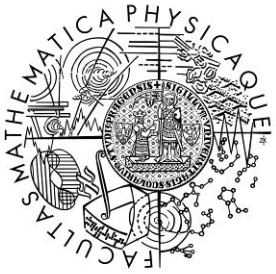


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

# MASTER THESIS

Peter Zborovský

## Scene Depth Estimation Based on Odometry and Image Data

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. David Obdržálek, Ph.D.
Study programme: Computer Science
Specialization: Artificial Intelligence

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... signature

ACKNOWLEDGEMENT

I would like to thank my thesis advisor, RNDr. David Obdržálek, Ph.D. from the Faculty of Mathematics and Physics at Charles University. The door to Prof. Obdržálek's office has always been open whenever I ran into a trouble spot or had a question about my research or writing. He consistently cared for this paper to remain my own work, but always steered me in the right direction whenever he thought I needed it.

Title: Scene Depth Estimation Based on Odometry and Image Data

Author: Peter Zborovský

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract:

In this work, we propose a depth estimation system based on image sequence and odometry information. The key idea is that depth estimation is decoupled from pose estimation. Such approach results in multipurpose system applicable on different robot platforms and for different depth estimation related problems. Our implementation uses various filtration techniques, operates real-time and provides appropriate results. Although the system was aimed at and tested on drone platform, it can be well used on any other type of autonomous vehicle that provides odometry information and video output.

Keywords: Depth estimation, Triangulation, Odometry, Feature tracking, Filtering.

# CONTENTS

# Introduction

In robotics there are many problems related to depth perception of the surrounding scene (collision avoidance, object tracking, object following, etc.). Each approach to these problems may have different assumptions about the scene or the robot platform. For instance, in [44] authors estimate depth of objects with glossy surface using Light-field cameras. Approach in [41] expects non-metallic smooth surfaces of objects and requires the use of linear polarising filter or in [34] RGB-D cameras are used for depth estimation.

In 2007, first widely used real-time SLAM (Simultaneous Localization and Mapping) algorithm called PTAM ([24]) was proposed. Since then, SLAM like algorithms were rapidly evolving. DTAM in 2011 ([32]), LSD-SLAM in 2014 ([6]), as well as SVO ([9]) and recently DSO algorithm ([5]) was proposed. All of these were state-of-the-art SLAM or VO (Visual Odometry) algorithms and each of them outperformed the previous one. Hence, we can say with no doubt that the robotics and more specifically computer vision algorithms are rapidly evolving. Considering wide range of depth estimation related problems and fast progressing robotic environment, it makes sense to think of a modular system that would allow to use different odometry and feature tracking algorithms according to application purpose and provide appropriate depth map.

A key idea is in decoupling depth estimation from the motion estimation, as such approach is providing high variability in terms of robot platform and application purpose.

Our main contribution is a software that is able to estimate relative depth of salient features in the scene, it is independent on pose estimation of the robot (that can use lasers, wheel or visual odometry, or any other pose estimation system that suits the robot platform), and independent on image processing algorithm that can be voluntarily chosen based on application purpose. For implementation reasons, we have chosen the visual odometry that is suitable for drone like platforms and for evaluation purposes we have chosen salient feature tracking rather than single object tracking as more points provide more complex information that can be evaluated to greater extent.

# 1 Related work

The problem of depth estimation is common in robotics. For instance, earlier works, such as [31] were focused on classic stereo depth estimation where depth calculation is based on image disparity of two cameras mounted on a static stereo-rig, or in [36], the authors propose algorithm that combines monocular cues with stereo vision in order to achieve more accurate dense depth map. However, for such approaches the setup of 2 calibrated cameras is necessary, which is not always possible.

Many monocular depth estimation approaches are based on machine learning algorithms like [4] and [26], where monocular cues are taken into consideration. In [4], the authors propose a Deep Convolutional Neural Network for depth estimation of the scene. They employ two deep network stacks: one for global prediction of the depth and second one for refining the global prediction locally. In [37], the authors use Markov random field to infer plane parameters that capture the 3-space location and the 3-space orientation of the patch using superpixel segmentation algorithm. The Markov random field is trained via supervised learning. In more recent work ([26]), the authors builds on findings of [37] and use superpixel segmentation as well. Their approach outperforms also prior work of [4] by transforming depth estimation into a continuous conditional random field learning problem. The main disadvantage of machine learning approach is the necessity of reliable training dataset with a ground truth depth map. Such a system may not work properly in scenarios omitted in training process. There are also works like [13], where authors propose unsupervised learning algorithm of image sequences (i.e. no single image depth estimation) requiring known camera poses in a global frame.

Different approaches to monocular depth estimation take advantage of geometrical constraints arising from motion. These approaches are related to the structure from motion problem like [24], where authors parallelize pose estimation and scene reconstruction. In more recent work [33], the dense depth map is estimated on per-pixel basis by probabilistic depth measurement. The depth filter is assigned to each pixel not only to estimate its depth but also to reject erroneous measurements.

In [21], the authors follow the work of [9] and [33] in terms of operating directly on pixel intensities. The work is very similar to [9], [6], [32], [5] or [33]. The difference is that authors have imposed the gradient component in photometric error computation in the mapping thread with aim to obtain a dense depth-map - it makes the system more accurate in peripheral areas, however, such a dense depth map is obtained at the cost of greater time consumption.

All mentioned works tend to build depth maps in regard to some assumptions about the scene or platform requirements. Our work tends to provide a more general solution in terms of robot platform (different sensors availability) and application purpose (object tracking, object following or even dense depth reconstruction).

# 2 Problem statement

In this chapter, we discuss depth estimation problem in general, and in subsections, we provide underlying theoretical background for motion estimation and image processing that are necessary for depth estimation understanding.

Suppose the situation in figure 1. We have a moving camera with its centre at point $C_k$ in space observing the scene at discrete time steps $t = 1, ..., n$. We have a set of observed features $x_{1,k}, ..., x_{m,k}$ on the image plain $I_k$ at time step $k$. Each feature has its position on the image plain $x = (x_x, x_y)^T$ and corresponds to a 3-space point $X = (X_x, X_y, X_z)^T$. Any observed 3-space point $X$ is mapped to the image coordinates $x$ through the camera projection model $P : \mathbb{R}^3 \rightarrow \mathbb{R}^2$:

$$x_{i,k} = P(X_i).$$

The 3-space point $X_i$ corresponding to the image point $x_i$ can be recovered, given the inverse projection function $P_k^{-1}$ and the depth $d_{i,k}$:

$$X_i = P_k^{-1}(x_{i,k}, d_{i,k}), \tag{1}$$

where the depth of the feature $d_{i,k}$ is unknown. So, we can formulate the problem as follows: Given two positions of the camera $C_{k-p}$ and $C_k$ at different timestep $(p \neq 0)$ and two sets of corresponding features $\{x_{i,k-p}\}$ and $\{x_{i,k}\}$ estimate depth of the corresponding 3-space points $\{X_i\}$.

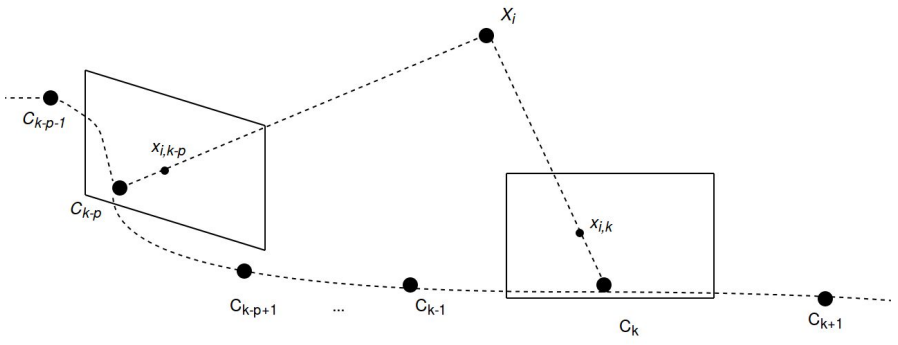


**Fig. 1. Depth estimation with moving camera.** Depth estimation from stereo correspondences of selected features.

In order to determine depth of each feature, we first need to determine the camera motion (Pose estimation) and correctly detect and track the features that correspond to observed 3-space points in the scene.

## 2.1 Epipolar Geometry

Epipolar geometry describes the relation between two camera views of the same scene ([48]). "The epipolar geometry is the intrinsic projective geometry between two views"[1]. It does not depend on the scene structure. Important parameters here are the first and second camera intrinsic parameters and their relative position [19].

The main concept of epipolar geometry is shown in figure 2.

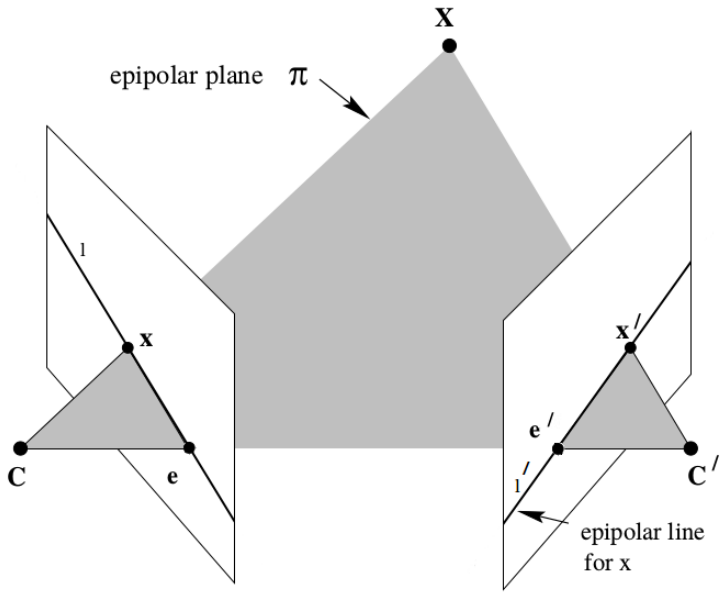


**Fig. 2. Epipolar geometry.** The camera centres $C$ and $C'$ form the baseline between the two views and together with space point $X$ and its images $x$ and $x'$ they form an epipolar plane $\pi$. The ray formed by camera center $C$ and image point $x$ is projected onto second image plane as epipolar line $l'$. The baseline intersects both image planes at epipoles $e$ and $e'$. ([19], 2004).

[1] Hartley et al., Multiple View Geometry in Computer Vision, Cambridge University Press, 2004. page 239.

### 2.1.1 Terminology

**Epipole**. "Epipoles are intersections of all epipolar lines." [2] The pipole is a point on the image plane which is formed by the intersection of the baseline (line through the camera centres) and the image plane. The epipole on the first image is the only point back-projected onto the second image plane as a point (not a ray) and vice versa.

**Epipolar line**. The epipolar line is a "line resulting from the intersection of the epipolar plane with the image plane." [3]

**Epipolar plane**. It is a plane formed by 3-space point $X$ and both camera centres $C$ and $C'$ [39]. "There is a one-parameter family (a pencil) of epipolar planes." [4] Pencil of epipolar planes is a line which is formed by the intersection of all epipolar planes [19].

### 2.1.2 Epipolar constraint

Consider the situation in figure 2. Assume that camera centres $C$ and $C'$ are known as well as image point $x$. The line defined by the first camera center $C$ and the image point $x$ is back-projected onto the second image plane as (epipolar) line $l'$. Then the image point $x'$ (the projection of unknown 3-space point $X$ onto the second image plane) is constrained to lie within this line ([48]). We will algebraically define this constraint in the following section.

## 2.2 Motion estimation

The goal is to retrieve the relative pose of camera centres from the set of corresponding points in two image planes. The relation between the corresponding image points and the relative camera positions is defined by the fundamental matrix which is an algebraical representation of the epipolar constraint (among other properties).

---

[2] Xu et al., Epipolar Geometry in Stereo, Motion and Object Recognition. Springer Science + Business Media Dordrecht, 1996. page 32.

[3] Sharipo et al., Computer Vision. Prentice hall. 2001. page 438.

[4] Hartley et al., Multiple View Geometry in Computer Vision, Cambridge University Press, 2004. page 241.

### 2.2.1 Fundamental matrix

"The fundamental matrix is the algebraic representation of epipolar geometry."[5] As explained in previous section, there is a corresponding epipolar line in the second image plane to each point on the first image plane. Thus, there is a following map:

$$x \rightarrow l'.$$

This relation is a projective mapping from points to their corresponding epipolar lines and it is represented by a fundamental matrix, denoted F ([19]).

Following derivation of fundamental matrix is performed according to Hartley and Zisserman ([19]). Considering situation in figure 2, we may derive the following. The 3-space point $X$ is projected onto the first image plane as $x = PX$. Where $P$ is the projection matrix of the first camera. The solution for $X$ is given by inverse projection function (1):

$$X(\lambda) = P^{-}x + \lambda C$$

where $P^{-}$ is a pseudoinverse of $P$, $C$ is the camera centre of the first camera and $\lambda$ is the scaling parameter as the distance of $X$ from camera centre cannot be known from a single view. To obtain a projected epipolar line in the second image, we need a projection of two points on a the line that is formed by point $x$ on the first image plane and the first camera center $C$. First easily calculated point is camera centre $C$ which is projected onto the second image plane as $P'C$. The second point is represented by the back-projection of the point $x$ to the unknown depth, i.e. $P^{-}x$, which is projected onto the second image plane as $P'P^{-}x$. Note that we have retrieved these points by setting the scaling parameter $\lambda$ to infinity and to zero respectively. The epipolar line $l'$ can now be derived: $l' = (P'C) \times (P'P^{-} + x)$. Point $P'C$ is the projection of first camera centre onto the second image plane and so it

[5] Hartley et al., Multiple View Geometry in Computer Vision, Cambridge University Press, 2004. page 241.

represents the epipole of the second image plane (denoted $e'$). Now, we can write $l' = [e_x']_\times (P'P^-)x = Fx$, where $F$ is the fundamental matrix:

$$F = [e_x']_\times P'P^-,$$

where $[...]_\times$ represents a skew symmetric cross product matrix. [6] Further, we can derive:

$$l' = Fx. \tag{2}$$

Since $x'$ lies on the epipolar line $l'$, we can write: $x'^T l' = 0$. From this observation and (2) we can see that

$$x'^T Fx = 0. \tag{3}$$

According to [19] and [43], this is one of the most important observation as it allows to calculate the Fundamental matrix solely from the image point correspondences, i.e. without prior knowledge of camera matrices $P$ and $P'$. Equation (3) is an algebraic representation of the epipolar constraint described geometrically in the previous section.

### 2.2.2 Visual odometry

In the previous section, we have described the basic idea behind motion estimation based on visual input. In this section, we describe one of the state-of-the-art visual odometry systems that is fast and highly accurate - SVO - semi-direct visual odometry ([9], [10]). It is derived from SLAM like algorithms (Simultaneous Localization and Mapping) in which the pioneering work was done by Klein and Murray ([24], 2007) in their work Parallel Tracking and Mapping for Small AR Workspaces. Since then, a whole area of SLAM methods has evolved. The algorithms are now divided into SLAM algorithms, that, except for camera path, also create a map (these approaches are also referred to as Structure from Motion algorithms) and Visual Odometry algorithms that are oriented solely on creating the accurate camera path from visual input (usually creating also a sparse point map). A more detailed insight on the field and its history is provided in [38].

[6] Explained in Szeliski R., *Computer Vision: Algorithms and Applications*. Springer. 2010. page 42.

#### 2.2.2.1 SVO - semi direct visual odometry

SVO is a monocular visual odometry algorithm that calculates the global camera pose solely from the visual (camera) input. The algorithm operates directly on image intensities (direct method). Rather than evaluating the intensity differences along the whole image, only small patches around tracked features are used instead (sparse model), which significantly reduces the computational time.

##### 2.2.2.1.1 Motion Estimation Thread

Firstly, a frame-to-frame motion is estimated by minimizing the intensity difference of detected features that corresponds to the projected real-world 3D points (This process is called Sparse Model-based image alignment). In order to work with picture intensities, small patches around detected features are used. For computational reasons, these patches are not transformed. Such approach requires a small frame-to-frame motion and not big patches. The sparse image alignment aligns the camera pose of a new frame with respect to the previous frame.

The second step optimizes the camera pose with respect to the 3D map, but at the cost of violating the geometrical constraints (relaxation step).

In the last step, both the camera pose and the 3D map are optimized in order to minimize the reprojection error introduced by the previous optimization step (this step is called a local bundle adjustment).

##### 2.2.2.1.2 Mapping Thread

The key contribution of Mapping Thread is a depth filter. The depth filter is assigned to each detected feature. The filter iteratively refines the probability distribution of the feature depth.

Each new frame updates the depth filter of the detected features - the variance of features depth is reduced. When the variance is small enough, the depth estimate is used to calculate a new 3D point in the map.

Another important part of mapping thread is the keyframe selection. When sufficient distance between current frame and last keyframe is detected, the current

frame is selected to be a new keyframe. Keyframes are used to extract new features and for pose and structure optimization.

Further details and derivations can be found in [9] and [10].

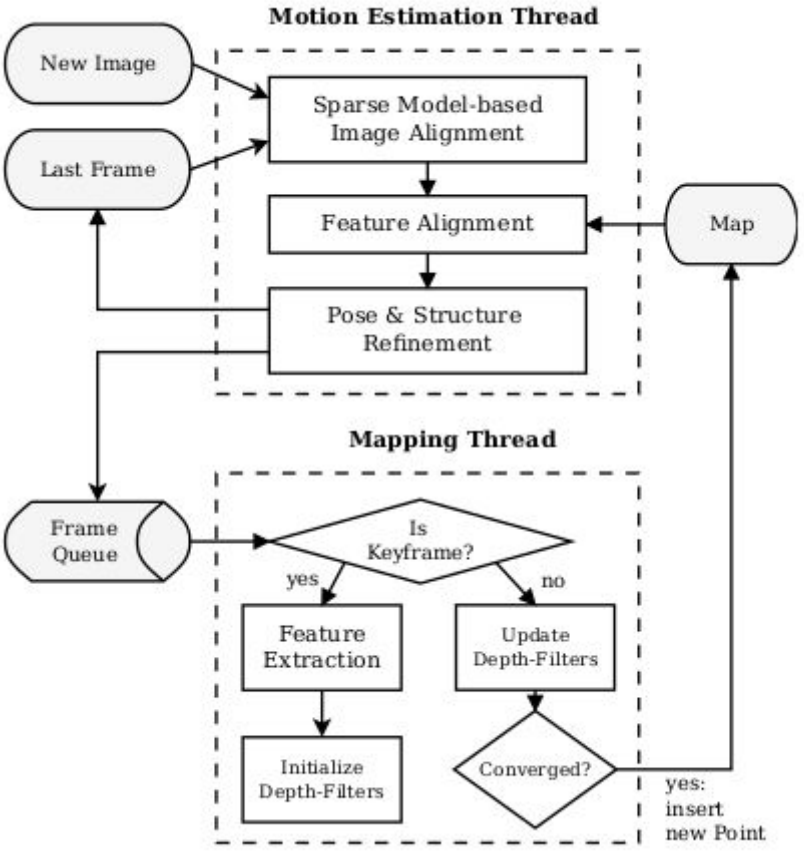


**Fig. 3. SVO pipeline.** ([9], 2014)

# 2.3 3D reconstruction

Triangulation is "the problem of determining a point's 3D position from a set of corresponding image locations and known camera positions." [7]

"We suppose that a point $X$ in $\mathbb{R}^3$ is visible in two images. The two camera matrices $P$ and $P'$ corresponding to the two images are supposed known. Let $x$ and $x'$ be projections of the point $X$ in the two images. From this data, the two rays in

[7] Szeliski R., *Computer Vision: Algorithms and Applications*. Springer. 2010. page 305.

space corresponding to the two image points may easily be computed. The triangulation problem is to find the intersection of the two lines in space." [8]

### 2.3.1 Linear triangulation

Considering the situation in figure 2, one can form equations to reconstruct the 3-space point $X$ from its projections in two image planes. The Following applies to both image points: $x = PX$ and $x' = P'X$. "First the homogeneous scale factor is eliminated by a cross product to give three equations for each image point, of which two are linearly independent."[9] i.e. $x \times PX = 0$. Hartley and Zisserman [19] then provide the following derivation:

$$\begin{aligned} x_1(p^{3T}X) - (p^{1T}X) &= 0, \\ x_2(p^{3T}X) - (p^{2T}X) &= 0, \\ x_1(p^{2T}X) - x_2(p^{1T}X) &= 0, \end{aligned} \tag{4}$$

where $x_1$ and $x_2$ are the coordinates of image point $x$ and $p^{iT}$ represents the i-th row of the matrix $P$. These equations can be directly rewritten as $AX = 0$. By eliminating the last row of (4), since it is linearly dependent on the first two rows, we get:

$$A = \begin{bmatrix} x_1 p^{3T} - p^{1T} \\ x_2 p^{3T} - p^{2T} \\ x_1' p^{3T} - p^{1T}, \\ x_2' p^{3T} - p^{2T}, \end{bmatrix},$$

where two equations for a second view are added. $AX = 0$ is linear in $X$ and has a unique solution in the perfect case which can easily be computed as a set of linear equations.

---

[8] Hartely, R., et al., *Triangulation*. ARPA Image Understanding Workshop 1994, 1994. page 957.

[9] Hartley et al., Multiple View Geometry in Computer Vision, Cambridge University Press, 2004. page 312.

In presence of noise in image points measurements (all or almost all real-world scenarios), the direct method will fails. As illustrated in figure 4, due to noisy measurements, the line defined by the first camera centre $C$ and the image point $x$ does not intersect with a line defined by the second camera centre $C'$ and image point $x'$.

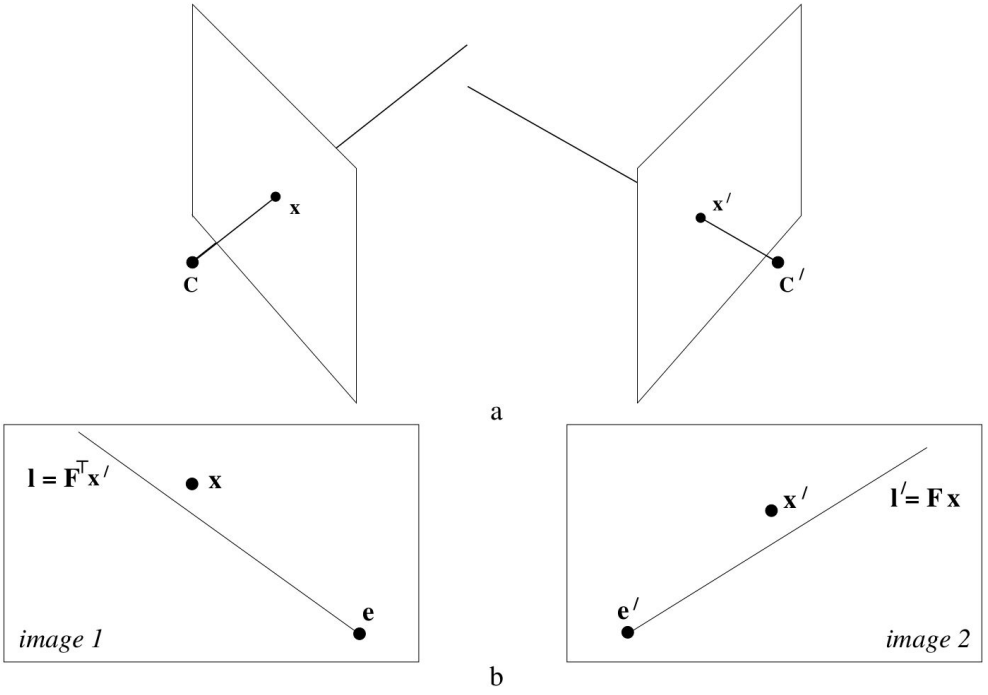


**Fig. 4. Measurement errors.** a) In the presence of noise in image points measurements, the lines formed by the camera centres and image points do not intersect in 3-space. b) Violation of epipolar constraint - point $x$ does not lie on epipolar line formed by image point $x'$ and the same is valid for point $x'$ and epipolar line in image 1. ([19], 2004).

The authors in [19] and [43] describe two ways for solving $AX = 0$ - homogeneous and inhomogeneous.

**Homogeneous method**. "Note that if we use homogeneous coordinates $X = (X, Y, Z, W)$, the resulting set of equations is homogeneous and is best solved as a singular value decomposition (SVD) or eigenvalue problem (looking for the smallest singular vector or eigenvector)."[10] Here we solve 4 equations of 4 unknown

[10] SZELISKI R., Computer Vision: Algorithms and Applications. Springer. 2010. page 306.

parameters. To avoid a zero solution for $AX = 0$, an additional constraint is required - $||X|| = 1$ ([19]).

The following algorithm is according to Hartley and Zisserman [19].

| |
|---|
| i) compute the matrix $A$ from matrices $P$ and $P'$ and from corresponding image points $x$ and $x'$.<br>ii) calculate SVD of $A$. The unit singular vector corresponding to the smallest singular value is the solution $X$. Specifically, if $A = UDV^T$ with $D$ diagonal with positive diagonal entries, arranged in descending order down the diagonal, then $X$ is the last column of $V$. |

**Algorithm 1. DLT.** Basic Direct Linear Algorithm for the estimation of 3-space point $X$. ([19], 2004).

Singular vector or eigenvector in the context of transformation matrix can be understood as a base vector whose orientation (and in this case also the magnitude) remains unchanged under this transformation, which holds for observed 3-space point $X$. For a detailed derivation of SVD, we refer reader to [42].

Inhomogeneous method. By setting the 4th homogeneous coordinate $W$ to 1 , the problem is reduced to estimation of 3 unknown parameters from 4 equations and resolved in least squares fashion ([43]) or by using methods for solving linear equations such as Gaussian elimination ([19]). However, this "system may be singular or poorly conditioned, i.e., if all of the viewing rays are parallel, as occurs for points far away from the camera."[11] That is the case when $W = 0$, which holds for points at infinity. Such cases have no solution and that is why the homogeneous method is preferred in general ([19]).

The problem of linear triangulation is that the homogeneous method is neither affine invariant nor projective invariant and inhomogeneous method is not projective invariant. "To see this, suppose that camera matrices $P$ and $P'$ are replaced by $PH^{-1}$ and $P'H^{-1}$. One sees that in this case the matrix of equations, $A$, becomes $AH^{-1}$. A point $X$ such that $AX = \epsilon$ for the original problem corresponds to a point $HX$ satisfying $(AH^{-1})(HX) = \epsilon$ for the transformed problem. Thus, there is a one-to-one

[11] SZELISKI R., Computer Vision: Algorithms and Applications. Springer. 2010. page 306.

correspondence between points $X$ and $HX$ giving the same error. However, neither the condition $||X|| = 1$ for the homogeneous method, nor the condition $X = (X, Y, Z, 1)^T$ for the inhomogeneous method, is invariant under application of the projective transformation $H$. Thus, in general the point $X$ solving the original problem will not correspond to a solution $HX$ for the transformed problem."[12]

There are also other triangulation methods (using Geometric cost function, Sampson approximation, etc.). We refer reader to [19] and [18] for further reading.

## 2.3.2 Image rectification

Image rectification is "the process of resampling pairs of stereo images taken from widely differing viewpoints in order to produce a pair of 'matched epipolar projections'. These are projections in which the epipolar lines run parallel with the *x*-axis and match up between views, and consequently disparities between the images are in the *x*-direction only, i.e. there is no *y* displacement."[13]

Image rectification preserves camera centres but changes both cameras orientation and intrinsic parameters. After rectification, the orientation of the first camera equals to the orientation of the second camera and the intrinsic parameters of the first camera equals to the intrinsic parameters of the second camera ([12]).

One way to rectify the images is first to change the orientation of the cameras so that it is perpendicular to the baseline (line joining the camera centres) and then to change the up vector of both images so that they are parallel to each other and perpendicular to the baseline ([43]). Another way is to choose such a projective transformation $H$ for the first image plane that maps its epipole to the point at infinity and exhibits minimal projective distortion and then to find transformation $H'$ for the second image plane that match up the epipolar lines ([17]). We describe the latter method according to [17]. This method is based on the properties of fundamental matrix.

Consider the following projective transformation:

---

[12] Hartley et al., Multiple View Geometry in Computer Vision, Cambridge University Press, 2004. page 313.

[13] HARTLEY, R. I. Theory and Practice of Projective Rectification. International Journal of Computer Vision. 1999. page 115.

$$H = GRT\,, \tag{5}$$

where $T$ represents a translation of the selected image point $x_0$ to the origin, $R$ represents a rotation around the image centre that maps the epipole $e$ to the point $(f, 0, 1)^T$ so that it is parallel to the *x*-axis and $G$ is a transformation taking the translated and rotated epipole to infinity, i.e. to point $(f, 0, 0)^T$. Transformation $H$ (5) is to first-order a rigid transformation in the neighbourhood of $x_0$. Note that (5) maps epipole to the point at infinity lying on the *x*-axis, and so all epipolar lines are parallel to the *x*-axis ([17]).

Given $H$ (5) for the first image plane *I* the goal is to find the matching transformation $H'$ for the second image plane *I'* so that $H^{-T} l = H'^{-T} l'$ holds for the corresponding epipolar lines $l$ and $l'$ and so that the sum of squared disparities $\sum_i d(Hx_i,\ H'x'_i)^2$ is minimized ([17]).

In [17], the author proves that for $M = P'P^-$ (see section 2.2.1) and the epipole $e$ in the first image plane the corresponding transformation for the second image plane $H'$ that matches the first transformation $H$ is

$$H' = H_A H M\,,$$

where $H_A$ is an affine transformation of the form

$$H_A = \begin{bmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6}$$

Given $H$ we can find matching transformation $H'$ that minimizes the disparity by finding $H_A$ (6) that minimizes the squared distances:

$$\sum_i d(Hx_i,\ H_A H M x'_i)^2\,. \tag{7}$$

Finding $H_A$ that minimizes (7) is linear in $a, b, c$ (6) ([17]). Now we can outline the rectification algorithm:

Given fundamental matrix $F$ for two image planes $I$ and $I'$:

1. Select a projective transformation $H$ for the first image plane that maps the epipole $e$ to the point at infinity $(1, 0, 0)^T$ (according to (5)).
2. Find the matching projective transformation $H'$ that minimizes the least-squares distance (7).

**Algorithm 2. Image rectification.** ([17], 2004)

Rectified image geometry results in a simplified epipolar search for feature matching and it is used in many stereo matching algorithms (e.g. triangulation - see section 2.3.1). For a more detailed derivation, we refer reader to [19], [17], [12].

# 2.4 Feature detection, matching and tracking

Feature detection, matching, and tracking are building blocks for many computer vision applications. "In image processing and computer vision tasks, we need to represent the image by features extracted therefrom. The raw image is perfect for the human eye to extract all information from; however, that is not the case with computer algorithms."[14]

In this section, we will describe basic ideas behind feature detection, feature description, feature matching and tracking, and discuss some of the well-known and widely used algorithms.

## 2.4.1 Feature detection

**Feature** "is an image pattern which differs from its immediate neighborhood."[15] Features such as corners and blobs are of most interest since their position can be measured with high accuracy.

[14] HASSABALLAH, M., et al.: Image Features Detection, Description and Matching. In: Image Feature Detectors and Descriptors. Studies in Computational Intelligence. Springer. 2016. page 13.

[15] TUYTELAARS, T., et al.: Local invariant feature detectors: a survey. Foundations and Trends in Computer Graphics and Vision. 2007. page 178.

**Corner** is "a point at the intersection of two or more edges."[16]

**Blob** is "an image pattern that differs from its immediate neighbourhood in terms of intensity, colour, and texture."[17]

"During the feature-detection step, the image is searched for salient keypoints that are likely to match well in other images." [18] In [20], the authors provide following list of desirable properties of a good feature detector:

- Robustness in feature scaling, rotation, shifting, photometric deformations and noise,
- Repeatability in detecting the same feature under different viewing conditions,
- Accuracy of feature localization,
- Efficiency for real-time feature detection.

All feature detectors firstly apply a feature-response function (e.g. corner response function or gaussian (blob) response function), and consequently nonmaxima suppression is used in order to identify all local maxima and minima of feature response function. The result represents the detected features ([11]).

### 2.4.1.1 Feature response function

In general, feature detector search along the image for regions with high contrast changes (gradient) in more than one direction as the gradient of line suffers from well-known aperture problem (see figure 5). Identifying strong features is done via autocorrelation function which defines how distinguishable is the feature comparing to its close neighbourhood ([43]):

$$E_x = \sum_i w(x_i)[I\,(x_i + \Delta u) - I(x_i)]^2\,,$$

where $u$ is a displacement vector, $\Delta u$ represents small change in position of examined pixel $x_i$ in the image $I$, $w(x_i)$ is a spatially varying weighting (or window) function and the summation goes through all the pixels in the patch.

[16] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 79.

[17] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 79.

[18] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 79.

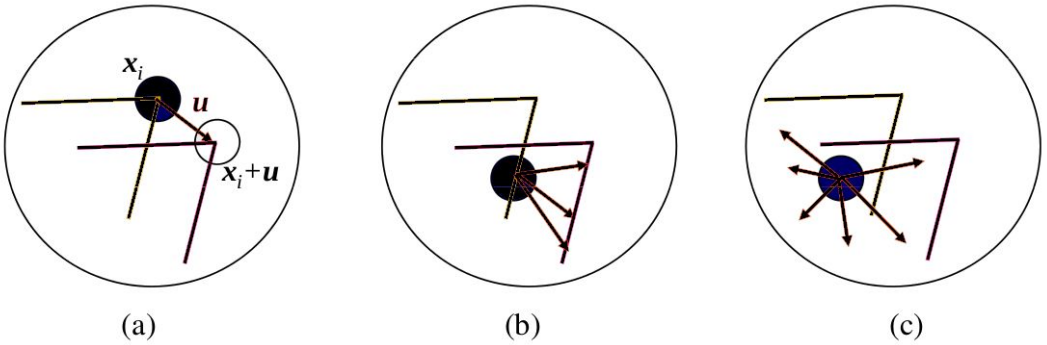


**Fig. 5. Aperture problem** for different image regions. a) corners usually do not suffer from aperture problem b) edge - "barber pole" problem c) textureless region. ([43], 2010).

Using Taylor expansion of an image function $I(x_i + \Delta u) \approx I(x_i) + \nabla I(x_i)$, we can further derive:

$$E_x = \sum_i w(x_i)[I(x_i + \Delta u) - I(x_i)]^2,$$

$$E_x \approx \sum_i w(x_i)[I(x_i) + \nabla I(x_i)\Delta u - I(x_i)]^2,$$

$$E_x = \sum_i w(x_i)[\nabla I(x_i)\Delta u]^2,$$

$$E_x = \Delta u^T A \, \Delta u,$$

where $\nabla I(x_i) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(x_i)$ is a gradient at $x_i$ and $A$ represents autocorrelation matrix ([43]):

$$A = w * \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}, \tag{8}$$

where local image derivatives $I_{xx}$, $I_{yy}$, $I_{xy}$ are calculated using weighting window $w$, which can be rectangular or circular such as a Gaussian and it results in values close to centre to be weighted more than distant ones ([20]).

### 2.4.1.2 Harris corner detector

Feature response function is used to determine strong features in the image. For corner detection typical feature, the response function is represented by Harris corner response function ([15]). "For finding interest points, the eigenvalues of the matrix $A$ are computed for each pixel. If both eigenvalues are large, this indicates existence of the corner at that location" [19] (see figure 6).

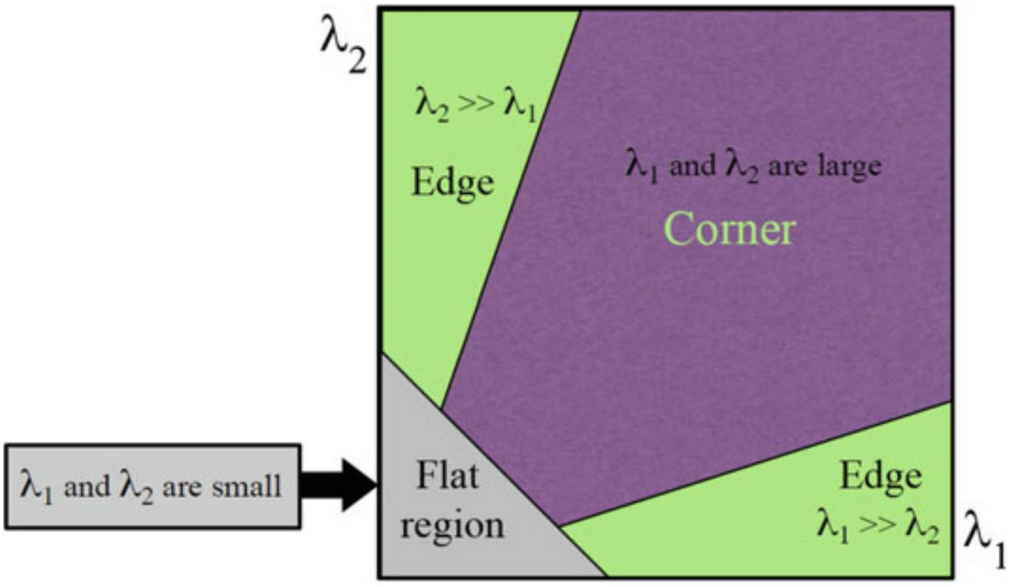


**Fig. 6. Harris corner detection** based on the eigenvalue analysis of the Autocorrelation matrix $A$. The large value of R (9) means that both eigenvalues are large and it indicates the corner region. Negative R indicates edge region and small R indicates flat region ([20], 2016).

Eigenvalues of autocorrelation matrix $A$ (8) represent a change in two orthogonal directions in a neighbourhood of the defined point ([47]).

In [11], the author proposed following corner selection criteria for corner detection:

$$R = Det(A) - k * Tr(A)^2, \qquad (9)$$

where $Det(A) = \lambda_1\lambda_2$, $Tr(A) = \lambda_1 + \lambda_2$ and $\lambda_1$, $\lambda_2$ are the eigenvalues of the autocorrelation matrix $A$. Such approach is computationally less expensive as it

[19] HASSABALLAH, M., et al.: Image Features Detection, Description and Matching. In: Image Feature Detectors and Descriptors. Studies in Computational Intelligence. Springer. 2016. page 19.

is not necessary to calculate eigenvalues of $A$. When corner selection criteria (9) exceeds certain threshold, the examined pixel is marked as a corner.

### 2.4.1.3 Shi Tomasi corner detector

Shi-Tomasi corner detector is based on the Harris corner detector with the difference in corner selection criteria (9):

$$R = min(\lambda_1, \lambda_2). \tag{10}$$

"In practice, when the smaller eigenvalue is sufficiently large to meet the noise criterion, the matrix $A$ is usually also well conditioned. In fact, the intensity variations in a window are bounded by the maximum allowable pixel value, so that the greater eigenvalue cannot be arbitrarily large."[20] In [40], the authors experimentally prove that criterion (10) has better performance than Harris corner selection criteria (9).

### 2.4.1.4 SIFT feature detector

In [27], the author proposes a scale invariant feature transform (SIFT) algorithm for blob detection that is scale and rotation invariant. Scale invariance is achieved by scale-space filtering. The Laplacian of Gaussian (LoG) (11) response function is used for determining blob-like features in windows with different $\sigma$ values in gaussian kernel. LoG is a convolution of the variable-scale Gaussian (12) with original Image $I$:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x,y), \tag{11}$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}. \tag{12}$$

Laplacian is approximated with Difference of Gaussians (DoG) which is easier to compute. The SIFT DoG is described in figure 7. After finding DoG images, they are searched for local maxima and minima over scale and space. These local extrema are often found at edge regions and are therefore to be refined. To

[20] SHI, J., et al. Good Features to Track. Technical Report. Cornell University, Ithaca, NY, USA. 1993. page 3.

reject edge-like regions, the Harris corner detector approach is used (see section 2.4.1.2).

Rotation invariance of the feature is achieved by calculation of its orientation. The neighbourhood is chosen for each keypoint. The area of the chosen neighbourhood depends on the scale of the feature. Gradient magnitude and direction is calculated for this neighbourhood. The calculation is based on an orientation histogram. For further details, we refer reader to [27].

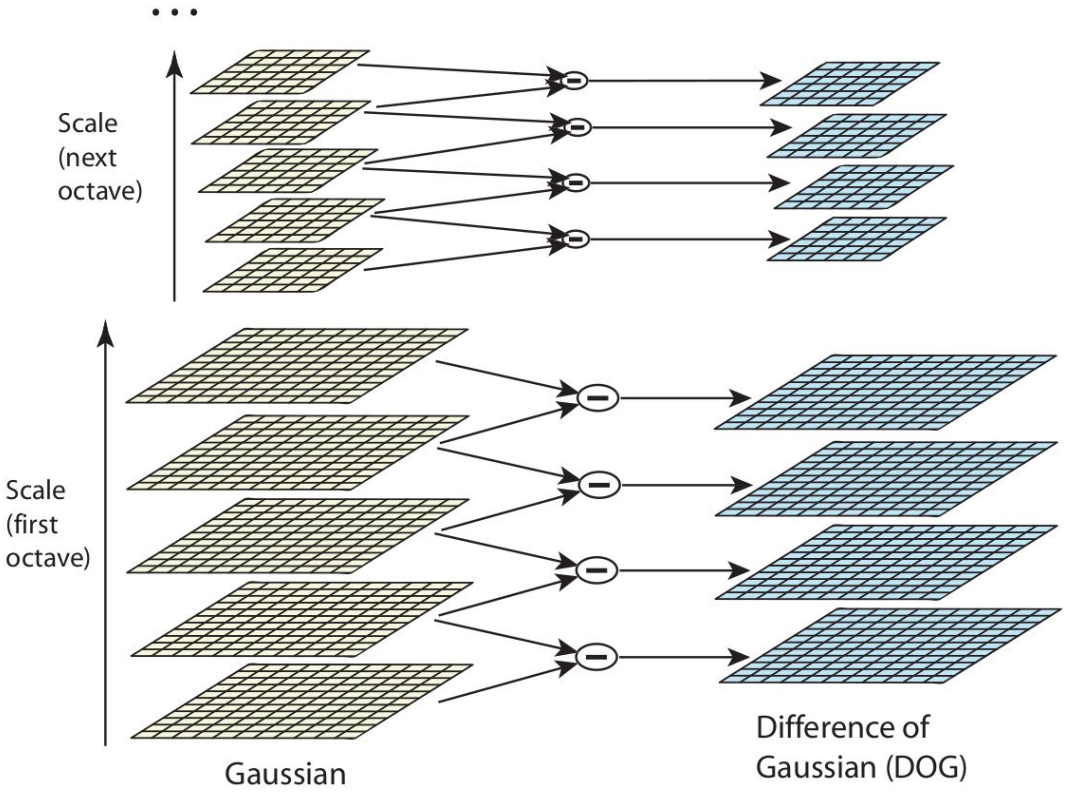


**Fig. 7. SIFT Difference of Gaussian**. For each octave, the initial image is smoothed with Gaussian filters with different σ values (images on the left). These Gaussian images are then subtracted in order to produce DoG images (images on the right). This is repeated after downsampling the image by a factor of 2. ([27], 2004).

## 2.4.2 Feature Description

"In the feature description step, the region around each detected feature is converted into a compact descriptor that can be matched against other descriptors."[21]

In cases when the local feature undergoes mostly a translational transformation (e.g. aerial footage of the ground), simple error metrics, such as sum of squared differences (SSD) or normalized cross-correlation (NCC), can be used to compare intensities. NCC shows better results comparing to SSD in the presence of slight brightness changes of features. However, in most real-world scenarios, the features undergo more severe deformations, i.e. rotation, scale change and also affine transformation. In such cases, local appearance of the feature is not sufficient because the feature orientation, scale and viewpoint is changed ([43], [11]).

In order to account for these changes, more sophisticated feature descriptors were introduced. In the following section, we detail one of the most popular SIFT descriptor ([11]).

Later on, more detectors / descriptors were proposed, which were mostly aimed at time-consumption efficiency. Most of them were based on SIFT, like SURF ([2]), that is using integral map for feature detection. A simple binary descriptor BRIEF ([3]) that is using pairwise brightness comparisons from patch around the feature, ORB ([35]) which is built on BRIEF, but it is also rotation invariant or BRISK ([25]) that is based on FAST with a binary descriptor. For further details about mentioned algorithms, we refer reader to the original papers ([2], [3], [35], [25]).

### 2.4.2.1 SIFT feature descriptor

"The SIFT descriptor is basically a histogram of local gradient orientations." [22] First, a grid of image gradients and their orientations is formed around the feature. This grid is created using a the scale of the feature at which it was detected in feature detection step (see section 2.4.1.4). "A Gaussian weighting function with $\sigma$ equal to

[21] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 80.

[22] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 80.

one half the width of the descriptor window is used to assign a weight to the magnitude of each sample point."[23] (this is illustrated as the blue circle in figure 8).

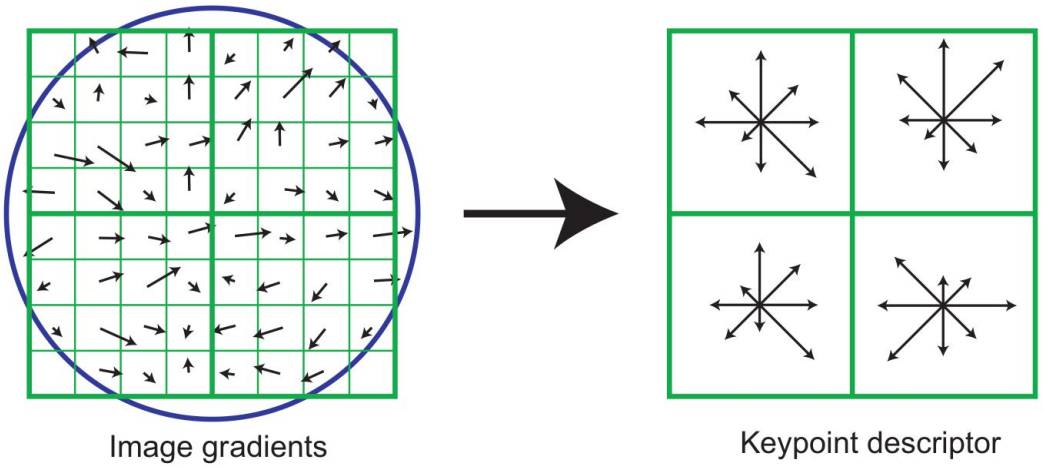


**Fig. 8. SIFT feature descriptor**. On the left side of the image, there is neighbourhood grid with image gradients weighted by Gaussian weighting function (blue circle) from which a grid of orientation histograms is formed (right part of the image). Instead of 16 × 16 grid and 16 orientation histogram, only 8 × 8 grid and 4 orientation histograms are depicted in the figure.

The 16 ×16 grid of image gradients is divided into 16 4 × 4 orientation histograms (right part of the figure 8). The histograms are formed by 8 bins. Each bin represents different orientation and its value represents that orientation magnitude. 16 orientation histogram, each with 8 orientation bins gives 128 element feature vector for each feature.

Finally, to account for change in illumination, the vector is normalized to unit length. "A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be cancelled by vector normalization."[24]

[23] LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. Volume 60. Issue 2. 2004. page 105.

[24] LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. Volume 60. Issue 2. 2004. page 106.

### 2.4.3 Feature matching

"The feature-matching step searches for corresponding features in other images."[25] In order to correctly match the features in two images we have to determine which of examined feature couples (one in the first and one in the second image) are to be marked as the correct match and which not, and we need to determine the way we search through feature space of images, i.e. define a data structure to store features.

#### 2.4.3.1 Correct match selection

Descriptors are compared using a similarity measure. If simple descriptors are used (based on local feature appearance), then the sum of squared distances or normalized cross-correlation is a good measure. For SIFT descriptors, the Euclidean distance is used ([11]). In case Euclidean distance is used for ranking potential matches, we need to set appropriate threshold (maximum distance) for accepting / rejecting match with the closest distance ([43]). "Setting the threshold too high results in too many false positives, i.e., incorrect matches being returned. Setting the threshold too low results in too many false negatives, i.e., too many correct matches being missed."[26] In [27], the author proposes a distance-ratio test that accepts the closest match in case the ratio between closest match and second closest match is smaller than the specified threshold. OpenCV Brute-Force matcher is using simpler strategy: "It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned."[27]

#### 2.4.3.2 Mutual consistency

In order to reject multiple matches, i.e., more features in first image are matched to one feature in the second image, a mutual consistency check can be used

[25] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 81.

[26] SZELISKI R., Computer Vision: Algorithms and Applications. Springer. 2010. page 201.

[27] From OpenCV 3.0.0-dev documentation. Available online on <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher>. Cited on <10.4.2018>

([43]). After matching all features in the first image, with all features in the second image the same search is performed in the opposite direction, i.e. all features in the second image are matched against all features in the first image. "Only pairs of corresponding features that mutually have each other as a preferred match are accepted as correct."[28] The mutual consistency check is also implemented in OpenCV library.[29]

#### 2.4.3.3 Constrained matching

The simplest way to compare all features is by using brute-force, i.e. all features in the first image are matched against all features in the second image. This approach is quadratic in the number of features. More efficient approach is to use an indexing structure, such as multi-dimensional search tree or a hash table, that allows for fast searching of features that are close to given feature ([43], [11]).

### 2.4.4 Feature tracking

Feature detection, feature description and consequent feature matching allows us to track the feature over time. However, exhaustive matching across images may be time consuming. In case of video sequences, where frame-to-frame motion is sufficiently small, it is expected that also changes in feature positions and appearance are small. This assumption gives us another approach to track features. First, we detect interesting features in one image and then we search for corresponding counterparts in following images at near positions assuming only a small appearance deformation. This detect-then-track approach is often called Kanade-Lucas-Tomasi tracker (klt tracker) ([11]). Shi-Tomasi detector is described in section 2.4.1.3. In following section, we provide an overview of Lucas-Kanade feature tracker ([28]) which was later refined by Tomasi and Kanade ([45]).

[28] FRAUNDORFER, F., et al. Visual odometry: Part II - Matching, robustness, optimization, and applications. IEEE Robotics and Automation Magazine. 2012. page 81.

[29] From OpenCV 3.0.0-dev documentation. Available online on <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher>. Cited on <10.4.2018>

### 2.4.4.1 Lucas - Kanade feature tracker

The Lucas-Kanade feature tracker ([28]) is a technique for estimating the movement of interesting features in a video sequence. As mentioned above, Lucas-Kanade feature tracker is based on assumption of small inter-frame changes. "Lucas-Kanade approach minimizes the sum of squared intensity differences between two consecutive windows. An underlying assumption is that given small inter-frame motion, the current window can be approximated by a translation of the past one. It is also assumed that the image intensities in the translated window can be written as those in the original window plus a residue term that depends almost linearly on the translation vector."[30]

In [28], the authors first derive solution for one dimensional image registration problem which they further generalize to multiple dimensions. The basic idea is described below.

Consider two Images $I$ and $I'$ (two consecutive frames) and an image point $x(x_x, x_y)$. The goal is to find a displacement vector $d$ that satisfies: $I(x) = I'(x + d)$. Let us denote gradient change in $x$ direction at point $x$ as $I_x(x)$, gradient change in the $y$ direction at point $x$ as $I_y(x)$ and temporal gradient change as $I_t(x, x + d)$ which represents a change in gradient between $I(x)$ and $I'(x + d)$. Assuming linear gradient change, the following equation holds:

$$I_x(x)\, d_x + I_y(x)\, d_y = -I_t(x, x + d). \tag{13}$$

The above equation has two unknowns $[d_x, d_y]^T$ and as such it is under-determined. According to [29], we can compute only the projection of the vector $d$ to the direction of image gradient. Assuming that the close neighbourhood of examined point has constant motion, we can compose a set of linear equations from equation (13). In perfect case such system has unique solution. However, due to noise, we may seek for solution that minimizes the following error function ([29]):

$$E(d) = \sum_{n(x)} \left(\nabla I^T(x)\, d + I_t(x, x + d)\right)^2, \tag{14}$$

[30] ANTIĆ, B., et al. Robust Detection and Tracking of Moving Objects in Traffic Video Surveillance. In: Advanced Concepts for Intelligent Vision Systems (ACIVS). 2009.Springer. Berlin. page 499.

where $n(x)$ represents the neighbourhood of point $x$ and $\nabla I = [\partial I/\partial x, \partial I/\partial y]^T$ is spatial gradient. In order to minimize the error, we compute the derivative of (14) with respect to $d$ ([28], [29]):

$$\nabla E(d) = 2 \sum_{n(x)} \nabla I \left( \nabla I^T d + I_t \right) = 2 \sum_{n(x)} \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} d + \begin{bmatrix} I_x I_t \\ I_y I_t \end{bmatrix}.$$

For $u$ that minimizes error function, we set $\nabla E(d) = 0$. We get:

$$\begin{bmatrix} \sum I_{xx} & \sum I_{xy} \\ \sum I_{xy} & \sum I_{yy} \end{bmatrix} d + \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} = 0,$$

which can be rewritten as $Ad + b = 0$ or $u = -A^{-1}b$. In case $A^{-1}$ exists, we have a unique solution that minimizes (14). In case $A^{-1}$ does not exist it means that either the gradient is changed only along one direction (cases where $I_x = 0$ or $I_x = 0$ - line) or gradient is not changed at all ($I_x = 0$ and $I_y = 0$ - textureless region) ([29]). For details about correct features to track, see section 2.4.1.

# 2.5 Dealing with noise

Even in the case of perfect odometry and a very good feature tracker, it is (most likely) impossible to obtain noise free measurements. There are several methods addressing estimation of a random variable. In following section, we describe one of the most popular filtering method - Kalman filter.

## 2.5.1 Kalman filter

Kalman filter ([23]) is a filtering method used to estimate true value of a random variable. It is widely used in many different technical fields. Its main advantage is low computational cost and recursivity, i.e. it is using previous state of the system to estimate the current one ([7]). The following equations are according to [7] and [23].

Each system that can be described by following equations, can be modelled with the Kalman Filter:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1},$$
$$z_k = Hx_k + v_k,$$

where each subscript denotes the timestamp, $x_k$ is the current state vector. $A$ is the state transition model, $B$ is the control model, $H$ is the measurement model, $u_k$ is the control signal at the timestep k. $z_k$ is the observation vector and finally $w_{k-1}$ denotes the process noise at the timestep k - 1 and $v_k$ measurement noise at the timestep k.

There are two important assumptions: 1. The Kalman filter is expecting linear process with the 2. zero mean Gaussian noise.

The algorithm is composed of two steps: 1. prediction step and 2. measurement update step. Equations for prediction step are as follows:

$$\hat{x}_{k|k-1} = A_k \hat{x}_{k-1|k-1} + B_k u_k,$$
$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k,$$

where $Q_k$ is the process noise covariance matrix and $\hat{x}_{k|k-1}$ stands for estimated state of current timestep $k$ from previous state at timestep $k-1$.

The measurement update step equations are given by

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - H_k\hat{x}_{k|k-1}),$$
$$P_{k|k} = P_{k|k-1} - K_k H_k P_{k|k-1},$$

where

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_t^T + R_k)^{-1}$$

is called Kalman gain and $R_k$ is the measurement noise covariance matrix.

Each time a new measurement is observed, it is used to update the current state vector which is consequently used for the prediction step.

For a detailed derivation, we refer reader to the original paper ([23]) or to [7], where intuitive derivation can be found. There are also modifications to the original Kalman filter algorithm that allows filter to be used also for non-linear processes like Extended Kalman Filter ([14]) or Unscented Kalman Filter ([22]).

# 3 Proposed solution

As discussed in the previous section, in order to provide an accurate depth estimation we need the correctly tracked features and an accurate pose estimation of a moving robot. To reach modularity of our system, we have decoupled the depth estimation from the pose estimation.

## 3.1 Middleware platform

We aim to propose a modular system for depth estimation. As modularity is the key requirement, we firstly describe a modular framework suitable for our purposes. As the modularity is a key requirement of the application, we have paid close attention to choosing the appropriate robot middleware platform. Desirable requirements for the platform are:

- Open-source
- Robot software independent architecture
- Real time operation
- Distributed environment

Based on [30] and more recent survey [46], we have come across to following platforms that meet above requirements:

- ROS[31]
- Orocos[32]
- OPRoS
- YARP[33]

All of the listed frameworks meet our requirements. However, there are couple of more requirements that one may consider:

- Maintenance
- Community
- Hardware interfaces and drivers

[31] More about ROS available online on <http://www.ros.org/>. Accessed <23.3.2018>
[32] More about Orocos available online on <http://www.orocos.org/>. Accessed <23.3.2018>
[33] More about YARP available online on <http://www.yarp.it/>. Accessed <23.3.2018>

The most maintained and most widely used framework is ROS ([46], [30]). Currently, there are 11 distributions of ROS[34] and the community is still growing. In addition, many mentioned computer vision algorithms are already developed and wrapped in ready-to-use packages in the ROS main repository (e.g. openCV applications, visual and other odometry algorithms like SVO ([9]), PTAM ([24]), and many others).

### 3.1.1 Ros

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms."[35]

### 3.1.2 Computational graph

The ROS building block is represented by a node - it is a computational unit that communicates with other nodes via messages and services. Ros computational graph "is the peer-to-peer network of ROS processes that are processing data together."[36] An overview of simplified ROS computational graph is provided in figure 9.

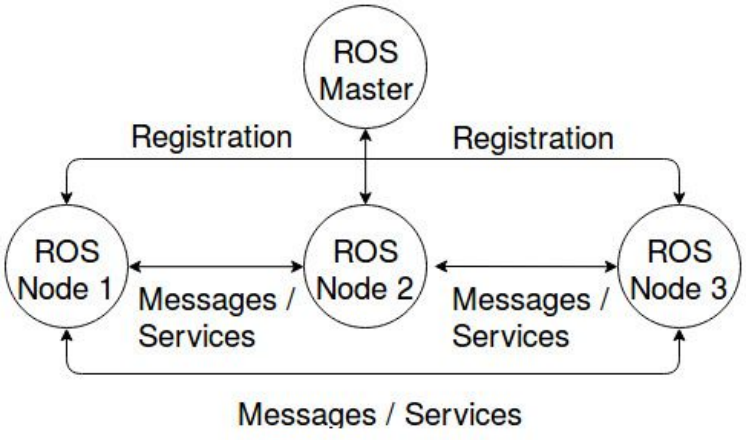


**Fig. 9. ROS nodes registration and communication flow.**

[34] <http://wiki.ros.org/Distributions>. Accessed <23.3.2018>
[35] Available online on <http://www.ros.org/about-ros/>. Accessed <23.3.2018>
[36] <http://wiki.ros.org/ROS/Concepts>. Accessed <23.3.2018>

### 3.1.3 Nodes

"Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, one Node provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as roscpp or rospy."[37]

### 3.1.4 Master

"The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services."[38]

### 3.1.5 Messages

"Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs)."[39]

### 3.1.6 Topics

"Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's' existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly

[37] Available online on < http://wiki.ros.org/ROS/Concepts>. Accessed <24.3.2018>
[38] Ibid.
[39] Ibid.

typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type."[40]

# 3.2 System architecture

The system is decoupled into 3 main parts (figure 10):

- Odometry
- Feature tracking
- Depth estimation

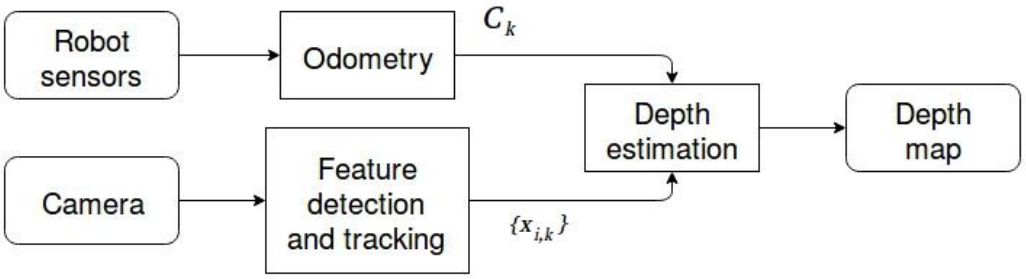


**Fig. 10. System overview.** At each timestep $k$ we receive camera pose $C_k$ and set of tracked features $\{x_{i,k}\}$ of which the depth is being estimated.

## 3.2.1 Odometry

The odometry module is to be chosen by the user. An important part is that the module must register to the Master node and publish a global pose on specific topic with specified message.

For evaluation purposes, we use SVO which is detailed in section 2.2.2.1. However, depending on the robot platform, different odometry modules and models can be used. We have chosen SVO as it is one of the state-of-the-art visual odometry algorithm that is low-cost (in the matter of time consumption), open-source, free and ready to use system that is suitable for our application.

[40] Available online on < http://wiki.ros.org/ROS/Concepts>. Accessed <24.3.2018>

### 3.2.2 Feature detection, matching and tracking

The feature detection, matching and tracking module is to be chosen by user. Important part is that the module registers to the Master node and publishes the array of tracked features on a specific topic with a specific message.

Each feature must have id and "age" - the number of preceding frames in which the feature was successfully tracked. When the feature is lost, it is indicated by negative age which means that the feature is no longer tracked and that its id is free to be used again. In such way, we know in how many consecutive frames and under what id the feature was previously tracked. That allows us to match features between any two frames in which the feature was tracked.

For evaluation purposes, we use Shi-Tomasi good features to track for feature detection and for consequent tracking of the detected features, the Lucas-Kanade feature tracker is used. Both methods are computationally fast and sufficiently accurate. Both methods are part of openCV library.

### 3.2.3 Depth estimation

The depth estimation module is simultaneously receiving data from the previously described modules: at each time step we receive a camera pose $C_k$ and a set of tracked features $\{x_{i,k}\}$.

We use $p$ time steps as a distance between the two frames in order to gain appropriate disparity between two (figure 11). From the set of two camera poses, $C_{k-p}$ and $C_k$ at different time steps $k-p$ and $k$ observing the same scene with the corresponding sets of tracked features $\{x_{i,k-p}\}$ and $\{x_{i,k}\}$ the set of 3-space points $\{X_i\}$ is reconstructed using triangulation. The reconstructed 3-space points are then used to estimate the depth of features in the second image $\{x_{i,k}\}$ as described in following sections (overview of Depth estimation process can be seen in figure 12).

For triangulation, we use Direct Linear Transformation (DLT) method described in section 2.3.1. The method is computationally fast, and it is part of the openCV library[41].

[41] <https://docs.opencv.org/3.1.0/d0/dbd/group__triangulation.html>. Accessed <2.4.2018>

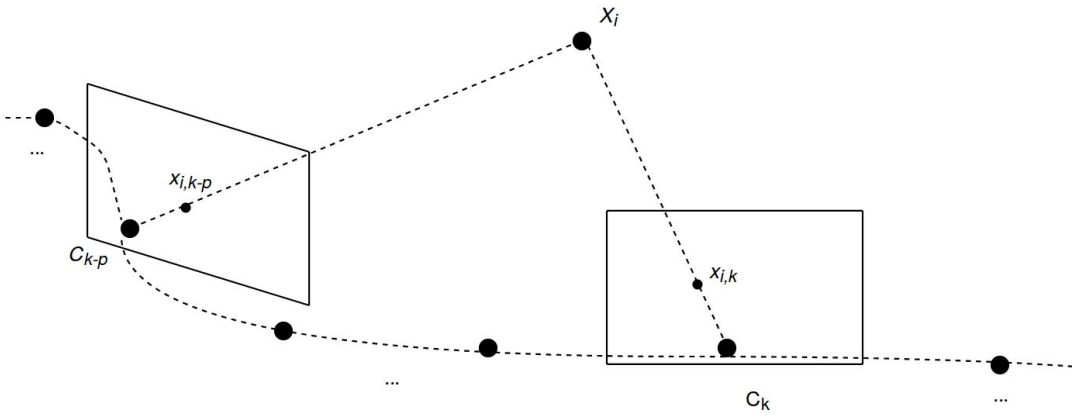


**Fig. 11. Triangulation frame disparity**. The depth of each feature is estimated from 2 different images taken at timestep $k-p$ and $k$.

After triangulation the points positions are optimised via Position estimator (see following section). For each point we also initiate a Position filter (see section 3.2.3.2).

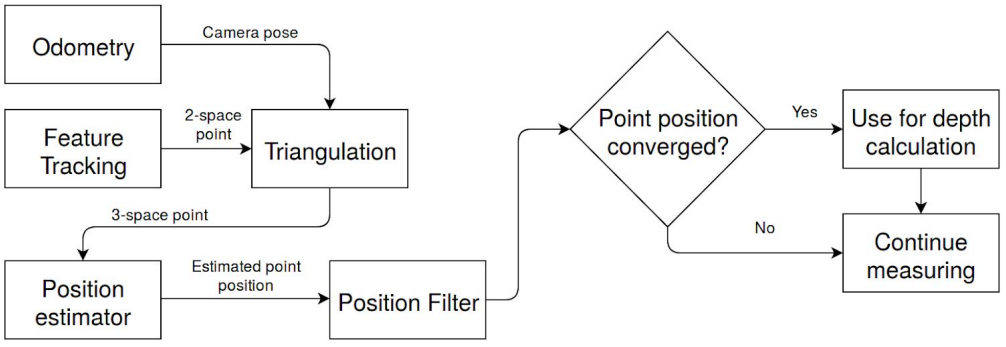


**Fig. 12. Depth estimation.** Triangulated point is first refined by the Position estimator, and then, in case its position has converged it is used for depth calculation.

### 3.2.3.1 Points position estimation

Due to possible noise in the feature tracking and the projectivity problem of linear triangulation[42], we expect positions of triangulated 3-space points to be noisy as well. To account for these inaccuracies, we employ 3 different methods for the random variable value estimation.

[42] See section 3.3.

#### 3.2.3.1.1 Median estimator

We track each generated 3-space point in time. In such way we get a set of positions of tracked point and we return the median value of this set. The problem of such an approach is that over time it accumulates too many values and at certain point the system stops operating in real time which is caused by the complexity of processing the continuously increasing amount of data. For these reasons, we have employed also the following method.

#### 3.2.3.1.2 Sliding window Median estimator

Sliding window Median estimator is based on the previous method, the only difference is the size of the generated set. In order to maintain this size, the oldest values are removed. Another advantage of this method is that it accounts for a possible drift in pose that is accumulated over time by odometry. In such case, the oldest values in the set are with each new frame becoming more and more inaccurate. The proposed estimator is returning the median value of the most recent values of the measured point pose.

#### 3.2.3.1.3 Kalman filter estimator

We have also implemented Kalman filter for point pose estimation[43]. The state vector is simply $x$, $y$ and $z$ coordinate of a point. We have set the process noise covariance matrix to 0 since the observed points are static and empirically estimated the value of measurement noise. The particular value of measurement noise is directly dependent on the odometry units. Each filter is initialized with the first measured value.

### 3.2.3.2 Points position filtering

In order to remove the incorrectly estimated points, we are coupling pose estimator with one of position filters. The filter is not only removing incorrectly reconstructed points, but also increasing the accuracy of correct ones.

[43] cv::KalmanFilter class - part of OpenCV library. For details see <https://docs.opencv.org/3.4/dd/d6a/classcv_1_1KalmanFilter.html>. Accessed <1.5.2018>.

#### 3.2.3.2.1 Pose filter

Pose filter is iteratively measuring the variance of point position. When a new point is detected, it is not directly used for the depth calculation, but first, variance of its pose is measured and when it becomes sufficiently small, the point is used and its depth is calculated. For iterative calculation of variance, we use the following formula ([8]):

$$S_n = S_{n-1} + (x_n - \mu_{n-1})(x_n - \mu_n),$$

$$\sigma_n = \sqrt{S_n/n},$$

where $x_n$ is the position of the tracked feature in n-th frame and $\mu_n$ represents the mean of all positions of the tracked feature.

#### 3.2.3.2.2 Pose change filter

We are also proposing a pose change filter that in general is the same as pose filter, the only difference is that it is not filtering features position but the change in position.

## 3.3 Possible problems

Below we list some of the triangulation and feature tracking related problems. However, we believe that by using a point pose estimator coupled with position filter, we will be able to overcome these problems.

### 3.3.1 Projectivity

As discussed in section 2.3.1, the homogeneous DLT is neither affine invariant nor projective invariant. Therefore, in case of highly projective transformations (figure 13), the DLT may result in incorrectly reconstructed 3D points. This problem can be minimized also by using stereo rectification (see section 2.3.2).

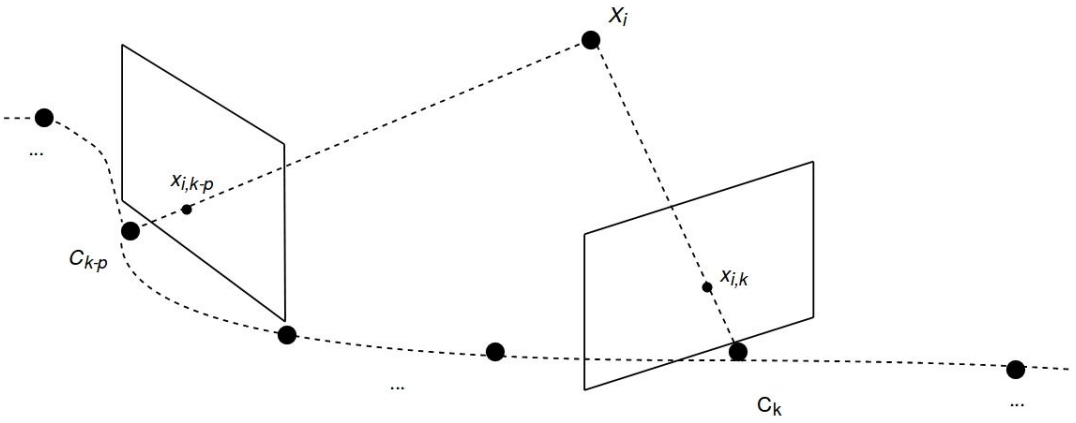


**Fig. 13. Projective transformation.** Linear triangulation provides best results when used image planes are parallel to each other, i.e. there is no projective transformation. In shown image the angle between two camera poses is significant.

## 3.3.2 Chirality

"Another problem to watch out for with triangulation is the issue of chirality, i.e., ensuring that the reconstructed points lie in front of all the cameras."[44] In the presence of noise in image points measurements and / or pose estimation it is possible that some reconstructed points appear behind the cameras (see figure 14). This scenario is mostly probable for points at infinity as they appear parallel in stereo images or with small disparity as the parallax is not sufficient and even with a small amount of noise it is probable that the rays will diverge. "A useful heuristic is to take the points that lie behind the cameras because their rays are diverging and to place them on the plane at infinity by setting their W values to 0."[45] More details about Chirality can be found in [16].

[44] Szeliski R., *Computer Vision: Algorithms and Applications*. Springer. 2010. page 307.
[45] Szeliski R., *Computer Vision: Algorithms and Applications*. Springer. 2010. page 307.

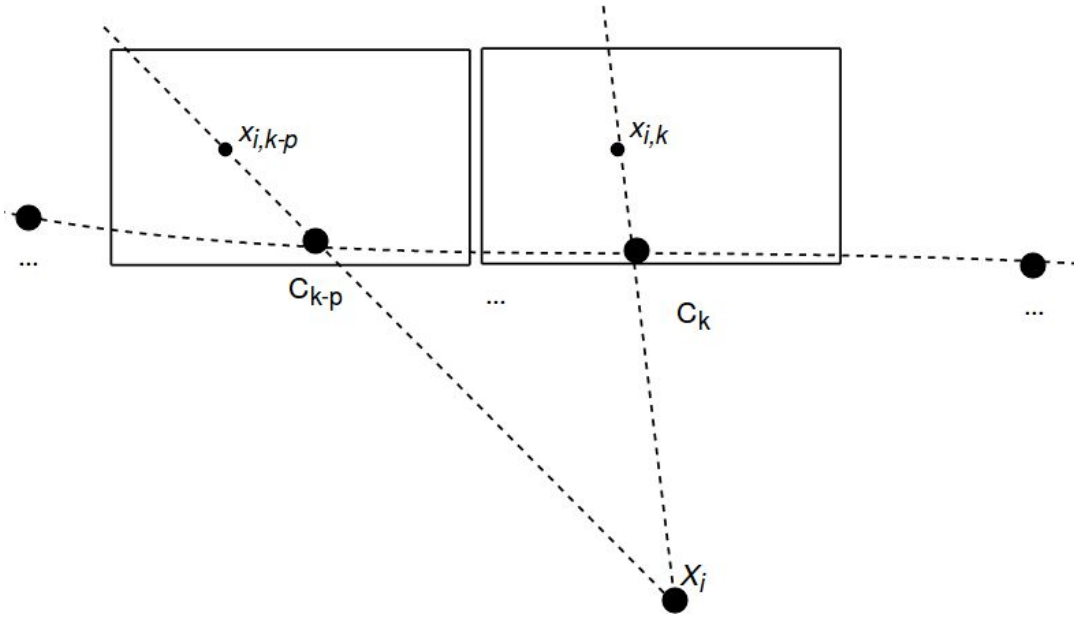


**Fig. 14. Chirality.** In presence of noise and / or small frame disparity (distance between $C_{k-p}$ and $C_k$), it is possible for 3-space point $X$ to be wrongly reconstructed in the area behind cameras.

### 3.3.3 Tracking drift and occlusions

Since in feature detection and tracking step we use feature tracking instead of feature matching (i.e. we detect features once and then track them from frame to frame instead of detecting and matching features in each new frame), in certain cases some features that are tracked for a long period of time may exhibit more drift that results in incorrect tracking. The same holds for occlusions when occluded feature may drift as well.

# 4 Implementation

In this section, we discuss the implementation details of the system proposed in the previous section. At the end of the section, we provide user manual for usage purposes.

The project was developed on Ubuntu 16.04 with ROS Kinetic Kame.

## 4.1 Nodes

ROS Package name: “depthest” - all executables are stored within this package.

Nodes:

- depthest - depth estimation node as described in section 3.2.3.
- visualizer - node used for various visualisations
- remapper - supporting node for remapping the topics and images

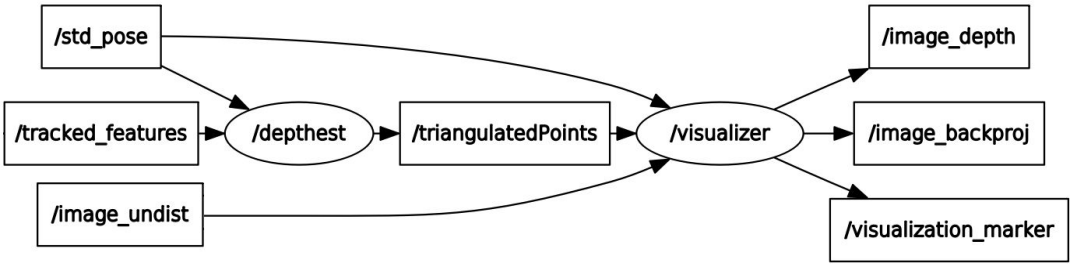


**Fig. 15. ROS topics graph of depthest node and visualizer node.**

### 4.1.1 Depth estimation node

Main node used to reconstruct the 3-space points from the camera poses and corresponding tracked features. To each reconstructed 3-space point, a position estimator and position filter are assigned. Optimised and filtered 3-space points are then used to estimate the depth of each tracked feature. The method works as described in the previous section (proposed solution).

Subscribed topics:

- std_pose (geometry_msgs/PoseStamped) - global position of the camera with timestamp.
- tracked_features (depthest/AgedFeaturesStamped) - array of tracked features (2-space points) with timestamp.

Published topics:

- triangulated_points (depthest/PointsStamped) - array of reconstructed 3-space points.

### 4.1.2 Visualizer node

Node used for 2D and 3D visualizations: 3D visualizations of the reconstructed scene (point cloud and camera path) and 2D visualization of reprojection of reconstructed 3-space points. In such way, the reprojection error is visible on the image plane.

Subscribed topics:

- image_undist (sensor_msgs/Image) - undistorted image from camera feed used for 2D visualization of calculated depth and difference between the originally tracked and reprojected points (see figure 16).
- std_pose (geometry_msgs/PoseStamped) - global camera pose used for 3D visualizations of the camera path and orientation (figure 16).
- tracked_features (depthest/AgedFeaturesStamped) - array of tracked features used for 2D visualization of the originally tracked features (green colour).
- triangulated_points (depthest/PointsStamped) - array of reconstructed 3-space points used for reprojection on image plane in order to visualize the reprojection error between originally tracked features and reprojected 3-space points (blue colour). These points are also used for 3D visualization (see figure 17).

Published topics:

- image_backproj (sensor_msgs/Image) - undistorted image from camera feed with visualized originally tracked features in green colour and the reprojection of reconstructed 3-space points in blue colour.

- image_depth (sensor_msg/Image) - undistorted image from camera feed with visualized depth of tracked features.
- visualization_marker (visualization_msgs/Marker) - marker used in Rviz for 3D visualization of the camera path and orientation.

**Fig. 16. 2D visualizations.** On the left side, there is a visualization of original (green) and reprojected (blue) points. In ideal case of zero reprojection error, all points would be blue. On the right side, the depth of tracked features is visualized. Red colour is used for points that are close to the camera and blue for the distant ones.

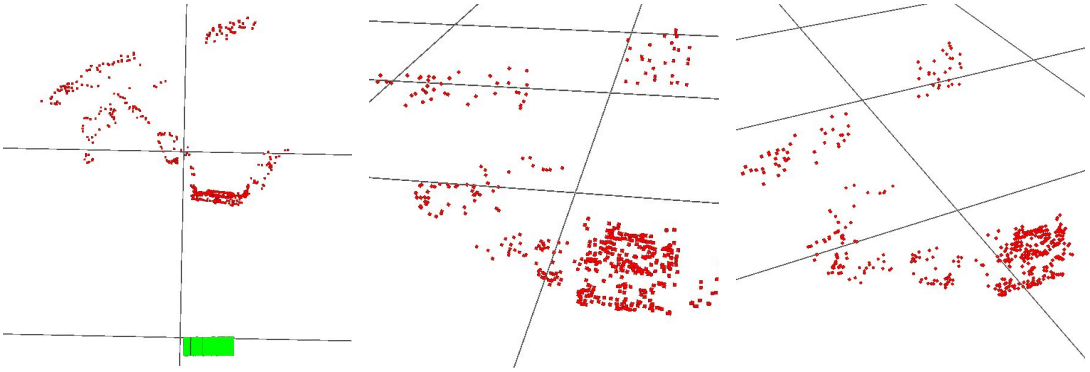

**Fig. 17. 3D visualizations.** Visualizations of generated point cloud (red colour). On the left, there is a top view of model with the illustrated camera path (green lines that, when being close to each other, form a rectangular shape in case of pure lateral motion) and remaining two are perspective views from different angles.

### 4.1.3 Remapper node

A supporting node used to remap original image to its undistorted version necessary for feature tracking and visualizations.

Subscribed topics:

- image_raw (sensor_msgs/Image) - raw image from the camera feed.

Published topics:

- image_undist (sensor_msgs/Image) - undistorted image for further usage.

### 4.1.4 Odometry and feature detection, matching and tracking

For evaluation purposes, we have used 3rd party softwares for visual odometry and feature tracking that are necessary for depth estimation.

#### 4.1.4.1 Odometry

As mentioned in section 3.2.1, the odometry module is to be chosen by the user. For our purposes, we have implemented SVO_ROS package[46] and modified its published topics as described in section 4.1.1. Only very minor changes were necessary (changing topic name).

#### 4.1.4.2 Feature detection, matching and tracking

Feature detection, matching and tracking module is to be chosen by the user. For our purposes, we have implemented opencv_apps package[47]. We have modified whole lk_flow application (class lk_flow_nodelet) in order to publish tracked features as described in section 3.2.2 and 4.1.1.

## 4.2 Installation

In order to run the depthest package, there are couple of dependencies needed:

- OpenCV[48],
- ROS[49],

[46] Source code available online on <https://github.com/uzh-rpg/rpg_svo>. Downloaded on <17.10.2017>. For further details about installation, we refer reader to <https://github.com/uzh-rpg/rpg_svo/wiki>. Accessed <25.3.2018>.

[47] Source code available online on <https://github.com/ros-perception/opencv_apps>. Downloaded on <1.10.2017>. For further detail about installation and usage, we refer reader to <http://wiki.ros.org/opencv_apps>. Accessed <25.3.2018>

[48] For installation instructions visit <https://opencv.org/>. Accessed <26.3.2018>

[49] For installation instructions visit <http://wiki.ros.org/ROS/Installation> Accessed <24.3.2018>. We have developed our system with ROS kinetic kame.

- Eigen[50],

For installation, a workspace for the ROS-Catkin projects (catkin_ws) is necessary. Then, to compile depthest package:

cd catkin_ws/src

git clone https://github.com/brzzda/depthest.git

catkin_make

# 4.3 Running the system

Below we provide an overview of the system settings and parameters that are necessary to run the system.

## 4.3.1 Camera calibration and system start-up

For proper depth estimation, a calibrated camera is necessary. It is Important to store camera calibration data in .yaml format (see figure 18). In order to run depthest, please follow instructions below:

1. Calibrate camera and store calibration file in
   <PATH_TO depthest PACKAGE>
   /params/your_calibration_file.yaml
2. Adapt launch file in
   <PATH_TO depthest PACKAGE>/launch/depthest.launch
3. Run depthest:
   roslaunch depthest depthest.launch

In order to visualize 3D reconstructed points, run rviz configuration file:

rosrun rviz rviz -d <PATH_TO depthest PACKAGE>/rviz_config.rviz

[50] For installation instructions please visit <http://eigen.tuxfamily.org/index.php?title=Main_Page>. Available on <20.4.2018>

```
cam_model: Pinhole
cam_width: 752
cam_height: 480
cam_fx: 414.536145
cam_fy: 414.284429
cam_cx: 348.804988
cam_cy: 240.076451
cam_d0: -0.283076
cam_d1: 0.066674
cam_d2: 0.000896
cam_d3: 0.000778
```

**Fig. 18. Camera calibration file example.**

It is assumed that the odometry module is publishing camera pose on topic "std_pose" (geometry_msgs/PoseStamped) and that the feature tracking module is publishing features on topic "tracked_features" (depthest/AgedFeaturesStamped).

## 4.3.2 Parameters

The program has several parameters:

- "calibration_file" - name of the camera calibration file.
- "estimator" - setting the estimator ("dummy", "median", "median_window", "kalman"),
- "kf_meas" - measurement covariance for the Kalman filter pose estimator,
- "kf_proc" - process covariance for the Kalman filter,[51]
- "triang_window" - size of the window between the two frames that are used for triangulation,
- "filter" - setting the filter type ("none", "pose_filter", "pose_change_filter"),
- "variance_thresh" - variance threshold used for the pose or pose change filter (depends on what filter type is chosen),
- "enable_measurements" - if true, the program calculates and shows various measurements on the screen.

All parameters can be changed either in launch file or directly in console.

[51] To be exact, both values (kf_proc and kf_meas) represents variance of, x, y and z coordinates (not covariance) - both values (kf_meas and kf_proc) are used to create identity matrix for measurement and process noise covariance matrix respectively.

### 4.3.3 Launch file

Provided launch file “depthest.launch” launches 3 nodes at the same time:

- “depthest” node for calculations
- “visualizer” node for 3D, reprojection and depth visualizations
- “remaper” node for remapping the raw image to undistorted image for feature tracking purposes.

# 5 Evaluation

In this section, we provide an evaluation of our system that was tested in real-world scenarios.

## 5.1 Setup

The project was developed with C++ programming language using CLion IDE (student subscription). gcc version: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609. On Intel® Core™ i5-5200U CPU @ 2.20GHz × 4 machine with 7,7 GiB memory, 64-bit OS type. Using ROS Kinetic Kame distribution (roscpp 1.12.12).

The footage was obtained by Parrot AR.drone 2.0 with 640x320 QVGA 30fps pinhole camera as it represents a typical onboard camera on drone-like platforms.

## 5.2 Measurements

We are proposing a technique for measuring the relative depth estimation precision based on trade-off between the reprojection error and the stability of the reconstructed 3-space model (point cloud). Considering static scene and perfect scenario of measurements, one would get zero reprojection error and a static pointcloud of triangulated points (we are expecting static scene). In presence of noise, however, there is expected a trade-off between the reprojection error and the point cloud stability. Therefore, we evaluate our system mostly by monitoring these two values.

### 5.2.1 Reprojection error

In [19], the authors states following reprojection error calculation:

$$RE' = \sum_i d(x_i, \hat{x}_i)^2 + d(x_i', \hat{x}_i')^2 \quad ,$$

where $x_i$ and $x_i'$ are image points of the first and second image plane respectively. These points are used for triangulation from which we get $\hat{x}_i$ and $\hat{x}_i'$

that are reprojected points of the first and second image plane respectively, $d$ represents euclidean distance and the summation goes over all points that were detected in both images.

Since we are focused on depth estimation of the actual frame, we have chosen to measure only the Euclidean distance of the second (actual) image plane and since we are not minimizing this error, but only monitoring it, we do not calculate its square value:

$$RE = \sum_i d(x_i', \hat{x}_i') \ . \tag{15}$$

We are measuring the shift in pixels.

### 5.2.2 Model stability error

We say that the model is stable when there is no movement of generated point cloud between the two consecutive time steps. Let's denote $\Delta X$ the change in position of a 3-space point $X$ between two consecutive time steps. Then the model stability error of one time step is:

$$ME = \sum_i \Delta X_i , \tag{16}$$

where the summation goes over all detected features in one time step. Units of the measured model stability error depend solely on odometry. In case odometry module is able to measure distances in meters, then the unit of ME is a meter. For our purposes, we will denote it simply as units.

### 5.2.3 Rejected points ratio

As we are using filtering methods for increasing precision and outlier removal, it is worth monitoring the ratio between all and rejected[52] points. If we set the filter too "tight", it will allow only a small amount of points to be used for depth estimation, resulting in strong model stability but weak performance as the depth map will be weakly populated. The ratio is calculated as follows:

[52] The ones that did not passed through pose filter.

$$RPR = \frac{\sum_r X_r}{\sum_i X_i}, \tag{17}$$

where $\{X_i\}$ represents the set of all detected points and $\{X_r\}$ represents the set of rejected points by the filter.

# 5.3 Experiments and results

The the following experiments demonstrate the system behaviour in 3 typical situations:

- Lateral motion - the pure translational motion to the side which represents the least noisy measurements.
- Rotation - camera rotating to the side - the noisiest measurements.
- Handheld camera motion - arbitrary handheld camera motion - combination of the former two motions.

All tests depict the same common indoor scene (figure 19) which we have chosen in order to show the behaviour of the system in real-world scenarios. We have executed several tests for each type of motion[53]. Each test consists of 12 execution of the system - one per each Position estimator (Dummy, Median, Median Sliding Window and Kalman) and Position filter (None,[54] Pose filter and Pose Change filter).

Below we provide the results of the test runs that were selected to be suitable for evaluation[55]. For comparison reasons we have tried to reach similar $RPR$ (rejected points ratio (17)) for all Position estimators and Position filters (except Dummy estimator). The RPR was chosen so that the results provide enough tracked points with as few outliers as possible (interesting parts to notice are depicted on figure 20). In the following sections we provide an overview of the performance of each estimator and filter for each motion scenario.

[53] All tests measurements can be found it attachment A.2 (140+ test runs). Selected tests measurements used for comparison can be found in attachment A1.

[54] Dummy estimator is doing same thing as none point filter - nothing. We name them differently for easier recognition.

[55] .bag files of selected test runs can be found in attachment A.4.


**Fig. 19. Experiments scene.** The scene used for evaluation. The scene consists of a distinguishable distant object (painting on the wall), a distinguishable object close to camera (big box in the middle), and challenging occluding parts just next to the box (where outliers are expected to be tracked)

**Fig. 20. Interesting regions of the scene.** The critical area is represented by points located around edges of the box (blue ellipses in the left picture). Some of these points get occluded throughout the process, and some are detected at intersections of the box and the blue pillow. When camera moves, the detected point at intersection no longer exists, and in case it is redetected, it is inevitably at shifted position which is resulting in ray of 3-space points creating a non existing "wall" between the box and the pillow (blue circles in the right picture - top view of generated 3-space point cloud). The green line represents a real wall. One easy distinguishable cluster of points is represented by painting on the wall and another one is the edge of the couch (green circles). In this case, both correctly reconstructed.

| Lateral motion | Rotation | Handheld motion |
|---|---|---|
| 0,027 | 0,006 | 0,038 |

**Table 1. Average frame disparity.** Average frame disparity of selected test runs used for triangulation. Measured in odometry units.

### 5.3.1 Lateral Motion

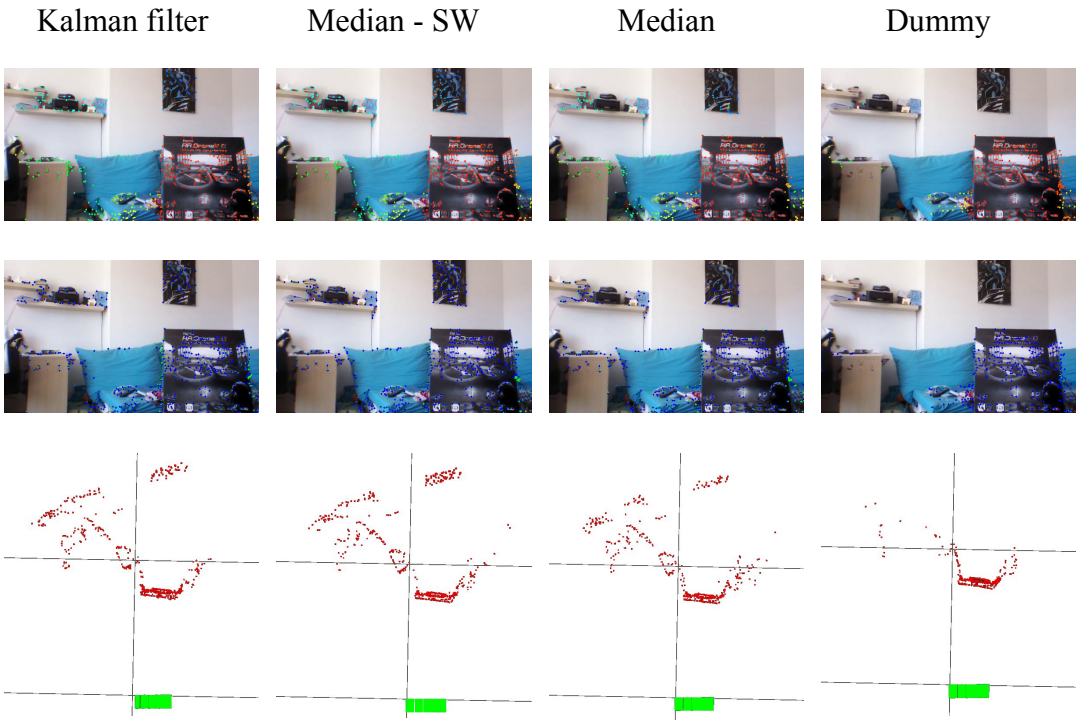


**Fig. 21. Lateral motion - Position estimators performance**. Depth map (first row), reprojected points (second row) and generated point cloud (third row, top view). Kalman filter estimator coupled with Pose filter. Median SW estimator with Pose filter. Median estimator with Pose filter. Dummy estimator with Pose filter.

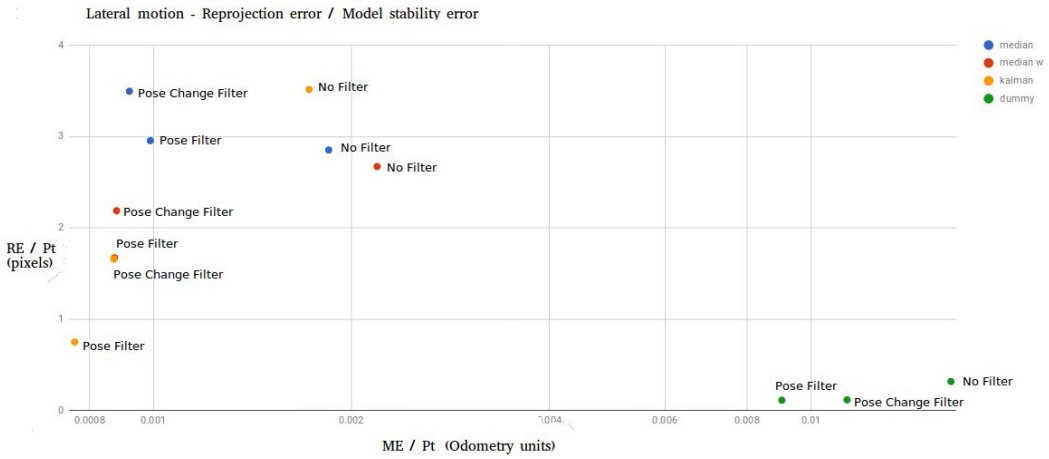


**Fig. 22. Lateral motion - overall performance.** Graph showing performance of each Position estimator coupled with each Position filter.

As can be seen in figure 22, the best results are given by Kalman Filter pose estimator with the Pose filter for outliers' removal. With lowest ME (model stability error (16)) it is the most stable estimator with the lowest RE (reprojection error (15)). In table 2, we show the results of estimators in terms of RPR and average depth. Kalman filter estimator with Pose filter has the lowest RPR and highest Average depth (measured in odometry units).

| Estimator | Kalman filter | Median - SW | Median | Dummy |
|---|---|---|---|---|
| RPR | 0,271 | 0,304 | 0,282 | 0,352 |
| Average depth | 1.0183 | 1.0301 | 1.076 | 0.97537 |

**Table 2. Lateral motion RPR and Average depth.**

## 5.3.2 Rotation

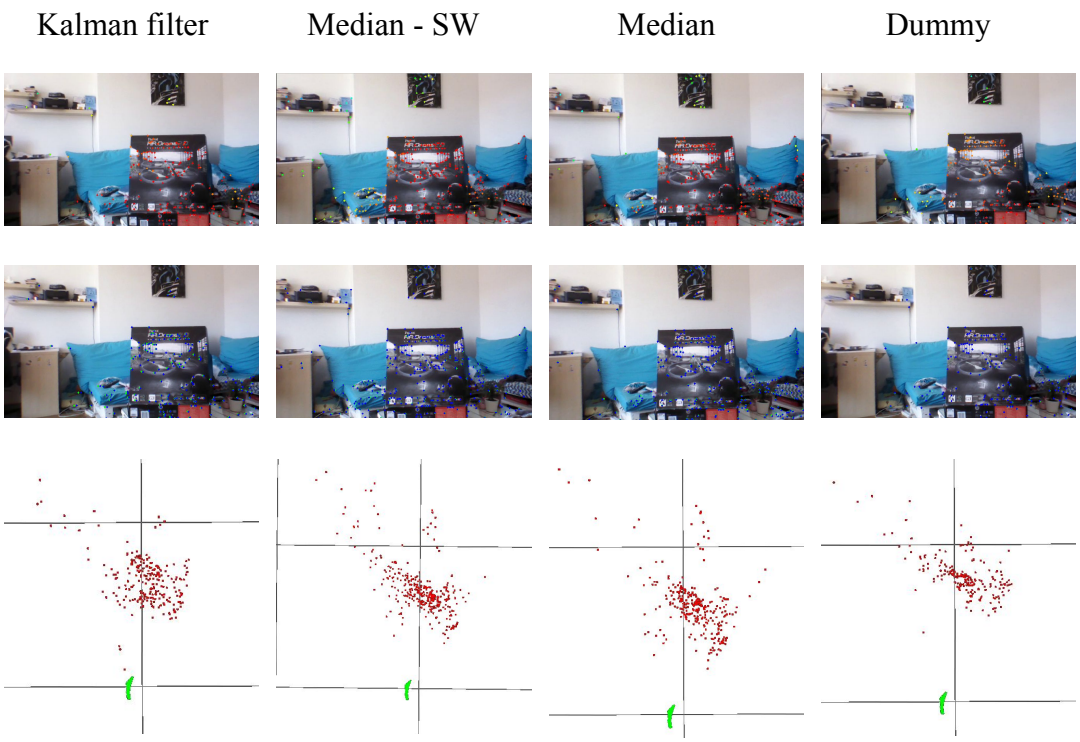


**Fig. 23. Rotation - Position estimators performance.** All pose estimators coupled with Pose filter.

As expected, all estimators are providing insufficient results (notice the generated point cloud in figure 23). The reason is only a very small frame disparity used for triangulation (see table 1) and very high projectivity (see section 3.3.1) that is coupled with rotation. RPR is high (approx 50% of points are rejected by Position filter) and average depth is low as well (77% of best case average depth: Kalman filter estimator - lateral motion scenario). That means that pose estimators are optimising very noisy measurements and only points close to the camera are passing through the filter and even after filtering the generated point cloud is still noisy. ME of points that pass through the filter is approximately 10 times higher comparing to lateral motion[56]. High RPR, low depth, small frame disparity and high ME are indicating that the reconstructed point cloud is inaccurate.

[56] See attachment A1 for all measured values.

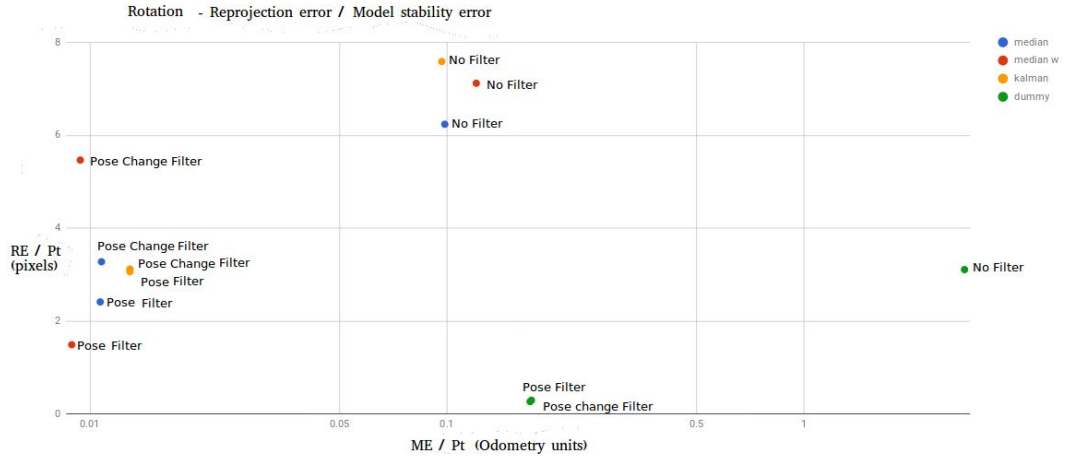


**Fig. 24. Rotation - overall performance.** Performance of each Position estimator coupled with each Position filter.

| Estimator | Kalman filter | Median - SW | Median | Dummy |
|---|---|---|---|---|
| RPR | 0,576 | 0,568 | 0,563 | 0,76 |
| Average depth | 0.7322 | 0.7813 | 0.77499 | 0.7862 |

**Table 3. Rotation RPR and Average depth.**

### 5.3.3 Handheld camera motion

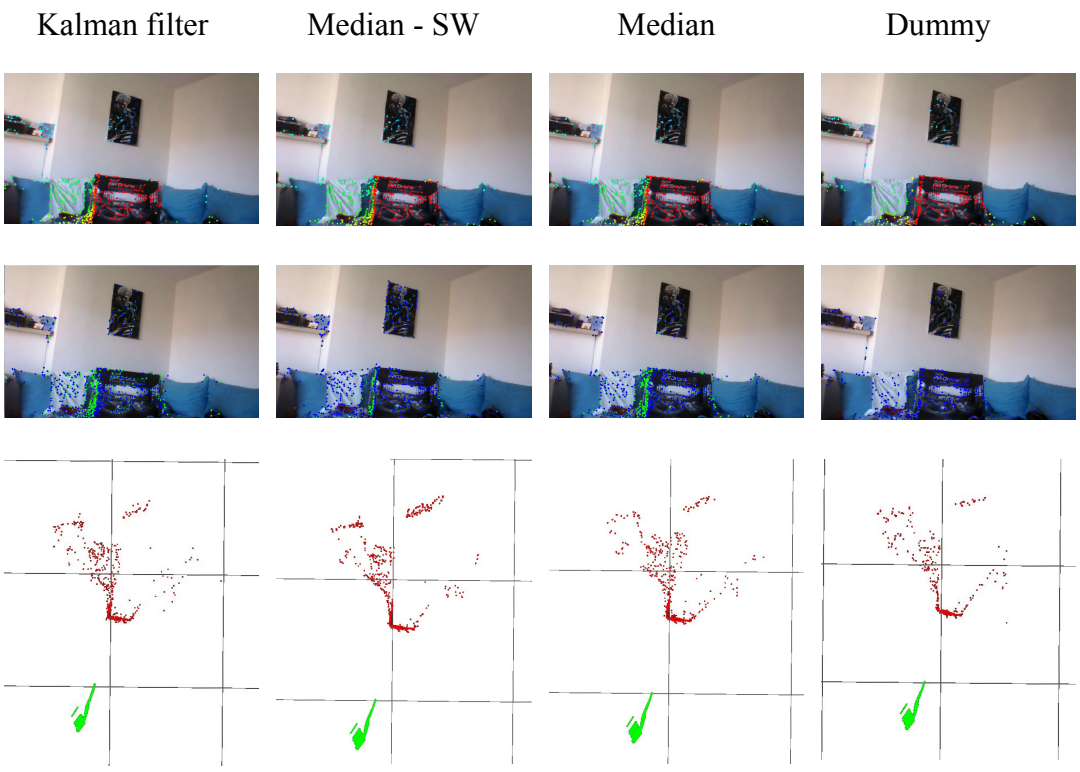


**Fig. 25. Handheld camera motion - Position estimators performance.** Kalman filter estimator with a Pose filter. Median SW estimator with a Pose change filter. Median estimator with a Pose change filter. Dummy estimator with a Pose filter.

In handheld scenario the best results in terms of model stability are represented by Kalman filter estimator and Median estimator with Pose change filter. However, the average reprojection error of both is over 13 pixels per point (see figure 27) what is indicating that the generated point cloud is not accurate. Sacrifying model stability for a lower reprojection error is resulting in point cloud with fewer outliers (see figure 26). Considering RE / ME trade-off the best results are provided by Median SW estimator with the Pose change filter. The Pose change filter outperformed Pose filter in all aspects - RPR, ME and RE.

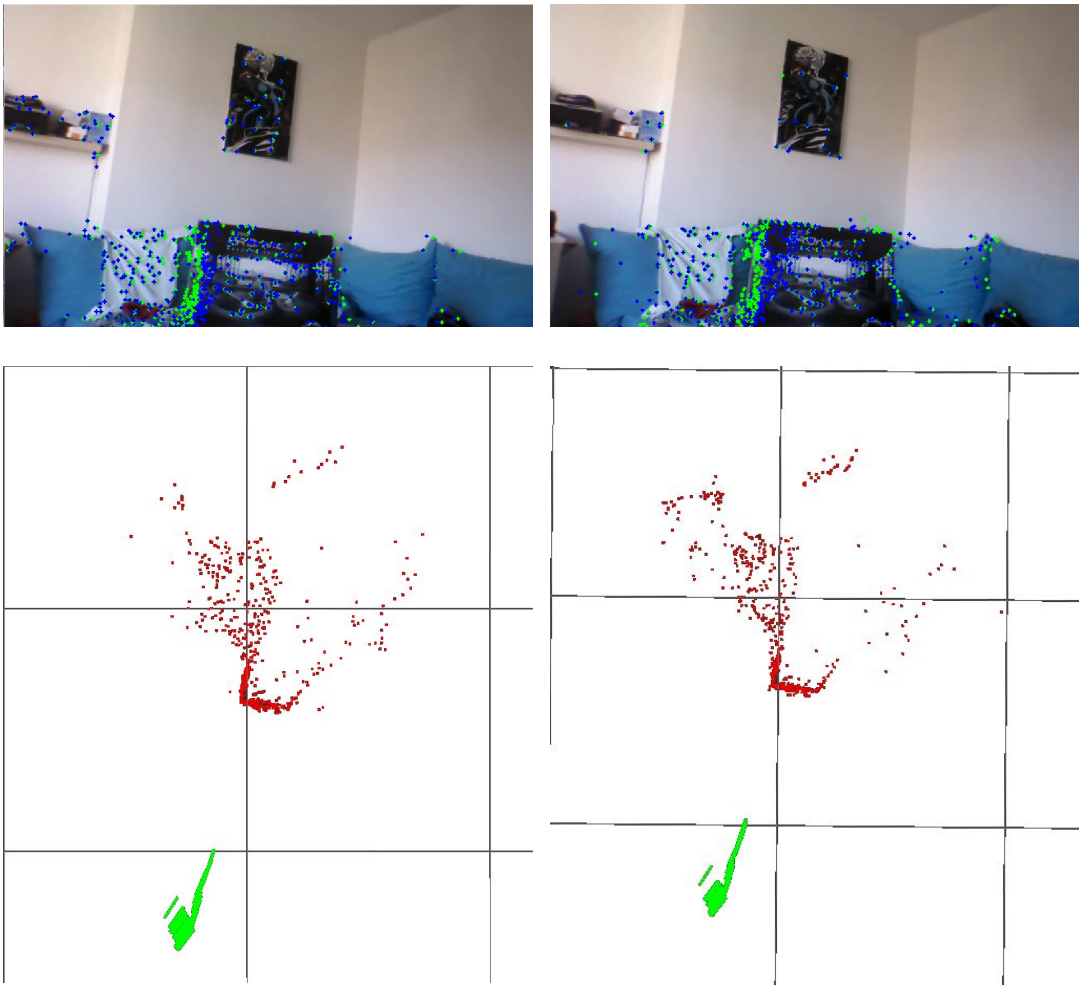

**Fig. 26. Reprojection error vs. model stability.** The right part represents the Kalman filter estimator with the Pose change filter and the left part represents the Kalman filter estimator with the Pose filter. Pose change filter is providing a more stable point cloud but at the cost of high RE (top row) resulting in more outliers in the model (bottom row). It can be seen on the points representing shelf and painting on the wall. Pose change filter providing more stable point cloud than pose filter at same RPR may seem unexpected as pose filter tends to stabilize the position. It is caused by higher filter variance that was necessary to set in order to achieve similar RPR (see attachment A1). Higher variance allows more noisy measurements to pass through the filter.

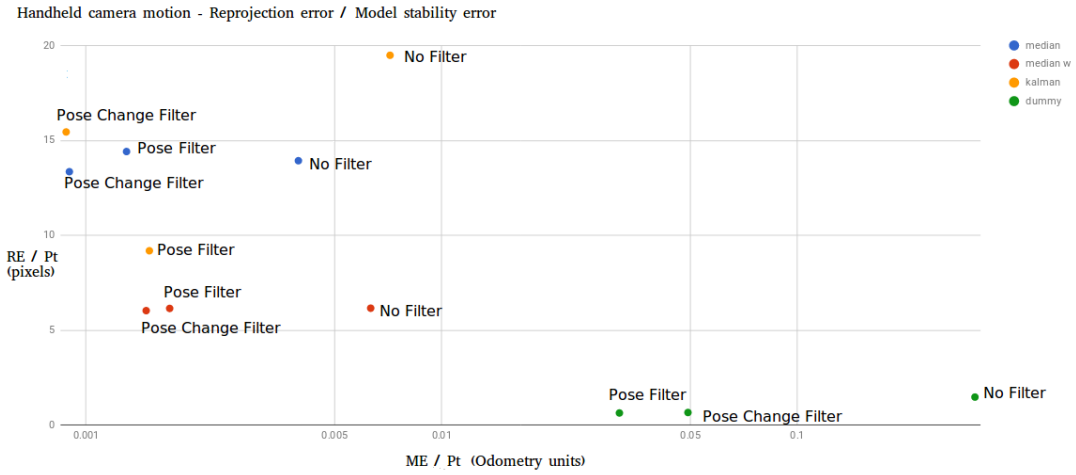


**Fig. 27 Handheld camera motion - overall performance.** Performance of each Position estimator coupled with each Position filter.

| Estimator | Kalman filter | Median - SW | Median | Dummy |
|---|---|---|---|---|
| RPR | 0,487 | 0,461 | 0,43 | 0,639 |
| Average depth | 1.0455 | 0.9989 | 1.0511 | 0.9571 |

**Table 4. Handheld camera motion RPR and Average depth.**

In figure 28, we depict challenging situation in handheld scenario with temporally small frame disparity of frames used for triangulation. As can be seen, we were able to account for short periods of time with noisy measurements by using Position estimator and Position filter (in this case Median SW estimator with Pose change filter). In figure 29, we are showing another challenging situation with a sufficient frame disparity but with a projective transformation of frames used for triangulation. In this case, we are also able to refine the results, but only for points close to camera - more distant points were filtered out. Comparison of these challenging scenarios can be found in figure 30. As can be seen, the projective transformation is introducing very noisy measurements which we are able to filter out only in exchange for a higher RE.

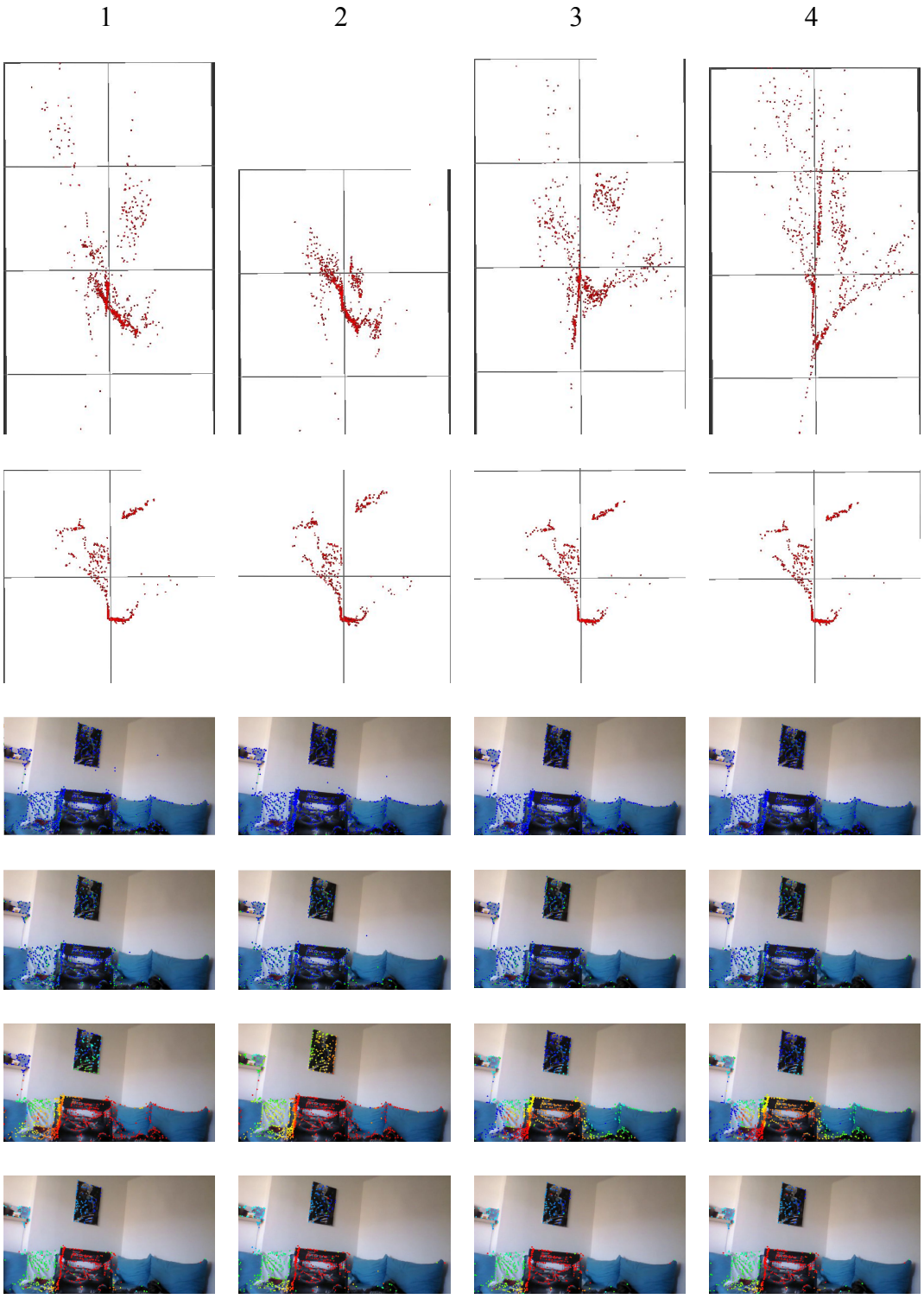


**Fig. 28. Performance under small frame disparity.** The second row (in the first one, there are numbers of frames) depicts 4 consequent frames of highly inaccurate measurements with no Position estimator and no filtering (top view of the generated point cloud). In the third row, there is a point cloud of the same frames, but with Median SW position estimator coupled with Pose change filter for outliers' removal. In rows below, there are corresponding reprojection and depth images.

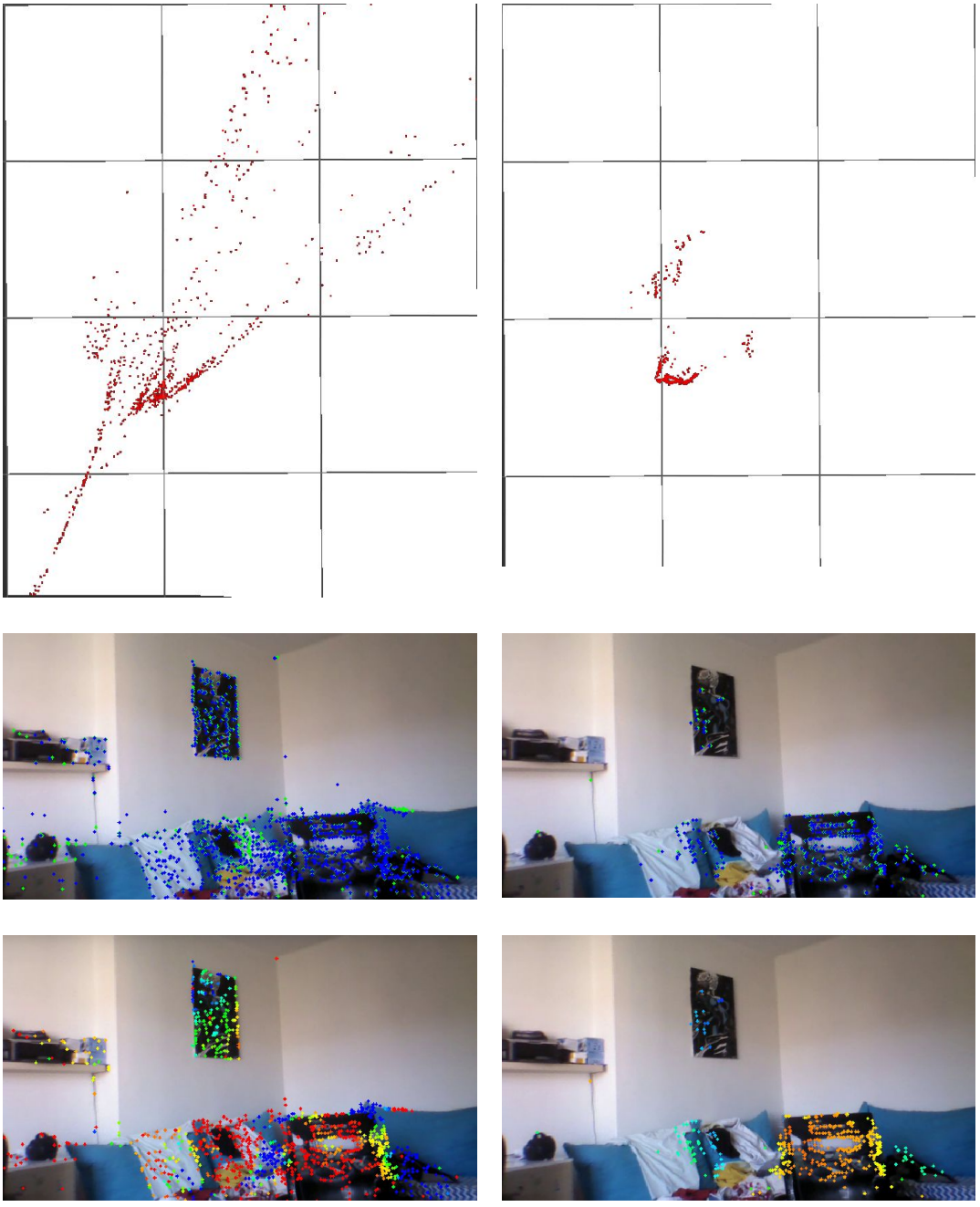

**Fig. 29. Performance under projective transformation.** Comparison of the system performance with no Position estimator and no Position filter (left) and the system performance with Median SW estimator with Pose change filter (right) under projective transformation of frames used for triangulation. The presence of projective transformation is sensible from two sources: 1. The angle between the two camera poses and 2. higher RE and ME of the raw data together with a sufficient frame disparity indicates the projective transformation as well.

| Dummy | 1 | 2 | 3 | 4 | Projective |
|---|---|---|---|---|---|
| ME / Pt | 0,206892 | 0,328321 | 0,131843 | 0,707268 | 5,34331 |
| RE / Pt | 0,644564 | 0,319892 | 0,134939 | 0,214536 | 3,29723 |
| FD | 0,012 | 0,007 | 0,007 | 0,0095 | 0,023 |

| Median SW | 1 | 2 | 3 | 4 | Projective |
|---|---|---|---|---|---|
| ME / Pt | 0,00193 | 0,00114 | 0,0017 | 0,000992 | 0,0015913 |
| RE / Pt | 6,83184 | 7,1644 | 7,61134 | 9,02981 | 19,84422 |
| FD | 0,012 | 0,007 | 0,007 | 0,0095 | 0,023 |

**Table 5. Performance of the worst and best estimator in challenging scenarios.** Performance under small frame disparity and under projective transformation of frames used for triangulation. The measurements are in columns - first 4 columns correspond to 4 examples in figure 28 and last column corresponds to measurements of the examples in the projective transformation example in figure 29.

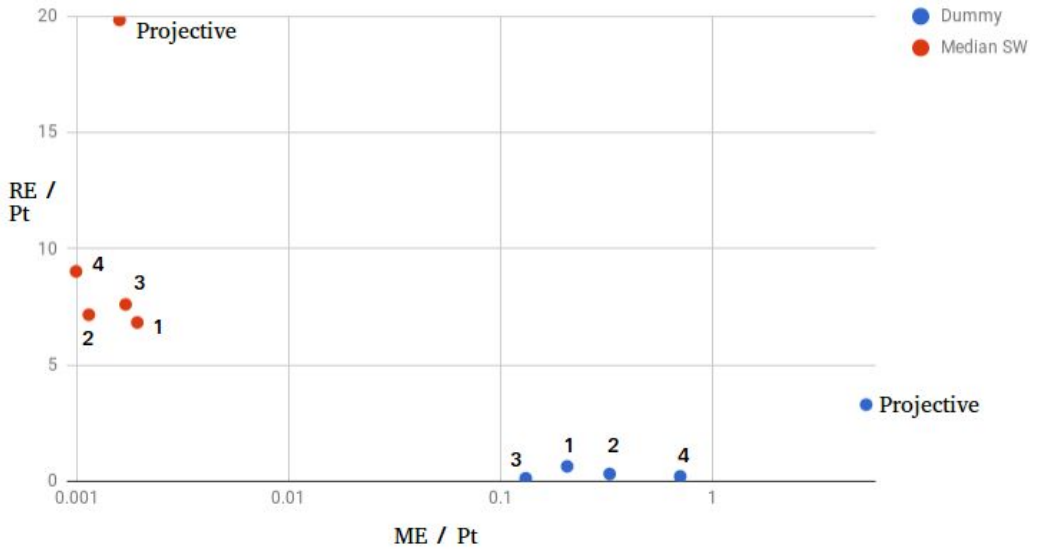


**Fig. 30. Overall performance in challenging scenarios.** Numbers 1 - 4 represent examples in figure 28 - small frame disparity scenario and points with label "Projective" represents example in figure 29 - projective transformation scenario. The graph shows the performance of the best and worst estimator in a small frame displacement scenario and a projective transformation scenario.

## 5.4 Observations

Below we list some observations related to behaviour of Position estimators and Position filters under different conditions and in different scenarios. Since the system performance is directly dependent on image processing module (feature detection, matching and tracking) and odometry module, we prefer to provide comparisons of different system settings and overall performance rather than exact measurements which can differ for different odometry and image processing methods.

The Kalman filter coupled with a Pose filter is giving the best results in case of pure lateral motion. As such, it outperformed all other estimators in all attributes. In case of rotation, the results of all estimators are not sufficient - Median SW estimator with a Pose filter is providing the best results among other estimators, but still, the performance (without any relevant measurements with appropriate frame disparity) is poor. Trade-off between model stability and reprojection error is most visible in the last case - handheld motion (see figure 27). Setting filter too tight results in stable model but at the cost of high reprojection error what is indicating an inaccurate model.

**Filters:**

The Pose filter has better results in case of lateral motion and the Pose change filter has better results in case of handheld camera motion which is caused by higher volume of noise in the handheld case. In general, the Pose change filter allows for points to “move” (how fast is depending on variance of the filter) which is advantageous in cases when periods of accurate measurements are interrupted by periods of noisy measurements. In such cases, the small movement of the whole model (allowed by Pose change filter) may result in a lower RPR and a lower RE (as proven in our experiments) and yet the variance of Pose change filter can still be set low. When using Pose filter in the presence of noise in our experiments, we had to set the variance of the filter high in order to reach a similar RPR as with Pose change filter which resulted in higher ME and in some cases also RE which means overall worse result. Considering the fact that we are not optimising the global map, but rather a certain amount of recent measurements, we can conclude that for real world

scenarios it is preferable to use a Pose change filter. However, both filters are significantly improving results of all estimators.

**Estimators:**

Median SW estimator is providing better results than Median estimator which is caused by accumulated erroneous measurements over time in case of the Median estimator. In addition, the Median estimator is not operating real time (see section 3.2.3.1.1). As of our tests, the Kalman filter estimator is better in the presence of stable noise. We believe that tuning the Kalman filter parameters (mostly measurement model covariance) during run-time would increase the depth estimation accuracy in terms of RE, ME and also RPR. Monitoring frame disparity, model stability and reprojection error provide enough information for determining the quality of the measurements.

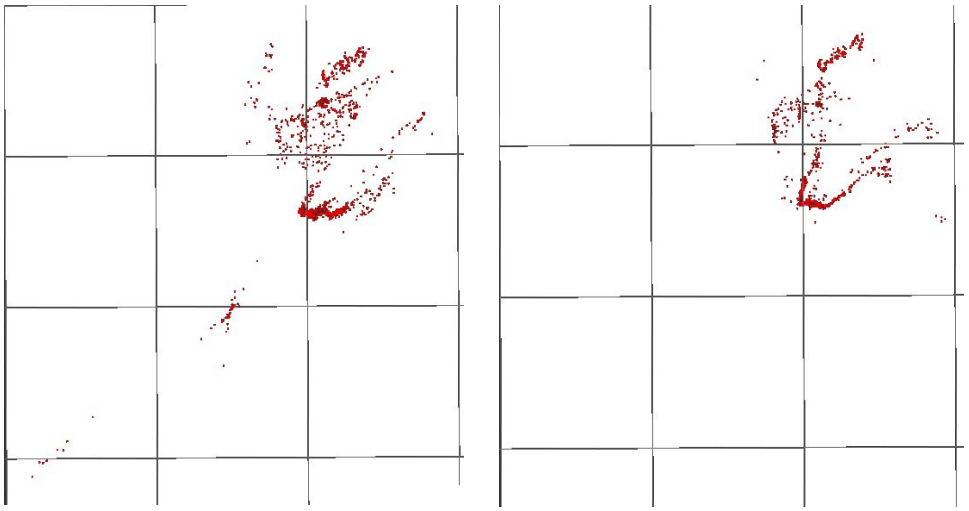

**Fig. 31. Removing chiral points.** Note the difference in the bottom left quadrant of the pictures. On the left, chiral points are visible whereas on the right image the chiral points are filtered out.

In every case, by using pose estimators and filters, we were able to remove the problem of chirality (see figure 31 and also figure 29) and reduce inaccuracies introduced by projectivity and tracking drift and occlusions.

We can conclude the following: in case of lateral motion (low amount of noise), the optimal solution is provided by the Kalman filter estimator coupled with the Pose filter. In case of a projective transformation, when the measurements are

noisy even in the presence of an appropriate frame disparity, the higher RE is acceptable (in our tests we have reached 20 pixels per point) in exchange for a stable point cloud providing more accurate depth map. Under the current conditions (inadaptive Kalman filter), in the presence of noise it is preferable to use the Median SW pose estimator coupled with the Pose change filter as it provides the most stable model with the lowest RE. The system is working correctly when enough relevant measurements (with low noise) are present.

# Conclusion

This work is aimed at providing a pioneering implementation of the depth estimation system with a decoupled pose estimation from image processing with sufficient theoretical background necessary for understanding the concepts of structure from motion problem and feature detection, matching, and tracking.

We have proposed a decoupled depth estimation system composed of simple techniques with 3 different methods for pose estimation of triangulated points to increase accuracy of measurements and 2 different filtering methods for outliers' removal. We have identified problems related to feature tracking drift and occlusions, problems introduced by projective interframe transformation and problems related to insufficient frame disparity. Our method is able to account for these problems by using mentioned estimation and filtering methods.

We have run multiple tests in 3 common real-world scenarios consisting of lateral motion of the camera, pure rotation and arbitrary handheld camera motion. We have also provided a measurement technique that is able to measure the relative system performance without ground truth measurements based on a trade-off between model stability and a reprojection error. Our system proved to work correctly under assumptions of a sufficient amount of relevant measurements.

For the future, it would be beneficial to implement a different triangulation technique that is projective invariant and use a threshold for the minimum frame disparity used for triangulation. Such approach may increase accuracy of measurements and thus provide more stable results.

# References


[1] ANTIĆ, B., NIÑO CASTANEDA, J.O., ĆULIBRK, D., PIŽURICA, A., CRNOJEVIĆ, V., PHILIPS, W. *Robust Detection and Tracking of Moving Objects in Traffic Video Surveillance*. In: Advanced Concepts for Intelligent Vision Systems (ACIVS). 2009. Lecture Notes in Computer Science. Volume 5807. Springer. Berlin. ISBN 978-3-642-04697-1.

[2] BAY H., TUYTELAARS T., VAN GOOL L. *SURF: Speeded Up Robust Features*. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/11744023_32

[3] CALONDER M., LEPETIT V., STRECHA C., FUA P. *BRIEF: Binary Robust Independent Elementary Features*. In: Daniilidis K., Maragos P., Paragios N. (eds) Computer Vision – ECCV 2010. ECCV 2010. Lecture Notes in Computer Science, vol 6314. Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-15561-1_56

[4] EIGEN, D., PUHRSCH, CH., FERGUS, R. *Depth map prediction from a single image using a multi-scale deep network.* In: Proceedings of the 27th International Conference on Neural Information Processing Systems. Volume 2. Montreal. 2014. Pages 2366 - 2374. ISBN 9781510800410.

[5] ENGEL, J., KOLTUN, V., CREMERS, D. *Direct Sparse Odometry.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 40. Issue 3. 2017. Pages 611 - 625. ISSN: 0162-8828.

[6] ENGEL, J., SCHÖPS, T., CREMERS, D. *LSD-SLAM: Large-Scale Direct Monocular SLAM.* In: European Conference on Computer Vision. ECCV 2014. Lecture Notes in Computer Science. Volume 8690. 2014. Pages 834 - 849. ISBN 978-3-319-10605-2.

[7] FARAGHER, R. *Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]*. In: IEEE Signal Processing Magazine. Volume 29. Issue 5. 2012. ISSN 1053-5888.

[8] FINCH, T. *Incremental calculation of weighted mean and variance.* In: University of Cambridge. 2009.

[9] FORSTER, C., PIZZOLI, M., SCARAMUZZA, D., *SVO: Fast semi-direct monocular visual odometry*. In IEEE Int. Conf. on Robotics and Automation (ICRA), Hong Kong, 2014. DOI: 10.1109/ICRA.2014.6906584. ISSN: 1050-4729.

[10] FORSTER, C., ZHANG, Z., GASSNER, M., WERLBERGER M., SCARAMUZZA, D. *SVO: Semi-Direct Visual Odometry for Monocular and Multi-Camera Systems*. IEEE Transactions on Robotics, Vol. 33, Issue 2, pages 249-265, Apr. 2017. DOI: 10.1109/TRO.2016.2623335. ISSN: 1552-3098.

[11] FRAUNDORFER, F., SCARAMUZZA, D., *Visual odometry: Part II - Matching, robustness, optimization, and applications.* IEEE Robotics and Automation Magazine. 2012. Volume 19. issue 2. Pages 78 - 90. ISSN: 1070-9932. Doi: 10.1109/MRA.2012.2182810.

[12] FUSIELLO, A., TRUCCO, E., VERRI, A. *A compact algorithm for rectification of stereo pairs.* Machine Vision and Applications. Volume 12. Issue 1. 2000. Pages 16-22. ISSN 1432-1769.

[13] GARG, R., KUMAR, V., CARNEIRO, G., REID, I. *Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue.* In: European Conference on Computer Vision. ECCV 2016. Lecture Notes in Computer Science. Volume 9912. Springer, Cham. ISBN 978-3-319-46484-8.

[14] GROVES, P. *Principles of GNNS, Inertial, and Multisensor Integrated Navigation Systems.* Second Edition. 2013. Artech House. Boston. ISBN 9781608070053.

[15] HARRIS, C., STEPHENS, M. J. *A combined corner and edge detector*. In Alvey Vision Conference. 1988. pp. 147–152.

[16] HARTLEY, R. I. *Chirality*. International Journal of Computer Vision. Kluwer Academic Publishers. Volume 26. Issue 1. pp: 41 - 61. 2018. ISSN 0920-5691.

[17] HARTLEY, R. I.. *Theory and Practice of Projective Rectification*. International Journal of Computer Vision. Volume 35. 1999. Pages 115-127. ISSN 1573-1405.

[18] HARTLEY, R., STURM, P., *Triangulation*, ARPA Image Understanding Workshop 1994, Monterey, CA, 1994, pp. 957–966. Doi: 10.1006/cviu.1997.0547

[19] HARTLEY, R. I., ZISSERMAN, A., *Multiple View Geometry in Computer Vision.* Cambridge University Press. 2004. Second Edition. ISBN 0521540518.

[20] HASSABALLAH, M., ABDELMGEID, A.A., ALSHAZLY, H.A.: *Image Features Detection, Description and Matching*. In: Image Feature Detectors and Descriptors. Studies in Computational Intelligence. Volume 630. Springer, Cham. 2016. ISBN 978-3-319-28852-9.

[21] HUANG, X., FAN, L., ZHANG, J., WU, Q., YUAN, CH. *Real Time Complete Dense Depth Reconstruction for a Monocular Camera.* In: IEEE Conference on Computer Vision and Pattern Recognition Workshops. CVPRW. IEEE 2016. Las Vegas. 2016. Pages 674 - 679. ISBN: 978-1-5090-1437-8.

[22] JULIER, S. J., UHLMANN, J. K. *Unscented filtering and nonlinear estimation.* In: Proceedings of the IEEE. Volume 92. Issue 3. 2004. Pages 401 - 422. ISSN: 0018-9219.

[23] KALMAN, R. E. *A New Approach to Linear Filtering and Prediction Problems.* In: Transactions of ASME, Journal of Basic Engineering. Volume 82. Issue 1. 1960. pages 34 - 35. doi:10.1115/1.3662552.

[24] KLEIN, G., MURRAY, D. *Parallel Tracking and Mapping for Small AR Workspaces*. In: IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR). 2007. ISBN 978-1-4244-1749-0.

[25] LEUTENEGGER S., CHLI M., SIEGWART R. *Brisk: Binary robust invariant scalable keypoints.* In Proc. International Conference on Computer Vision. 2011. Pages 2548-2555.

[26] LIU, F., SHEN, CH. LIN, G. *Deep Convolutional Neural Fields for Depth Estimation from a Single Image.* In: Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition. Volume 7. 2015. Pages 5162 - 5170. ISBN 9781467369640.

[27] LOWE, D. G. 2004. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision. Volume 60. Issue 2. 2004. Pages 91-110. DOI: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[28] LUCAS, B. D., KANADE, T. *An Iterative Image Registration Technique with an Application to Stereo Vision*. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI). Vancouver. 1981. Volume 2. Pages 674 - 679.

[29] MA, Y., SOATTO, S., KOSECKA, J., SASTRY, S.: *An Invitation to 3-D Vision: From Images to Geometric Models.* Interdisciplinary Applied Mathematics - Volume 26. Springer Science & Business Media. 2005. ISBN-10: 0-387-00893-4.

[30] MAGYAR G., SINČÁK P., KRISZÁN Z. Comparison Study of Robotic Middleware for Robotic Applications. In: Sinčák P., Hartono P., Virčíková M., Vaščák J., Jakša R. (eds) Emergent Trends in Robotics and Intelligent Systems. Advances in Intelligent Systems and Computing, vol 316. Springer, Cham. 2015. ISBN 978-3-319-10783-7.

[31] MANDELBAUM, R., KAMBEROVA, G., MINTZ, M. *Stereo depth estimation: a confidence interval approach.* In: Sixth International Conference on Computer Vision. Bombay. 1998. ISBN 81-7319-221-9.

[32] NEWCOMBE, R. A., LOVEGROVE, S. J., DAVISON, A. J. *DTAM: Dense tracking and mapping in real-time.* In: Proceeding of the 2011 International Conference on Computer Vision. ICCV 2011. Washington. 2011. ISBN: 978-1-4577-1101-5.

[33] PIZZOLI, M., FORSTER, CH., SCARAMUZZA, D. *REMODE: Probabilistic, monocular dense reconstruction in real time.* In: International Conference on Robotics and Automation. ICRA 2014. Hong Kong. 2014. ISBN: 978-1-4799-3685-4.

[34] PRONEÇA, P. F., GAO, Y. *SPLODE: Semi-probabilistic point and line odometry with depth estimation from RGB-D camera motion.* In: IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS 2017. Vancouver. 2017. ISSN 2153-0866.

[35] RUBLEE E., RABAUD K., KONOLIGE K. BRADSKI G. *ORB: An efficient alternative to sift or surf.* In Proc. IEEE International Conference on Computer Vision (ICCV). Barcelona. 2011. Pages 2564-2571.

[36] SAXENA, A., SCHULTE, J., NG. A. Y., *Depth estimation using monocular and stereo cues.* In: Proceedings of the 20th international joint conference on

Artificial intelligence. IJCAI 2007. Hyderabad. 2007. Pages 2197 - 2203. ISBN 9781577352983.

[37] SAXENA, A. SUN, M., NG., A. Y. *Make3D: learning 3d Scene Structure from a Single Still Image.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Volume 31. Issue 5. 2009. Pages 824 - 840. ISSN: 0162-8828.

[38] SCARAMUZZA, D., FRAUNDORFER, F. *Visual Odometry [Tutorial]*. In: IEEE Robotics & Automation Magazine. Volume 18. Issue 4. 2011. ISSN 1070-9932.

[39] SHAPIRO L., STOCKMAN C., *Computer Vision.* Prentice Hall. 2001. ISBN 0130307963.

[40] SHI, J., TOMASI, C. *Good Features to Track*. Technical Report. Cornell University, Ithaca, NY, USA. 1993.

[41] SMITH, W. A. P., RAMAMOORTHI, R., TOZZA, S. *Linear Depth Estimation from an Uncalibrated, Monocular Polarisation Image.* In: European Conference on Computer Vision. ECCV 2016. Lecture Notes in Computer Science. Volume 9912. 2016. Springer. Cham. ISBN 978-3-319-46484-8.

[42] STRANG G.: *Introduction to Linear Algebra.* Fifth edition. Wellesley-Cambridge Press. 2016. ISBN 0980232775, 9780980232776.

[43] SZELISKI R., *Computer Vision: Algorithms and Applications.* Springer. 2010. ISBN 978-1-84882-935-0.

[44] TAO, M. W., WANG, T., MALIK, J., RAMAMOORTHI, R. *Depth Estimation for Glossy Surfaces with Light-Field Cameras.* In: Computer Vision - ECCV 2014 Workshops. ECCV 2014. Lecture Notes in Computer Science. Volume 8926. 2014. Springer. Cham. Pages 533 - 547. ISBN 978-3-319-16181-5.

[45] TOMASI, C., KANADE, T. *Shape and motion from image streams under orthography: a factorization method.* International Journal of Computer Vision. Volume 9. Issue 2. 1992. Pages 137-156. ISSN 1573-1405.

[46] TSARDOULIAS, E., MITKAS, A.P. Robotic frameworks, architectures and middleware comparison. ArXiv e-prints. 2017. Eprint arXiv: 1711.06842.

[47] TUYTELAARS, T., MIKOLAJCZYK, K.: *Local invariant feature detectors: a survey*. Foundations and Trends in Computer Graphics and Vision. Volume 3. 2007. Page 177–280.

[48] XU G., ZHANG Z., *Epipolar Geometry in Stereo, Motion and Object Recognition*. Springer Science+Business Media Dordrecht. 1996. ISBN 978-94-015-8668-9.

# List of figures

# List of tables

# List of abbreviations

| | |
|---|---|
| DLT | Direct Linear Transformation |
| ME | Model stability Error |
| NCC | Normalized Cross Correlation |
| Pt | Point |
| RE | Reprojection Error |
| RPR | Rejected Points Ratio |
| SLAM | Simultaneous Localization and Mapping |
| SSD | Sum of Squared Differences |
| SVD | Singular Value Decomposition |
| VO | Visual Odometry |

# A. Attachments

## A.1 Experiments measurements

Below we provide a complete results of all experiments ordered by Position estimator and camera motion. Each table represents results for particular Position estimator and camera motion. In each table there are measurements for each Position filter (None, Pose and Pose Change filter) of accepted points per frame (PT / Frame), rejected points per frame (RP/F), Reprojection error per one point (RE / pt), Model stability error per one point (ME / pt), average depth of a point (Av depth) and variance value used for point position filter ( $\sigma^2$ ).

## A.1.1 Dummy pose estimator

Lateral motion

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 643.525 | 0 | 0.31765536 | 0.01633502 | 815.543 | - |
| Pose | 422.474576 | 229.74 | 0.11719670 | 0.011356154 | 412.073 | 0.03 |
| Pose change | 326.336134 | 323.60 | 0.11332183 | 0.009035716 | 291.597 | 0.017 |

**Table. A1.** Dummy estimator, lateral motion

Rotation

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 541.803738 | 0 | 3.10794680 | 2.81691477 | 1204.54 | - |
| Pose | 131.691588 | 418.06 | 0.26846000 | 0.17107728 | 103.477 | 0.3 |
| Pose change | 133.971698 | 413.95 | 0.29605731 | 0.17247447 | 107.701 | 0.3 |

**Table. A2.** Dummy estimator, rotation

Hand held

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 867.694656 | 0 | 1.48025389 | 0.31560553 | 1121.23 | - |
| Pose | 327.847942 | 580.5 | 0.64313269 | 0.03159739 | 313.514 | 0.1 |
| Pose change | 384.908592 | 535.39 | 0.66841767 | 0.04924267 | 402.866 | 0.15 |

**Table. A3.** Dummy estimator, hand held camera motion

## A.1.2 Median pose estimator

Lateral motion

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 638.641025 | 0 | 2.85184887 | 0.00184810 | 700.286 | - |
| Pose | 463.880341 | 182.38 | 2.95434277 | 0.00098952 | 449.178 | 0.016 |
| Pose change | 448.754386 | 193.25 | 3.49472223 | 0.00091967 | 438.729 | 0.008 |

**Table. A4.** Median estimator, lateral motion

Rotation

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 540.721153 | 0 | 6.23638303 | 0.098647283 | 527.674 | - |
| Pose | 239.669811 | 308.25 | 2.41302499 | 0.010691123 | 185.742 | 0.1 |
| Pose change | 269.6 | 275.83 | 3.27763529 | 0.010778896 | 215.724 | 0.08 |

**Table. A5.** Median estimator, rotation.

Hand held

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 839.89230 | 0 | 13.929258 | 0.00395499 | 1005.46 | - |
| Pose | 457.001976 | 414.95 | 14.414361 | 0.00130010 | 499.152 | 0.03 |
| Pose change | 499.073308 | 376.55 | 13.352077 | 0.00089795 | 524.566 | 0.01 |

**Table. A6.** Median estimator, hand held

## A.1.3 Sliding window Median estimator

Lateral motion

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 635.289473 | 0 | 2.67159604 | 0.00218964 | 687.644 | - |
| Pose | 452.588235 | 197.35 | 1.67304764 | 0.00087283 | 466.214 | 0.01 |
| Pose change | 462.577981 | 182.75 | 2.18714424 | 0.00087931 | 472.586 | 0.006 |

**Table. A7.** Sliding window Median estimator, lateral motion

Rotation

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 538.123809 | 0 | 7.11917951 | 0.12086243 | 537.397 | - |
| Pose | 234.89423 | 308.65 | 1.49151418 | 0.00889925 | 183.532 | 0.0548 |
| Pose change | 275.460784 | 273.33 | 5.46553012 | 0.00940153 | 230.177 | 0.038 |

**Table. A8.** Sliding window Median estimator, rotation

Hand held

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 858.49903 | 0 | 6.162345 | 0.0063150 | 998.718 | - |
| Pose | 439.107899 | 428.04 | 6.150454 | 0.0017191 | 455.492 | 0.017 |
| Pose change | 476.806331 | 407.16 | 6.033197 | 0.0014765 | 476.272 | 0.005 |

**Table. A9.** Sliding window Median estimator, hand held motion

## A.1.4 Kalman filter

Lateral motion kf_meas 0.001

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 636.956521 | 0 | 3.51508532 | 0.00172576 | 689.722 | - |
| Pose | 463.371681 | 172.33 | 0.75072095 | 0.00075928 | 471.851 | 0.014 |
| Pose change | 435.759615 | 190 | 1.65964165 | 0.00087079 | 426.812 | 0.0045 |

**Table. A10.** Kalman filter estimator, lateral motion

Rotation kf_meas 0.8

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 541.80373 | 0 | 7.58654891 | 0.09676349 | 797.804 | - |
| Pose | 231.714285 | 314.28 | 3.06285244 | 0.01294393 | 169.658 | 0.16 |
| Pose change | 279.792452 | 268.13 | 3.12072628 | 0.01295104 | 217.21 | 0.1 |

**Table. A11.** Kalman filter estimator, rotation

Hand held motion kf_meas 0.2

| Filter | PT / Frame | RP/F | RE/pt | ME/pt | Av depth | $\sigma^2$ |
|---|---|---|---|---|---|---|
| None | 864.75 | 0 | 19.482890 | 0.00715811 | 1079.09 | - |
| Pose | 449.699619 | 427.62 | 9.187079 | 0.00150741 | 470.181 | 0.045 |
| Pose change | 477.412213 | 434.20 | 15.445588 | 0.00087952 | 494.012 | 0.0128 |

**Table. A12.** Kalman filter estimator, hand held motion

# A.2 Test runs data

Attached file “tests_data.csv” that contains collected data that served for evaluation. File is organized in columns as follows:

- Id - id of test run.
- Estimator -name of used Position estimator.
- Filter - name of used Position filter.
- Test label - type of motion scenario (rot / lat / hand).
- Med window - size of sliding window used for Median SW Position estimator.
- Kf_proc noise - value used for Kalman filter estimator process noise cov.
- Kf_meas noise - value used for Kalman filter estimator measurement noise cov.
- Filter Variance - variance threshold used for Pose or Pose change filter
- Pts total - number of all points used for depth estimation (accepted by filter)
- Frame count - number of frames used in test
- Rej pts - number of rall ejected points by filter
- Average frame disparity
- Total RE - complete Reprojection Error of the test run
- Total ME - complete Model stability Error of the test run
- Average depth - average depth of frame
- Av depth / pt - average depth per point
- pts/frame - average point count per frame
- RE / frame - average Reprojection error of all points in one frame
- ME / frame - average Model stability error of all points in one frame
- RE / point - average Reprojection error per point
- ME / point - average Model stability error per point
- RP / frame - number of rejected points by filter per one frame
- RPR - Rejected points ratio

## A.3 Source code

Attached file “depthest.zip” contains source code of depthest package.

# A.4 Sample files

Attached compressed folder “depthest_test_bag_files.zip” contains 3 .bag files with sample data. Because of the significant size of the sample files we are providing only 3 sequences that were used for evaluation. Attached files represents 3 scenarios described in section 5.3.

We attache these files for convenience. The system as such cannot run without odometry data and tracked features coordinates.

Each .bag file consist of 3 recorded topics that are necessary to run the system:

- “/image_undist” - undistorted image used for 2D visualisations.
- “/std_pose” - odometry data.
- “/tracked_features” - array of tracked features in the image.