		0.0075	5.36	0.2419

Table 4.3: A summary of the results obtained by the detectors for all datasets using feature tracking.

5
Discussion

This chapter is dedicated to a short discussion of the results obtained by the algorithm by trying to explain the behaviour. The chapter is divided into three parts. The first part will cover the affect of the chosen detectors and descriptors on the results. The second part will discuss the benefit of using 3D–*to*–2D instead of 2D–*to*–2D. And finally in the last part we will recommend some changes to be considered in future work to improve the ego-motion.

5.1 Feature detectors and descriptors

Looking at the results for the matcher we can see a clear and unmistakable difference between the different feature detectors, both in terms of execution time and accuracy. The underlying trend is that the FAST detector combined with the ORB descriptor yields the fastest execution time. However this comes at the cost of losing accuracy. We can also see that the SIFT detector is the most accurate detector but not as fast as the FAST detector. This trade off between speed and accuracy is to be expected and it is also supported by the theory. The same holds for the on going trend. One behaviour deviating from the theory is that of the SURF descriptor. Given how it works in comparison with the SIFT descriptor and the fact the it uses less information, the SURF–SURF combination should run faster than SIFT–SIFT. This does not seem to be the case. In most of the results the accuracy of SURF–SURF is almost comparable to SIFT–SIFT which could be one of the explanations. This means that the parameters are tuned to achieve higher accuracy which then affects the speed of the algorithm.

By using feature tracking we are effectively eliminating the use of descriptors and as a result of that we can see the impact on the performance. The error increases for all detectors, especially the translation error, but the execution time decreases. And it seems that SURF and SIFT are affected significantly when used in a tracker without their corresponding descriptor, indicating that these descriptors are slow but at the same time crucial to a correct feature matching. The change of performance in absence of descriptors is however also supported by the theory. In the case of SIFT descriptors, for every feature, a vector of 128 number of floating precision is used yielding a total of 512 byte/feature. For the SURF descriptor the number is 256 byte/feature. This means that the run-time will be affected, especially if memory management in C++ is not done carefully. Another interesting result when using the KLT tracker is the one obtained by the FAST detector. For instance by referee-

ing to the trajectories of the vehicle in Figure 4.4a and 4.6a and comparing them to the corresponding trajectories with the matcher (Figure 4.1a and 4.3a), we see a significant difference. The results obtained using the matcher follow the ground truth much better than when using the KLT tracker. This particular behaviour suggests that the FAST detector alone, in absence of ORB descriptor, does not perform well compared to the other detectors. This is a result of two different factors. One of the factors is that the KLT tracker is not well tuned. By tuning the parameters of the tracker one would expect to get better results. The other factor is that the FAST detector simply is not well suited by itself in terms of accuracy.

5.2 Implementation

It is important to realize that the results are not only affected by the choice of detectors and descriptors. In section 2.7.1 and 2.7.2 we discussed different methods for motion estimation, mainly 2D–*to*–2D and 3D–*to*–2D, which would also change the performance of the overall algorithm. If the chosen implementation had been the latter, several things would have been different. First of all the RANSAC algorithm would be faster, as seen in Table 2.1, due to the number of correspondence points used in the P3P algorithm. Besides this gain in run-time, the accuracy could also be better by using Bundle Adjustment[1] since this optimizes the pose with respect to features detected in several consecutive images [36].

5.3 Future work

In order to improve the results of the Visual Odometry, there are several interesting ideas that could be investigated. Besides the detectors and descriptors used in this thesis, a numerous others are available which might even perform better. Even if the proposed detectors descriptors are not changed, some code optimization could be considered. For instance the detection of features could be done using the GPU for a faster execution time or even FPGA could be used. And the feature matching could implemented by utilizing parallelized loops in the C++ code, which could improve the performance.

It would also be interesting to considering 3D–*to*–2D for motion estimation by implementing P3P algorithm. This choice of implementation enables the opportunity to use Bundle Adjustment (BA) which could improve the accuracy, even if a small number of images and points are accounted for [36]. If it is possible to use stereo cameras the triangulation for the BA and P3P would be more convenient.

[1]Bundle Adjustment (BA) is the concept of of minimizing the reprojection error of the triangulated features with respect to several different features and images. This is done by optimizing the transformation matrix and the triangulated features until the reprojected points yield a minimum error in those images. The complexity of BA increases with the number of images and triangulated features used in the optimization resulting in a increase in run-time.

Currently there are even more advanced methods which can achieve better results than Visual Odometry. One such method where the ego-motion is obtained as a consequence of the algorithm is Simultaneous Localization and Mapping (SLAM). The purpose of SLAM is to build a map of the surroundings to together with the position of the camera in the map. Visual Odometry is in fact a special case of SLAM when some of the parts of SLAM is removed. One of the benefits of SLAM is the use of Loop Closure. Loop Closure is when the SLAM algorithm recognizes a particular part of the map where the camera has already visited. And as a result of that the algorithm corrects the position and map by comparing it to previous state. However by using BA and Loop Closure along with a 3D-map of the environment, more work has be put to code optimization in order to run the algorithm in real-time.

Bibliography

[1] Semcon. Semcon in new research project: Self-driving cars simplify logistics. `http://www.semcon.com/en/About/News/Self-driving-cars-simplify-logistics/`, 2016. (Acc 2016-08-11).

[2] De Cecco M. A new concept for triangulation measurement of AGV attitude and position. *Measurement Science and Technology*, 11(11), 2000.

[3] Jing C. and Wushan C. Study On The Path Tracking And Positioning Of Wheeled Mobile Robot. *International Journal of Computer Science & Engineering Survey*, 6(3), 2015.

[4] Tow Y. T., Joo S. C., and Wan L. Y. Design of a Positioning System for AGV Navigation. *Seventh International Conference on Control, Automation, Robotics And Vision*, 2002.

[5] Scaramuzza D. and Fraundorfer F. Visual Odometry: Part I - The First 30 Years and Fundamentals. *IEEE Robotics and Automation Magazine*, 18(4), 2011.

[6] Fraundorfer F. and Scaramuzza D. Visual Odometry: Part II - Matching, Robustness, and Applications. *IEEE Robotics and Automation Magazine*, 19(2), 2012.

[7] Sturm S. and Triggs B. A Factorization Based Algorithm for Multi-Image Projective Structure and Motion. *LNCS*, pages 710–720, 1996.

[8] Ceylan D., Mitra N. J., Zheng Y., and Pauly M. Coupled Structure-from-Motion and 3D Symmetry Detection for Urban Facades. *ACM Transactions on Graphics*, 33(1), 2014.

[9] Song S. Real-Time Monocular Large-scale Multicore Visual Odometry. *UC San Diego Electronic Theses and Dissertations*, 2014.

[10] Kostavelis I., Boukas E., Nalpantidis L., and Gasteratos A. Stereo-based Visual Odometry for Autonomous Robot Navigation. *International Journal of Advanced Robotic Systems*, 13(21), 2016.

[11] Christensen et al. *Intelligent Autonomous Systems 11.* IOS Press, Amsterdam, NLD, 2010.

[12] Miksik O. and Mikolajczyk K. Evaluation of Local Detectors and Descriptors for Fast Feature Matching. *Pattern Recognition (ICPR), 2012 21st International Conference on*, 2012. IEEE.

[13] Sarvrood Y. B., Hosseinyalamdary S., and Gao Y. Visual-LiDAR Odometry Aided by Reduced IMU. *ISPRS International Journal of Geo-information*, 5(1), 2016.

[14] Durrant-Whyte H. and Bailey T. Simultaneous Localization and Mapping: Part I. *IEEE Robotics & Automation Magazine*, 13(2), 2006.

[15] Bailey T. and Durrant-Whyte H. Simultaneous Localization and Mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3), 2006.

[16] Künsch H. R. Particle filters. *Bernoulli*, 19(4):1391–1403, 2013.

[17] Todoran H. G. and Bader M. Extended kalman filter (EKF)-based local SLAM in dynamic environments: A framework. *Advances in Intelligent Systems and Computing*, 371:459–469, 2016.

[18] Govender N. Evaluation of Feature Detection Algorithms for Structure from Motion. *Council for Scientific and Industrial Research, Pretoria*, 2009. Technical Report.

[19] Lowe D. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[20] Mikolajczyk K. Detection of local features invariant to affine transformation. *Ph.D thesis, Institut National Polytechnique de Grenoble, France*, 2002.

[21] SIFT License. `http://www.cs.ubc.ca/~lowe/keypoints/`, 2016. (Acc 2016-07-26).

[22] Ehsan S., Clark A. F., Ur Rehman N., and McDonald-Maier K. D. Integral Images: Efficient Algorithms for Their Computation and Storage in Resource-Constrained Embedded Vision Systems. *Sensors*, 15, 2015.

[23] Bay H., Ess A., Tuytelaars T., and Van Gool L. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 110:346–359, 2008.

[24] Graps A. An Introduction to Wavelets. *IEEE Computational Science and Engineering*, 2(2):50–61, 1995.

[25] Rosin P. L. Measureing Corner Properties. *Computer Vision and Image Understanding*, 73(2), 1999.

[26] Rublee E., Rabaud V., Konolige K., and Bradski G. ORB: An efficent alternative to SIFT or SURF. *IEEE International Conference on Computer Vision*, pages 2564–2571, 2011.

[27] Tomasi C. and Kanade T. Detection and Tracking of Point Features. *Carnegie Mellon Univeristy Technical Report CMU-CS-91-132*, 1991.

[28] Suhr J. K. Kanade - Lucas - Tomasi (KLT) Tracker, 2009.

[29] Lucas B. and Kanade T. An Iterative Image Registration Technique with an Application to Stereo Vision. *Proc 7th Intl Joint Conf on Artificial Intelligence (IJCAI)*, pages 674–679, 1981.

[30] Shi J. and Tomasi C. Good Features to Track. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 539–600, 1994.

[31] Nistér D. An Efficient Solution to the Five-Point Relative Pose Problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):756–777, June 2004.

[32] Kneip L., Scaramuzza D., and Siegwart R. A Novel Parametrization of the Perspective-three-point Problem for a Direct Computation of Absolute Camera Position and Orientation. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2969–2976. IEEE Computer Society, 2011.

[33] Hartley R. and Zisserman A. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[34] Hartley R. and Sturm P. *Triangulation*, pages 190–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[35] Geiger A., Lenz P., and Urtasun R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. `http://www.cvlibs.net/datasets/kitti/eval_odometry.php`, 2012. (Acc 2016-07-26).

[36] Zhang Z. and Shan Y. Incremental Motion Estimation Through Modified Bundle Adjustment. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference ON*, volume 2, pages II–343–6 VOL.3, Sept 2003.

A

Appendix 1

A.1 Parameters

	Name	Value
FAST	Threshold	25
	Non-max suppression	True
	Type	TYPE_9_16
SURF	Hessian threshold	400
	Octaves	4
	Octave layers	2
	Descriptor elements	64
	Orientations	True
SIFT	Number of feature	UNLIMITED
	Octave layers	3
	Contrast threshold	0.04
	Edge threshold	10
	Sigma	1.6
FLANN	Number of trees	4
	Checks	50
	Eps	0
	Sorted	True
KLT	Termination criteria	iterations & accuracy
	Maximum iterations	30
	Accuracy	0.001
	Window size	21×21
	Maximum levels	3
RANSAC	Desired confidence	99.9%
	Threshold	0.5

Table A.1: A list of parameters used to produce results.