

AGH

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W
KRAKOWIE**

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA TELEKOMUNIKACJI

PRACA DYPLOMOWA MAGISTERSKA

Algorithm for Visual Odometry.

Algorytm do wyznaczania pozycji i orientacji obiektu na podstawie sekwencji wideo.

Autorzy: *Krzysztof Szczęsny, Jan Twardowski*
Kierunek studiów: *Teleinformatyka*
Opiekun pracy: *dr inż. Jarosław Bułat*

Kraków, 2017

Uprzedzeni o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór; artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzeni o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchylające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczamy, że niniejszą pracę dyplomową wykonaliśmy osobiście i samodzielnie i że nie korzystaliśmy ze źródeł innych niż wymienione w pracy.

We would like to thank our remarkable supervisor, Jarosław Bułat – for his useful insights, substantial support, and limitless patience.

Contents

Abstract	7
Introduction	8
1. Theoretical background and state-of-the-art	11
1.1. Pinhole camera model	11
1.2. Stereo vision	14
1.3. Monocular visual odometry.....	16
1.4. Literature	18
1.5. The <i>Rebvo</i> algorithm outline	19
2. Analysis and improvements of <i>Rebvo</i> algorithm	23
2.1. Notation (Keyline structure) and main loop	23
2.2. Algorithm outline	25
2.3. Edge extraction	26
2.3.1. Edge detection algorithm choice	26
2.3.2. Data preprocessing	26
2.3.3. Difference of Gaussians strength tests.....	27
2.3.4. Depth initialization	35
2.3.5. Keyline joining	38
2.4. Edge tracking.....	39
2.4.1. Warping function	39
2.4.2. Auxiliary image	41
2.4.3. Keyline matching criteria	42
2.4.4. Energy minimization	43

2.4.5. Initial conditions.....	46
2.5. Mapping.....	47
2.5.1. Forward matching.....	48
2.5.2. Directed matching	48
2.5.3. Regularization.....	52
2.5.4. Depth reestimation.....	53
2.5.5. Scale correction	54
2.6. Tests on artificial data.....	55
2.7. Trajectory fitting	55
3. Discussion of results	57
3.1. Experimental setup	57
3.2. Trajectory comparison	59
4. Conclusion.....	67

Abstract

GPS applications (e.g. car navigation) struggle with precision during rapid changes of motion, e.g. when sharp turns are made during roundabout exit. Yet driver needs feedback from the positioning system particularly at such moments. Visual Odometry systems are usually more accurate (quick to act) than GPS in this regard. This work performs in-depth analysis of a chosen state-of-the-art, real-time VO algorithm and proposes some improvements that are especially suited for road scenarios. Analyzed algorithm was implemented in GNU Octave and tested using popular datasets. Much attention was paid to intermediate results. Tests show that algorithm converges quickly to the expected trajectory shape. Some challenges of urban scenarios were not solved, but solutions were suggested.

Keywords – visual odometry, motion estimation, edge detection

Abstrakt

Aplikacje korzystające z GPS (np. nawigacja samochodowa) mają niską dokładność lokalizacji podczas gwałtownych ruchów, np. przy zjeżdżaniu z ronda, chociaż właśnie wtedy informacja o pozycji jest dla kierowcy najważniejsza. Systemy odometrii wizyjnej zazwyczaj przewyższają w tym zakresie GPS. W niniejszej pracy przeprowadzono dogłębną analizę wybranego nowoczesnego algorytmu odometrii wizyjnej, mogącego pracować w czasie rzeczywistym na urządzeniu mobilnym. Na podstawie przeprowadzonych badań zaproponowano ulepszenia algorytmu, które mogą być przydatne szczególnie w warunkach drogowych. Analizowany algorytm został zaimplementowany w środowisku GNU Octave oraz przetestowany z użyciem popularnych sekwencji testowych. Podczas analizy dużo uwagi poświęcono wynikom pośrednim poszczególnych kroków algorytmu. Testy wykazały, że algorytm szybko zbiega do oczekiwanej trajektorii. Nie udało się wyeliminować wszystkich błędów, jakie mogą wystąpić w sekwencjach nagrywanych kamerą umieszczoną w samochodzie, ale wskazano możliwe rozwiązania.

Słowa kluczowe – odometria wizyjna, estymacja ruchu, wykrywanie krawędzi

Introduction

Odometry is a term describing measurement of distance (from Greek: *odos* – “path“, *metron* – “measure“). For instance, car odometer is a device that calculates total traveled distance through multiplication of wheel diameter by the number of wheel spins. Visual odometry (VO) aims to determine the *egomotion* of an object – its position relative to a rigid scene – by observing changes in video registered by one or more cameras attached to said object.

Main advantage of visual approach to odometry is ubiquity of cameras embedded into virtually every mobile device – be it a smartphone, game console or even a smartwatch. Not every of those devices has a gyroscope, and even if a GPS (Global Positioning System) chip is present, its locational accuracy is low. Practical applications of visual odometry include:

1. SLAM (Simultaneous-Localization And Mapping) used by autonomous robots, e.g. Mars Exploration Rovers, drones or self-driving cars. Calculation and maintenance of such maps is usually a computationally demanding task, however.
2. Handheld video stabilization, realized in software.
3. AR (Augmented Reality) – last year in particular saw the emergence of *Pokémon GO* [2], an acclaimed mobile game. Recently Google has released *ARCore* [3], a software development kit that internally uses many VO ideas.

Following this trend of development for mobile devices, a vision-based driving assistance system was initially envisioned for this thesis. It was meant to use peripherals available in every smartphone. Its purpose would have been a real-time navigation aid for consumer-grade smartphones – assistance for GPS in places where more precision is needed, e.g. on a roundabout, where exit roads are located too close to each other. Ultimately only a prototype of pure visual odometry system was created, but results and ideas for future improvement do not invalidate the original concept.

The developed and analyzed algorithm is very closely based on a solution presented in [1]. This work was chosen as it presented a novel approach, similar to efficient semi-dense methods.

Each step of the algorithm flow was analyzed and tested on video sequences from publicly available datasets that also contain ground truth data for validation purposes.

This thesis is divided into four chapters. Chapter 1 contains theoretical introduction and state-of-the-art. It is also explained what are the challenges of monocular vision that set it apart from the more popular stereo approach. Details of the presented algorithm are elaborated upon in Chapter 2. Final obtained results are given and commented in Chapter 3. The concluding Chapter 4 contains final remarks and directions of future development.

During algorithm development, Jan Twardowski was mainly responsible for edge joining (Sec. 2.3.5), post-minimization edge matching (Sec. 2.5.2) and regularization (Sec. 2.5.3); while Krzysztof Szczęsny – for energy minimization (Sec. 2.4.4) and depth reestimation (Sec. 2.5.4). Other algorithm sections and testing were performed jointly.

1. Theoretical background and state-of-the-art

In this chapter theory behind computer vision, 3D reconstruction and mathematical apparatus used in our algorithm are introduced. Pinhole camera model and basics of stereo and mono vision are briefly discussed. Literature related with this topic is overviewed. Finally, *Rebvo* [1] algorithm flow is described.

1.1. Pinhole camera model

Pinhole camera model [4] is a widely used and simple model that establishes connection between world coordinates \mathbb{R}^3 and image-domain coordinates \mathbb{R}^2 (i.e. projective geometry).

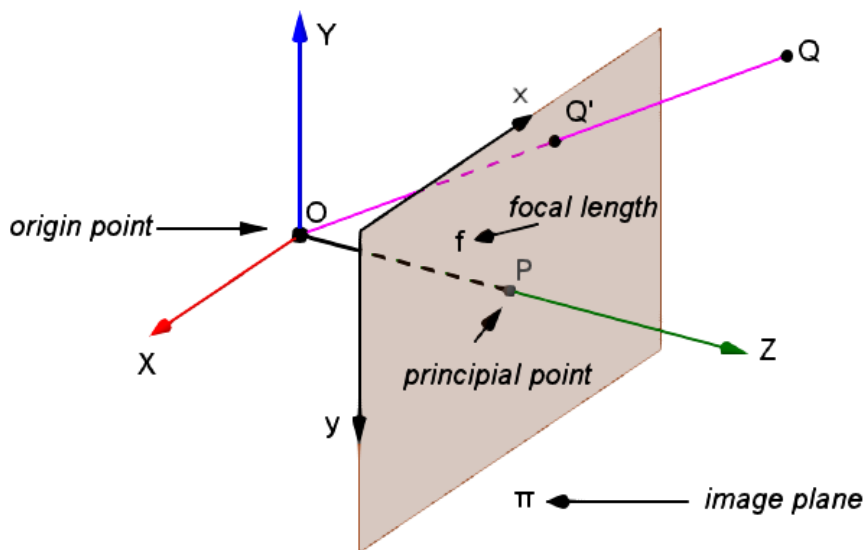


Fig. 1.1. Image formation in pinhole camera model. Source: [5]

Operation of pinhole camera is depicted in Fig. 1.1. Every 3D point Q can be associated with a ray QO that passes through camera origin O , which in the simplest case can be defined as origin of the 3D coordinate system. Such ray can be defined with homogeneous coordinates as set of points $\{(X, Y, Z)\}$ that satisfy (1.1):

$$(X, Y, Z) = k(Q_x, Q_y, Q_z) \quad (1.1)$$

where:

k – real parameter, $k \neq 0$,

Q – world coordinates point.

Image plane π is a rectangle parallel to plane XOY . Its distance from origin is equal to f (focal length). Usually it is assumed that image plane's z coordinate is positive – otherwise formed image would be upside down. Point where axis OZ intersects π is called principal point.

World coordinate points Q are projected onto π as Q' , thus forming a 2D image. This relationship can be concisely written using camera matrix as in (1.2).

$$q = Q' \sim \underbrace{\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \mathbf{R} [\mathbf{I} \mid -t] Q \quad (1.2)$$

$\underbrace{\hspace{10em}}_{\mathbf{C}}$

where:

q – 2D point on image plane,

Q' – 2D point on image plane expressed in homogeneous coordinates (3-vector),

Q – 3D point corresponding to q , expressed in homogeneous coordinates (4-vector),

\sim – equality between homogeneous points,

\mathbf{K} – 3x3 camera intrinsic parameters matrix,

\mathbf{C} – 3x4 camera matrix,

f_k – focal length along axis k ($f_x \neq f_y$ if pixels are not square, but rectangular),

s – skew factor (usually equal to 0),

c_k – k -coordinate of the principal point,

\mathbf{R} – 3x3 rotation matrix between 3D world coordinates and camera coordinates,

t – translation vector between 3D world coordinates and camera coordinates (3-vector).

Real cameras do not fully conform to this model [5]. They contain lenses that enlarge field of view (see Fig. 1.2). Lens curve passing light rays in a nonlinear fashion (this can be observed in fish-eye cameras [6]). Moreover, lenses themselves are imperfectly made and aligned. Other nonlinearity sources include color aberration and CCD (Charge Coupled Device) sensor quantization noise [7], but this list is far from being exhaustive. All these phenomena account for distortions.

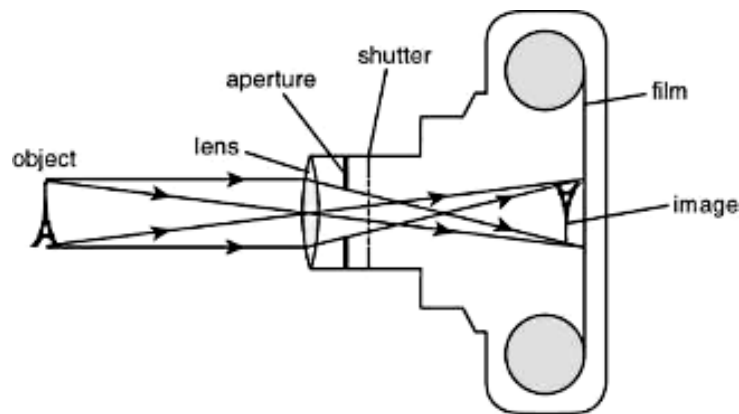


Fig. 1.2. Outline of an analogue camera. A digital camera would feature a CCD in place of film. Source: http://www.mauitroop22.org/merit_badges/images/camera.jpg

Conrady-Brown model [8] [9] is a classic approach to removing geometric distortions. The most significant distortion component is modeled with a radial, even-ordered polynomial, that is centered at the distortion center (usually located in proximity of the principal point). During camera calibration, coefficients of the said polynomial are measured – they are assumed to be constant¹ for given camera. Then each image taken by the camera can be rectified with inverse distortion field [11]. Example of such field is depicted in Fig. 1.3. More complex models [12], or even model-free methods [13] [14] also do exist, but calibration procedure is far more challenging. If undistortion procedure is successful, then one of the most fundamental projective

¹In fact they can vary with temperature, focus change and over long periods of time [10].

geometry properties is preserved – straight 3D world lines are mapped to straight 2D image lines [15].

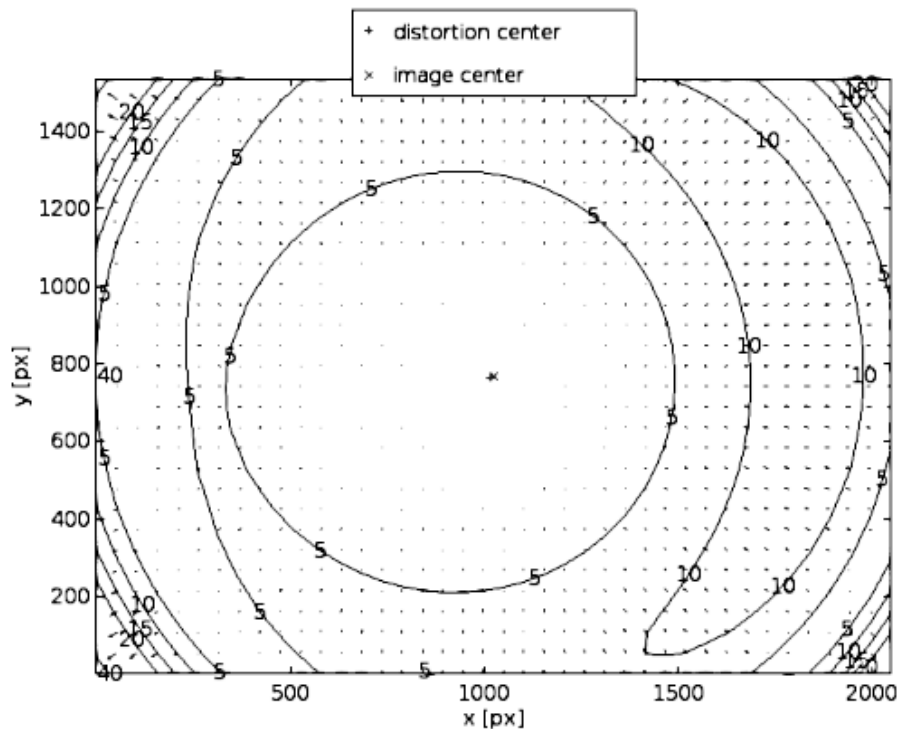


Fig. 1.3. Example of a distortion modeled with Conrady-Brown model. Vectors connect distorted pixel positions with their undistorted counterparts and contours mark areas with constant vector length. Tangential (non-radial) component is noticeable in this particular field. Source: [5]

1.2. Stereo vision

Most implementations of visual odometry systems use two cameras spaced by a constant baseline, that can be determined during stereo calibration. Abundance of such methods can be explained with similarity to how human visual system works [16]. Stereopsis (perception of depth) is a result of binocular vision determined by comparing object position seen by both eyes and by taking into account the baseline, i.e. spacing between eyes. This is illustrated in Fig. 1.4.

Correspondence between matched points is described by (1.3) with the fundamental matrix. At least 7 matching feature points (points with well defined surroundings) are needed to

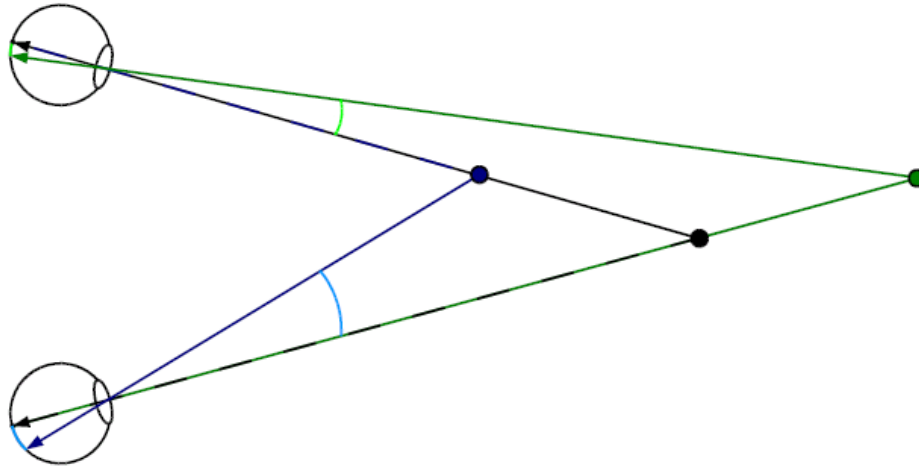


Fig. 1.4. Human stereo vision system. Source: https://commons.wikimedia.org/wiki/File:Binocular_disparity.png

obtain \mathbf{F} , but usually much more are used so as to mitigate noise [17]. When cameras are calibrated, essential matrix \mathbf{E} can be calculated with (1.4). Essential matrix can be then decomposed into rotation and translation between 3D points registered by both cameras using SVD (Singular Value Decomposition); 1 solution out of 4 has to be chosen. \mathbf{E} has scale ambiguity, which can be resolved using baseline in (1.5) [18]. It is worth mentioning that baseline can not be determined using autocalibration² alone – an object of known dimensions must be measured on images.

$$p_2^T \mathbf{F} p_1 = 0 \quad (1.3)$$

where:

p_i – point as registered by i -th camera, in homogeneous coordinates,

\mathbf{F} – 3x3 fundamental matrix.

$$\mathbf{K}_2^T \mathbf{F} \mathbf{K}_1 = \mathbf{E} = [t]_s \mathbf{R} \quad (1.4)$$

where:

\mathbf{K}_i – i -th camera intrinsic parameters matrix,

\mathbf{E} – 3x3 essential matrix,

²Calibration without any a priori knowledge about the scene.

t – translation vector $t = [t_x \ t_y \ t_z]^T$

$[t]_s$ – skew-symmetric matrix:
$$\begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix},$$

\mathbf{R} – 3x3 rotation matrix.

$$Z = \frac{fb}{|p_1 - p_2|} \quad (1.5)$$

where:

Z – world OZ coordinate of the point (i.e. depth),

f – focal length,

b – baseline.

1.3. Monocular visual odometry

There are two main issues in monocular visual odometry which will be discussed in this section: scale ambiguity and position drift over time.

In case when only one camera is available, matches between consecutive frames still can be searched for. However, without knowing the 3D transformation, baseline is unknown, so scene can be reconstructed only up to a scale [4] [19]. Main difficulty is that this transformation is *the* quantity that odometry is supposed to estimate. Information from only one camera is not sufficient to solve the ambiguity. For instance, Fig. 1.5 depicts a situation when two objects with differing dimensions appear to be of the same size after projection to image plane. There are 2 main ways of alleviating this problem.

One way is to use an IMU (Inertial Measurement Unit - accelerometer & gyroscope). Data from this unit can be used to estimate baseline and extrinsic camera parameters [20]. Actually, IMU alone can be used for odometry. By combining it with video input, however, accuracy and robustness can tremendously increase.

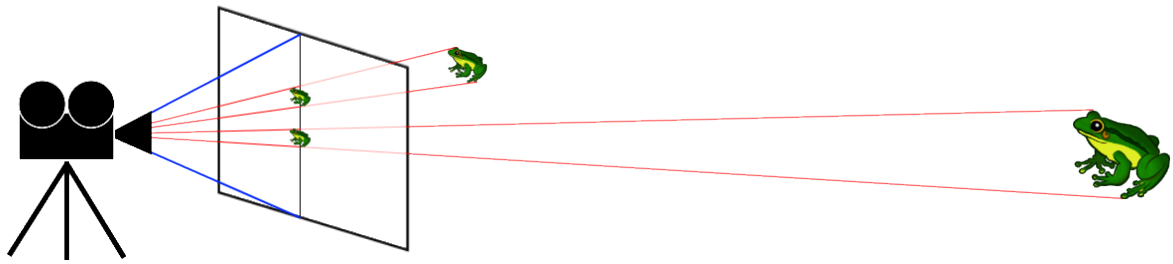


Fig. 1.5. Single pinhole camera scale ambiguity. Two objects have same size on the virtual image plane, despite differing in reality

Another approach is to use explicit depth data, e.g. RGB-D (Red, Green, Blue and Depth) from Kinect [21]. This helps in global scene scale estimation. A disadvantage is that such devices have narrow depth detection range (usually up to few meters [22]) and are not available in most consumer-grade mobile devices.

A very frequent problem in pure visual monocular odometry is unbounded position drift over time. Each 3D transformation that is estimated between consecutive frames is an instantaneous linear and angular velocity. To obtain accumulated position and azimuth after given frame, individual velocities have to be integrated. Each estimation error gains significance with time. In literature two ways of mitigation are the most common.

First of all, information from GPS can be employed to ensure that position does not drift away [23]. GPS signal quality greatly depends on location, so this approach may be unsuited for indoor use cases. GPS can not help with azimuth estimation, however. Finally, GPS is, out of the discussed methods, the only way to transform relative coordinates to real world coordinates (i.e. latitude and longitude). Other possibility would be to use special markers.

In SLAM systems loops in trajectories can be used as means of correcting position drift – knowing landmark³) descriptors in past frames, such loops can be detected and system parameters can be adjusted to close them [24] [25].

³Landmark is an excellent feature point, observable from many frames.

1.4. Literature

“The term VO was introduced by Nister in his landmark paper [26], and he provided the first real-time and long-run VO implementation with a robust outlier rejection scheme, which is still widely used up to now” [27]. Solution proposed by Nister supported both mono and stereo vision, and used additional data from IMU. Real-time operation was accomplished on then-popular 1 GHz Pentium III processor.

Many papers focus alone on the problem of detection of independently moving objects (visual odometry outliers). In [28] such objects are identified using particle filter and probabilistic approach. More focus is given in [29] to a related, but more difficult task – estimation of individual objects velocities.

In [30] a neural network-based classifier is utilized for separating the road from rest of objects registered. Then optical flow of the road is calculated. This flow has a special mathematical structure [31], making it easier to infer the camera egomotion from it.

Well known and efficient feature detector is described in [32]. Features that are especially good from the SLAM point of view are introduced in [33]. Classic approaches monitor only temporal properties (i.e. how they appear in a single image). However, spatial properties (i.e. how features appear in the 3D space) should be taken into account in systems that preserve feature history.

A comparison of the most popular feature detectors is presented in [27]. Moreover, a consistency check step for feature matching is proposed, ensuring that features are best matched not only between frames n and $n + 1$, but also in the reverse direction.

In [34] a novel approach to monocular systems scale ambiguity proposed: by assuming a known, constant camera height above ground plane, global scale can be estimated. This step is performed between all consecutive frames, independently from main visual odometry algorithm.

Similarly in [35], scale is estimated on the basis of known camera height and pitch angle (downward tilt). Moreover, this is a full system that estimates camera velocity using optical flow. As long as speed does not exceed a couple of meters per second, only 20% coverage between consecutive frames is needed.

Other approach to scale estimation is presented in [36], where a ground-extraction scheme and modified Kalman filter is used.

The celebrated MonoSLAM method is described in [25]. It has introduced a shift to the paradigm of visual odometry – “*from mobile robotics to the 'pure vision' domain of a single uncontrolled camera*“. Instead of full map, only 100 best feature points are remembered. They are used for loop closure, that remedies the position drift. MonoSLAM achieves real-time on a low-end 1.6 GHz Pentium M processor.

Authors of [37] aim to fully reconstruct the 3D scene by using bundle adjustment. Method is proved to be both fast and robust.

Quite different approach to visual inertial odometry is presented in [38] – instead of feature points, mutual information between visual and inertial input is used to determine camera motion.

A dense method is proposed in [39]. It operates not on features, but on all of image pixels, making it somewhat more robust on the one hand, but inadequate for mobile devices on the other. A semi-dense method presented in [40] uses probabilistic inverse depth maps. Real-time is achieved, but only on a quad-core i7 processor.

1.5. The *Rebvo* algorithm outline

In this section the *Rebvo* algorithm description is briefly paraphrased for the needs of this thesis.

Tarrío and Pedre present new-fashioned approach to monocular visual odometry in [1]. The algorithm is reported to be capable of running in real time on an ARM processor. This sets it apart from numerous other state-of-the-art methods whose authors claim to achieve real time, but do not stress out that it is only possible on high-end PCs (Personal Computers).

It is similar to semi-dense methods, that achieve VO using only selected features (in this case – edges). It is not a full SLAM system, so only two consecutive frames are stored at each time and no global map is created. Information from previous frames is retained in the form of estimated depth. Features are matched on pixel basis. This concept is compliant with argument made by Harris: “*Because of ill-conditioning, the use of parametrized curves (e.g. circular arcs) cannot be expected to provide the solution, especially with real imagery.*“ [41]. Similarly

to many other state-of-the-art systems, depth information is stored as its inverse. It is worth mentioning, however, that inverse depth is most useful in full SLAM systems, where the “ray from the first camera position from which the feature was observed” [42] can be stored along with that feature.

Algorithm consists of three main steps, similar to other visual odometry systems. First of all, **edge extraction** is performed, preferably with subpixel precision. Moreover, edge gradient is calculated for each pixel. Neighboring pixels are joined into connected⁴ edges, using gradient information.

Then **tracking** is performed. Edges from previous frames are first projected into 3D space using their already estimated depths. Then an iterative procedure (Levenberg-Marquardt algorithm) aims to find such 3D transformation that establishes consensus between frames (i.e. minimizes the geometric re-projection error). Projected points are rotated and transformed in 3D, then projected back onto image plane. Minimized cost function is essentially the sum of squared distances between back-projected edges from the previous frame and closest edges from the current frame. Actual cost function also takes into consideration gradient correspondence criteria. Obtained pairs of edge pixels do not constitute an exhaustive list of matches, considering that:

- transformation is not ideal,
- depth of pixels is only estimated,
- there is quantization noise,
- edges can be detected inconsistently between frames,
- even for undistorted images, some residual distortion noncompliant with pinhole camera model will be present [43],
- outliers can be present (e.g. objects moving with respect to the otherwise rigid scene).

Final step – **mapping** – associates matching edges between frames using obtained optimal 3D transformation. Due to aforementioned problems, after tracking an additional matching routine is needed for each edge pixel. Because depth of previous frame edges is estimated with some uncertainty, camera motion establishes a line segment defining the area where possible

⁴Connected in the sense that each edge has no gaps between neighboring pixels.

matches will be searched for. Once such candidate has been found, it is tested for gradient correspondence and, most importantly, for model consistency – deviation of position on the segment obtained from linear transformation equation can not exceed depth uncertainty. After matching, depth information is propagated for matched edge pixels from previous frame to the current, and is optionally regularized. Previous depth has to be reestimated (OZ axis velocity has to be taken into account). This is achieved using Extended Kalman Filter. Scale drift, inherent problem of pure visual odometry, can be then mitigated to some limited degree by diving estimated depths by frame shrinking factor.

Accuracy of results obtained in [1] is comparable with other state-of-the-art algorithms [44].

2. Analysis and improvements of *Rebvo* algorithm

This chapter contains thorough analysis of proposed algorithm. The algorithm is carefully explained, step by step, using various examples. Symbols used through the chapter are explained in Section 2.1. Short overview of all substeps is given in Section 2.2. Much attention is paid to edge detector, because it had to be implemented along with the main algorithm, as available ready-made edge detection methods did not meet algorithm requirements. Figures used throughout the chapter were generated using input images from TUM [45] and KITTI [46] datasets. Chapter is concluded with remark on trajectory postprocessing, used to estimate algorithm accuracy.

Note: considerations, modifications and improvements of the original algorithm are **emphasized**. The have also been collected in Chapter 4.

2.1. Notation (Keyline structure) and `main` loop

Pixels that contain subpixel edge positions are called Keylines and, after edge extraction, are stored as an array of structures defined in Table 2.1. When $(n + 1)$ -th frame of video input is being processed, only n -th and $(n + 1)$ -th Keyline arrays are available; $(n - 1)$ -th is discarded, as it is no longer needed. This is presented in Fig. 2.1. Fields of n -th Keyline array will be denoted with subscript $_p$ (for *previous*) and $_t$ (for *transformed*). Subscripts $_c$ (*current*) and $_r$ (*rotated*) will be associated with the $(n + 1)$ -th frame.

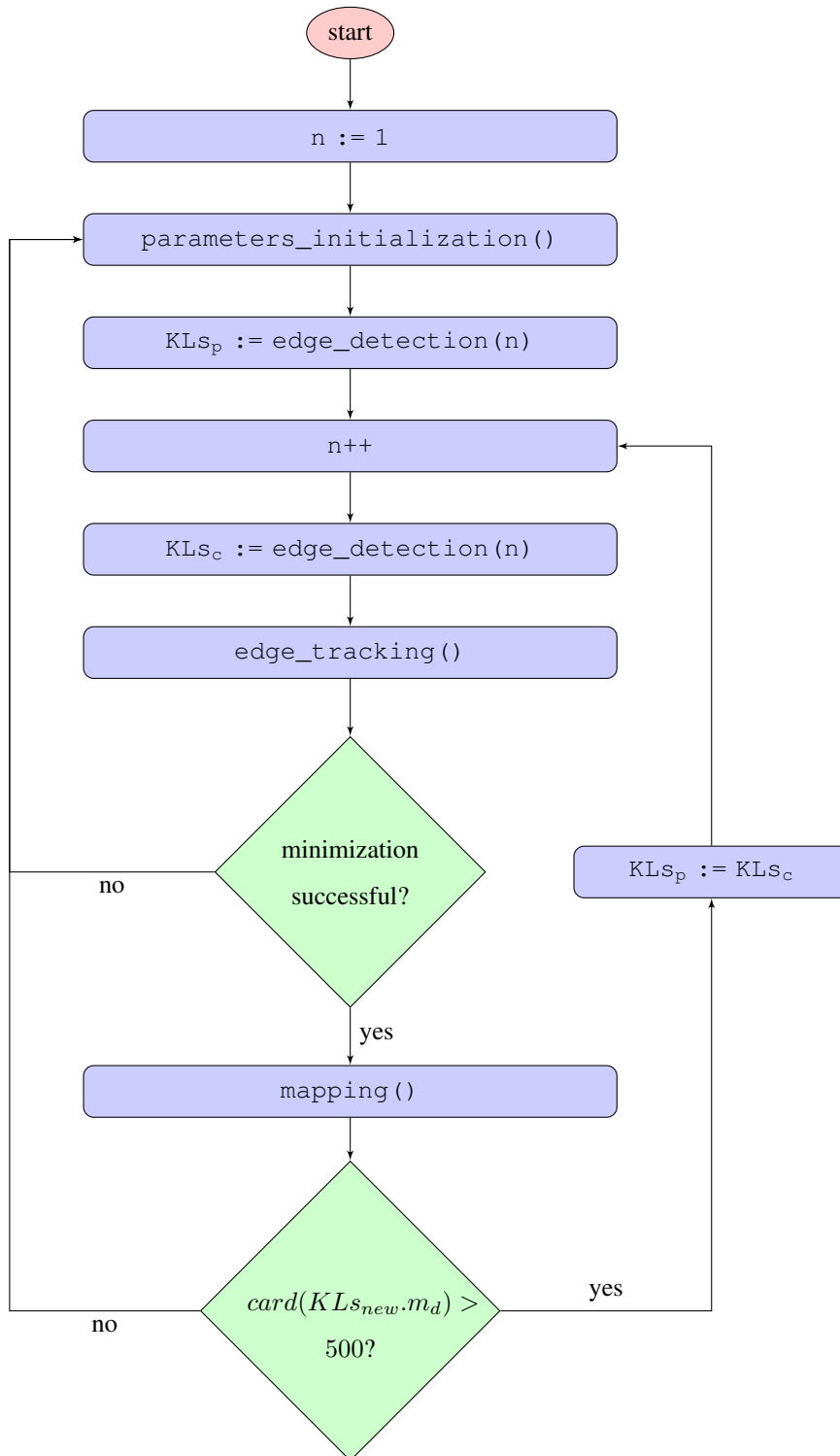


Fig. 2.1. Simplified flowchart of the algorithm. KLs_x are arrays of Keyline structures

Table 2.1. Keyline structure

Structure field	Description
$q = [q_x, q_y]^T$	subpixel position in image
$h = [h_x, h_y]^T$	normalized ^a q : $h = q - [c_x, c_y]^T$
ρ, σ_ρ	estimated inverse depth: $\frac{1}{Z}$, and its variance
ρ_0, σ_{ρ_0}	inverse depth predicted by Kalman filter and its variance
\vec{g}	edge gradient obtained from DoG
p_{id}, n_{id}	index of previous and next Keyline in an edge
m_f	index of next frame Keyline obtained during forward matching
m_d	index of next frame Keyline obtained during directed matching
k	number of consecutive frames that this Keyline has appeared on

^a Usage of normalized coordinates is more robust [47]. In the implementation, h is also often temporarily divided by f , so that calculations on homogeneous coordinates are more stable.

2.2. Algorithm outline

Explanation of the algorithm given in Section 1.5 is valid not only for [1], but also for the discussed implementation. Briefly summarizing:

1. Edge extraction – Section 2.3. A number of tests is performed to decide whether a pixel contains an edge – Section 2.3.3.
2. Edge tracking (minimization) – Section 2.4. Keylines from previous frame already had their depths estimated in previous iteration of the algorithm. By using pixel positions and depths, Keylines are projected from image plane to 3D space, where they transformed in order to “keep up“ with camera movement, and back-projected to image plane of the current frame – Section 2.4.1. They should coincide with new current frame Keylines that belong to same objects in the scene. Minimization is performed until this 3D transformation is optimal – Section 2.4.4.
3. Due to reasons mentioned in Sec. 2.5, in order to find matches between previous and current Keylines, additional matching has to be executed – Sec. 2.5.2. Depth information

is propagated from previous frame to current, smoothed out (Sec. 2.5.3) and updated to take into account the motion that camera has underwent (Sec. 2.5.4).

2.3. Edge extraction

Primal step of algorithm is subpixel edge extraction. Keyline structures are populated using extracted data (edge position and edge gradient). Keylines that are estimated to lie on same edge are joined.

2.3.1. Edge detection algorithm choice

While many edge detection algorithms could be used in this step, authors of *Rebvo* have chosen DoG (Difference of Gaussians), because it provides [1]:

1. repetivity – an edge is detected similarly throughout consecutive frames,
2. precision – edge positions are accurate,
3. low time & memory complexity.

Another advantage of DoG, unmentioned by Tarrio and Pedre, is that **edge gradient can be obtained directly** from normal vector of the fitted local plane. In this thesis DoG was used as well. Most vital property was subpixel precision, otherwise Canny detector [48] would be employed. In principle, subpixel edge detection precision is possible, because as long as Whittaker–Nyquist–Kotelnikov–Shannon theorem assumptions are satisfied, the true continuous image intensity function can be reconstructed from discrete pixel values.

2.3.2. Data preprocessing

First of all, RGB images are converted to grayscale. Color information is not used in the algorithm.

Before performing any edge detection technique, one critical issue has to be addressed. If input images were rectified¹ in such a way that extrapolation beyond original borders had been needed, artificial edges will be present in every frame (because extrapolation procedure usually

¹If images were *not* rectified, they should be.

assumes 0 pixel intensity beyond the image). Gradient of these edges is almost always strong, meaning that they tend to pass all tests and to be identified as valid keylines, thus generating false positives. During later minimization step, they greatly distort the procedure, making it behave as if the 3D space was curved. Example of such border is depicted in Fig. 2.2.



Fig. 2.2. Input image, already rectified by dataset authors, zoomed-in near left border. Individual pixels produced by rectification can be observed on the right

In case of DoG, these **artificial edges need to be removed** before Gaussian blurring – otherwise they would spread out, making it tricky to reject them later. In tested datasets, artificial borders' thickness did not exceed 1 pixel, so simply 1-pixel wide frame of outermost pixels was always discarded. In general case, **width of the frame can be figured out from distortion model**, instead of being set manually. However, for highly distorted images, a rectangular frame would also discard many valid pixels – either in corners (barrel distortions), or near middle of borders (pincushion distortions).

2.3.3. Difference of Gaussians strength tests

After necessary preprocessing, edge detection can be started. Generally edge detection works by finding positions in image where pixel intensity changes most rapidly. Basic idea of DoG is to approximate image second derivative (the Laplacian) [19] [49]. The zero-crossing

of Laplacian corresponds to such positions. Example based on real data, reduced to one dimension for clarity, is presented on Fig. 2.3. Other approaches to edge detection involve curve fitting [50] [51].

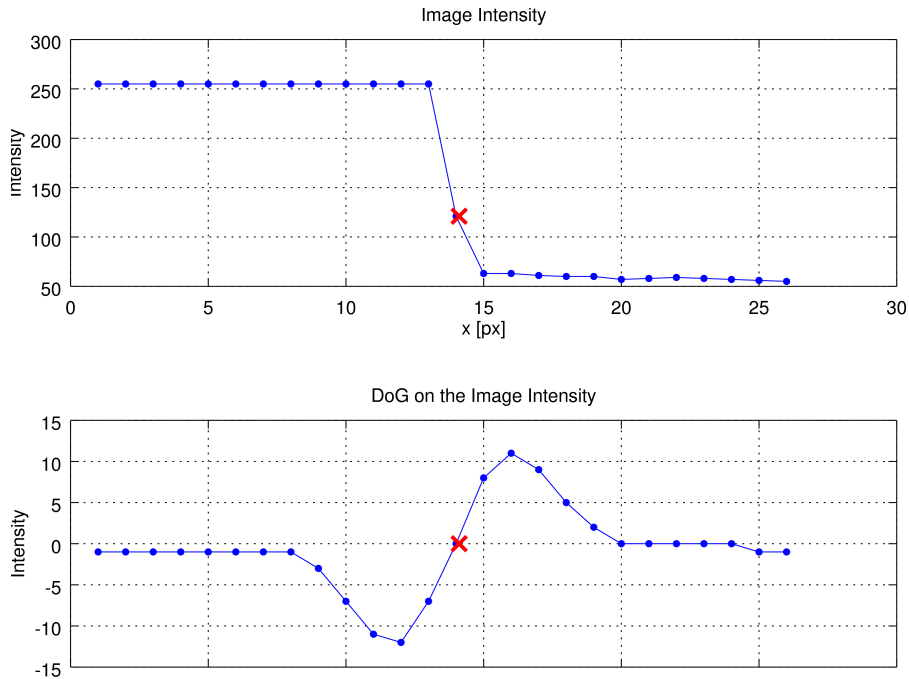


Fig. 2.3. DoG zero-crossing subpixel edge detection. Discussed method was applied to an image, then one row of pixels containing very a apparent edge was extracted. Red cross denotes calculated subpixel edge position

In order to filter out noise, image should be first smoothed with a Gaussian filter. These operations can be combined into one operator – Laplacian of Gaussian – which in turn can be approximated with difference of two images smoothed with Gaussian filters using two different sigmas² Edge detection results for initial sigma values were satisfactory in performed tests, therefore other sigma values (or automated sigma adjustment schemes) were not tested.

Exemplary Difference of Gaussians image is depicted in Fig. 2.4. *Implementation insight:* while individual smoothed images can be computed using `uint8` as underlying data type (for speed), the difference has to be computed using signed data type. Otherwise obtained function will not cross zero!

²LoG is best approximated by DoG when $\frac{\sigma_1}{\sigma_2} = \sqrt{2}$ [52].



Fig. 2.4. Difference of Gaussians applied to an image from the TUM dataset [45]. Near-zero values indicate feasible edge candidates

Another parameter is window size – it defines pixel neighborhood that will take part in later calculations of edge position. For window size w , $(2w + 1)^2$ immediate neighbors will be considered, including the center pixel itself. For sake of this thesis, value $w = 2$ was used.

Edge detection procedure is performed for every pixel that lies at least w pixels from image border, so that neighborhood does not need to be extrapolated. The procedure consists of five subsequent tests. If a pixel passes all of them, it is considered to be an edge (and additionally its subpixel position and edge gradient are calculated along the way). Influence of these tests on final result is depicted for an exemplary frame in Fig. 2.5. Color map explanation:

- dark blue (0 in the color map) – neighborhood outside image,
- blue (1 in the color map) – rejected by the first derivative norm test,
- light blue (2 in the color map) – rejected by DoG sign balance test,
- cyan (3 in the color map) – rejected by well-conditioning test,
- green (4 in the color map) – rejected by zero-crossing position test,
- orange (5 in the color map) – rejected by third derivative norm test,
- red (6 in the color map) – reserved for debug purposes,
- dark red (7 in the color map) – identified as a Keyline.

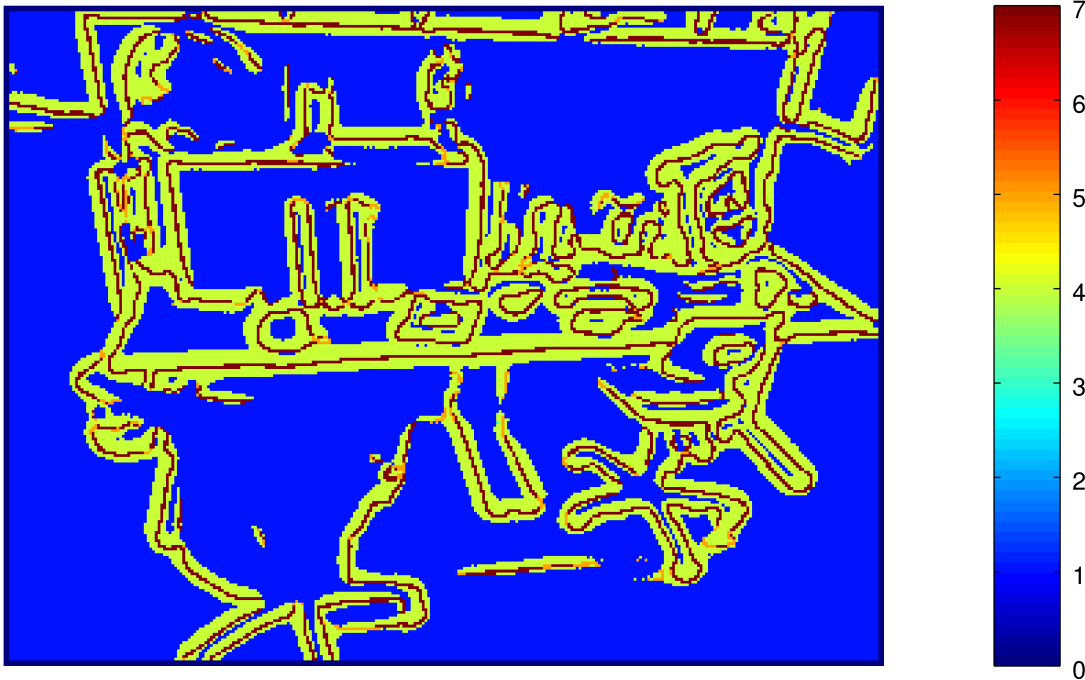


Fig. 2.5. Edge detection tests results. Pixel colors indicate which test has rejected given pixel (or if it has been identified as edge)

First test – first derivative norm

In order to speed up computations by early identifying non-edges, first derivative of pixel intensity is calculated by applying two Sobel operators (derivatives along x and y axes) and taking norm of the result. This norm is then compared with a threshold.

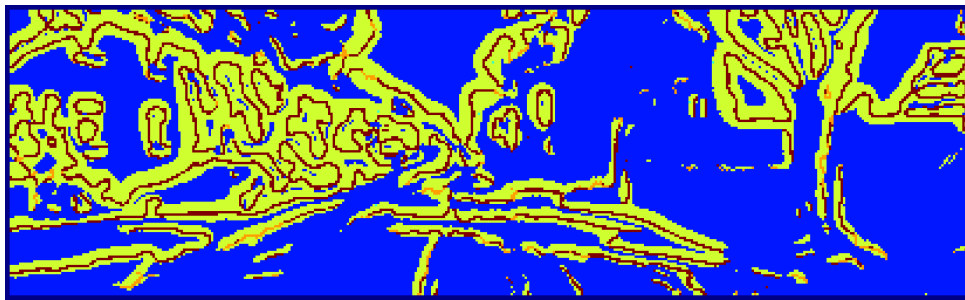
Tarrío and Pedre use hysteresis to determine threshold value after every frame. On the one hand, presented algorithm operates offline, so abundance of keylines is not an issue. On the other, hysteresis parameters still need to be adjusted for given dataset. Invalid parameters can alter overall algorithm results significantly, so from algorithm analysis point of view, the fewer of them, the better. **Thus instead simpler solution was tested out.**

Image is partitioned into a^2 rectangular chunks, for relatively small a , e.g. 7. Then for each chunk number of pixels that would pass the first derivative test for given constant threshold is counted. If this number is relatively low, threshold is locally multiplied by a constant $b_1 < 1$, e.g. $\frac{1}{3}$. However, if the number is relatively large, then threshold is locally multiplied by $b_2 > 1$, e.g. $\frac{5}{3}$. Local threshold array can finally be smoothed with Gaussian filter to avoid discontinuities.

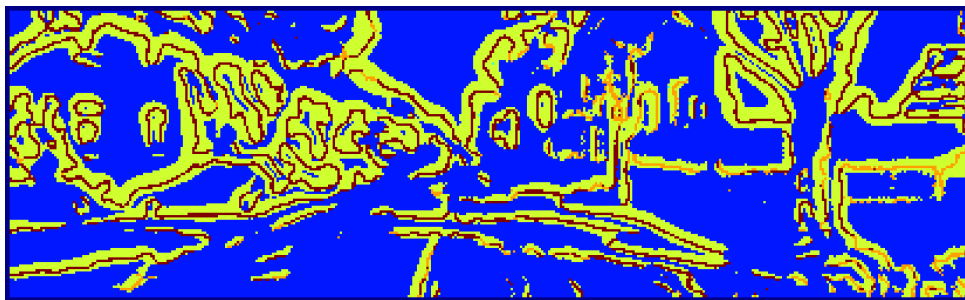
Proposed procedure also has parameters, but there are few advantages:

- parameters are much simpler to interpret,
- effectively only one parameter is crucial (the constant threshold) and rest of procedure serves as a refinement,
- operates without delay, which is vital especially if keyline number is too low.

In Fig. 2.6 some pixels on the right half of the image were considered further for being edges, although ultimately they were also rejected. On the other hand, more non-edge pixels have been rejected by the first test on the left half (green regions are a bit thinner). However, if any edges *are* added by this method, they are not very stable and tend to be nonetheless untraceable in later frames (they do not exhibit the *repetivity* property).



(a)



(b)

Fig. 2.6. Edge detection tests results. Pixel colors: dark blue – neighborhood outside image; blue, light blue (not present), cyan (not present), green and orange – rejected by tests 1 to 5, respectively; red – reserved for debug purposes (not present); dark red – final Keylines. (a) Constant first test threshold, (b) Chunked first test threshold

Second test – DoG sign balance

Neighborhood of pixels that have passed the first test is checked for “sign balance“. Number of positive and negative DoG values has to be comparable within a defined percentage, e.g. 20%. Example results of the test for 2 different pixels are depicted in Fig. 2.7. It was observed that pixels very rarely fail this particular test.

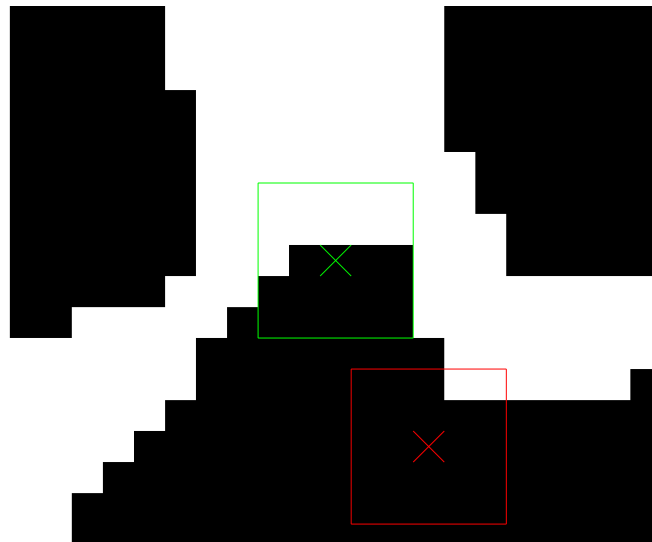


Fig. 2.7. Example of the DoG sign balance test. Negative values are denoted as black pixels, positive – as white. Green color marks neighborhood of pixel that has passed the test and is, possibly, an edge. Pixel marked with red was rejected

Third test – well-conditioning

Then DoG values are approximated by a plane using linear regression – (2.1) is solved for θ . Pseudo-inverse of \mathbf{A} , needed to solve it, has to be computed only once for the whole algorithm. Zero-crossing of the plane can be determined algebraically, resulting in (2.2). This has been visualized in Fig. 2.8. These closed-form formulas are the main reason for DoG not being computationally demanding, as curve-fitting approaches are usually iterative.

For more resilience, an additional test is performed. If (2.3) is not satisfied, then equation system 2.1 **is considered to be badly conditioned** and this pixel is rejected. This was not considered in [1]. The purpose of this test is to avoid divide-by-zero errors in fourth test (Section 2.3.3). All pixels that do not pass the third test would still be filtered out later – they also do not pass the fifth test (Section 2.3.3).

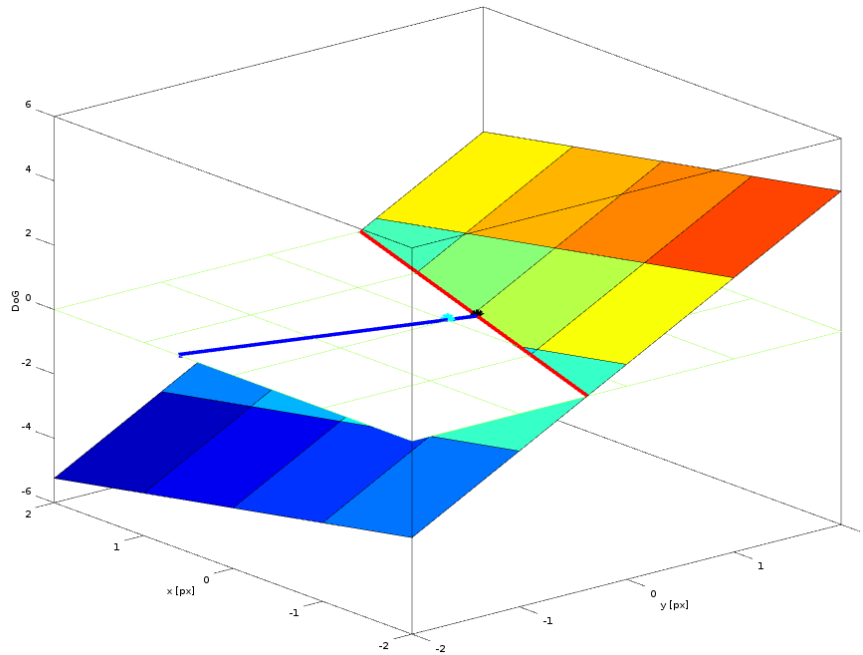


Fig. 2.8. Geometric interpretation of zero-crossing search. Local approximation of DoG values has been depicted with a colored plane. It intersects the $z = 0$ plane, creating the red line. Blue line also lies on the $z = 0$ plane; it passes through the origin (cyan point) and is perpendicular to red line. Intersection of these two lines (black point) is the zero crossing

$$\mathbf{A}\theta = \delta \quad (2.1)$$

where:

\mathbf{A} – $3 \times (2w + 1)$ matrix of pixel positions: $[X \ Y \ 1]$,

X – column vector of x coordinates of pixel centroids in neighborhood, assuming that central pixel's centroid is located at $x = 0$, $((2w + 1)$ -vector),

Y – column vector of y coordinates of pixel centroids in neighborhood, assuming that central pixel's centroid is located at $y = 0$, $((2w + 1)$ -vector),

δ – DoG values corresponding to X and Y $((2w + 1)$ -vector),

θ – parameters defining the approximated plane: $z = \theta_x x + \theta_y y + \theta_z$.

$$[x_s, y_s]^T = \left[\frac{-\theta_x \theta_y}{\theta_x^2 + \theta_y^2}, \frac{-\theta_y \theta_x}{\theta_x^2 + \theta_y^2} \right]^T \quad (2.2)$$

where:

k_s – estimated k -coordinate of the subpixel edge position (0 is the pixel center).

$$\theta_x^2 + \theta_y^2 > 10^{-6} \quad (2.3)$$

Fourth test – zero-crossing position

Obtained zero-crossing marks the subpixel position of the edge; an example of a few subpixel positions along an edge is presented in Fig. 2.9. Inequality (2.4) tests whether it lies within the pixel itself. If not – edge is not detected. After this test, most of Keyline candidates form 1-pixel wide edges.

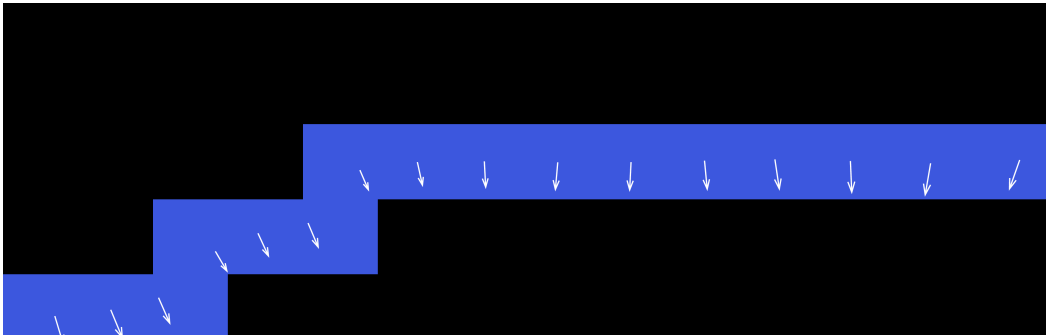


Fig. 2.9. Obtained zero-crossings of a particularly well detected edge. Black pixels denote background (non-Keylines) and blue – Keylines. The blue line is a 1-pixel wide edge. White vectors represent edge gradient. Initial points of vectors are the subpixel zero-crossings

$$\max(|x_s|, |y_s|) < 0.5 \quad (2.4)$$

Fifth test – third derivative norm

Normal vector of fitted plane is defined by (2.5). Vector \vec{r} can be projected onto $z = 0$ plane, creating third derivative vector: the edge gradient \vec{g} . If $\|\vec{g}\|$ is small, then r_z component must have dominated \vec{r} , meaning that fitted plane was almost parallel to the $z = 0$ plane. In turn this

implies that edge was not sharp. Therefore, as a final test, norm of \vec{g} is tested – it must exceed a threshold for edge to be finally detected.

$$\vec{r} = [\theta_x, \theta_y, -1]^T \quad (2.5)$$

where:

\vec{r} – normal vector of the fitted plane $z = \theta_x x + \theta_y y + \theta_z$.

Edge detection tests summary

Colored regions visible in Fig. 2.5 do resemble to some degree a probability distribution – the probability that given pixel contains an edge. Thus initially **use of fuzzy logic was considered**. Ultimately, as already mentioned, detected edges were deemed to be acceptable, so this idea was not pursued. Such approach would also require more processing power, making it less suitable for mobile devices.

Presented edge detection algorithm has proved to be quite robust under lighting conditions changes, as reported in [1]. An example is depicted in Fig. 2.10 (input images) and Fig. 2.11 (map of detected edges).

2.3.4. Depth initialization

After Keyline has been successfully identified on image, a data structure described in Section 2.1 is populated with obtained data. This step is different for the very first processed frame, because its Keylines must have some initial depth values. During first tests, a constant $\rho = 1$ was used. Then for some time explicit Kinect depth map was used, as it was available in the TUM dataset. Finally, after other ares of algorithm have been improved, it was observed that in most cases quick convergence (5-10 frames) could be achieved by **initializing depth with random values**, even with images scaled down by 80% (to 128 by 96 pixels). Convergence depends on speed of camera during these first frames. Overall, this is much better result than in [1], where 2-5 seconds needed for depth convergence were reported. Noise was generated using normal distribution with parameters chosen for each dataset: $N(2, 0.5)$ for TUM and $N(10, 3)$ for KITTI.



(a)



(b)

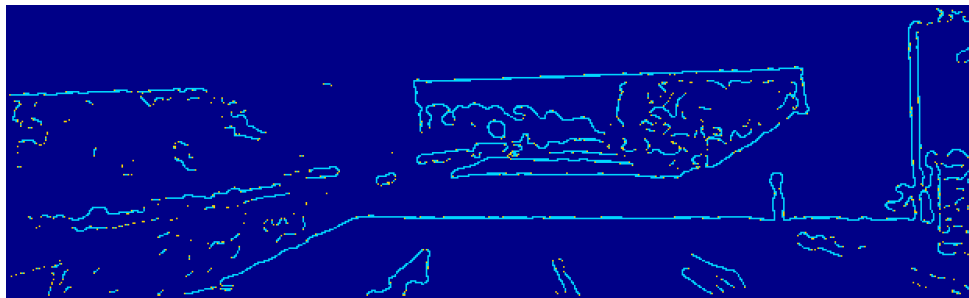


(c)

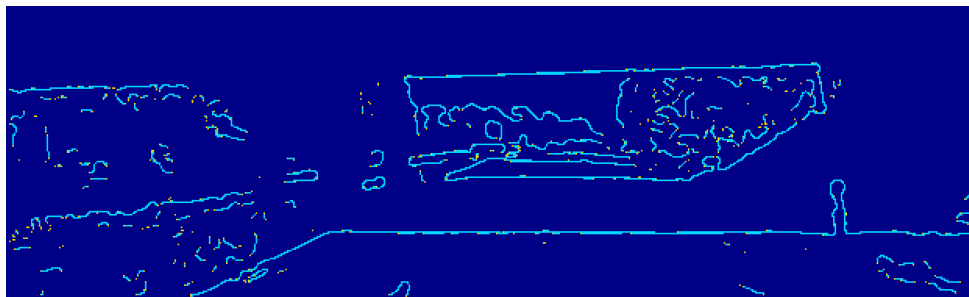


(d)

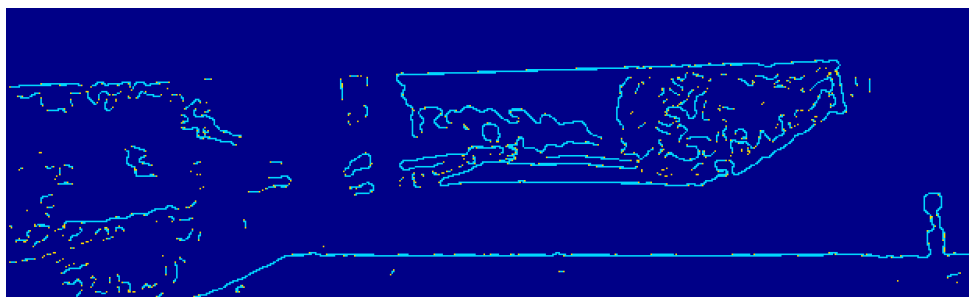
Fig. 2.10. Edge detection in 4 consecutive KITTI [46] frames under lighting conditions changes – input images



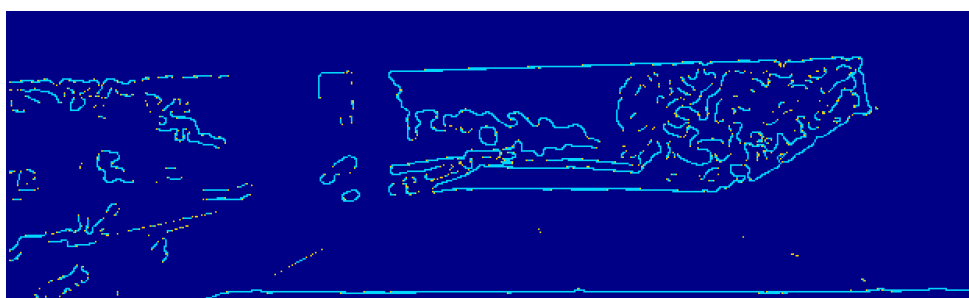
(a)



(b)



(c)



(d)

Fig. 2.11. Edge detection in 4 consecutive KITTI [46] frames under lighting conditions changes – input images. Detected edges are denoted with cyan

Some thought was given to more elaborate initialization schemes. Feature point correspondences could be used in order to determine first frames scene geometry [40] in a quicker and more robust way, while rest of algorithm would still operate on edges. Feature point descriptors, such as the SIFT (Scale Invariant Feature Transform) [52], or even simpler corner detectors [41] [32], are well suited for this task.

Their main disadvantage is their complexity, especially in case of more robust ones, like SIFT. They need to be somehow matched – this becomes unfeasible for real-time applications once number feature points is too large [5]. This in turn means that many parameters would have to be carefully chosen so that a consumer-grade mobile system would not be flooded by excessive features and remain responsive. Finally, feature point descriptors encode larger pixel neighborhood than edges, but they lack structural information, meaning that match is likely produce more outliers, that need to be dealt with (filtered).

2.3.5. Keyline joining

After obtaining individual Keylines, they are joined together to form connected edges. For each Keyline, its neighbors are searched among 3 out of 8 bordering pixels. Search is performed in direction perpendicular to \vec{g} , an example is depicted in Fig. 2.12.

Joined Keylines are used for pruning. First of all, edges consisting of 3 pixels or less are discarded, as they are unlikely to be good features to track. Secondly, outermost Keylines of every joined edge are likewise removed – out of all Keylines in that edge, they are most likely to be affected by noise. Pruned Keylines can be seen in Fig. 2.13, as well as in Fig. 2.11.

In later steps, the algorithm considers only individual Keylines, not joined edges (as mentioned in Section 1.5). Information about neighbors is used after edge detection only once – in Regularization step, where ρ and σ_ρ are averaged over Keyline and its 2 immediate neighbors.

Initially **more edge joining strategies had been considered**: morphological operations on Keylines, loop avoidance and edge segmentation (into parts with similar gradient). However, once it was understood that edge joining takes such small part in algorithm flow and that by-pixel approach is preferable [41], this direction of research was abandoned.

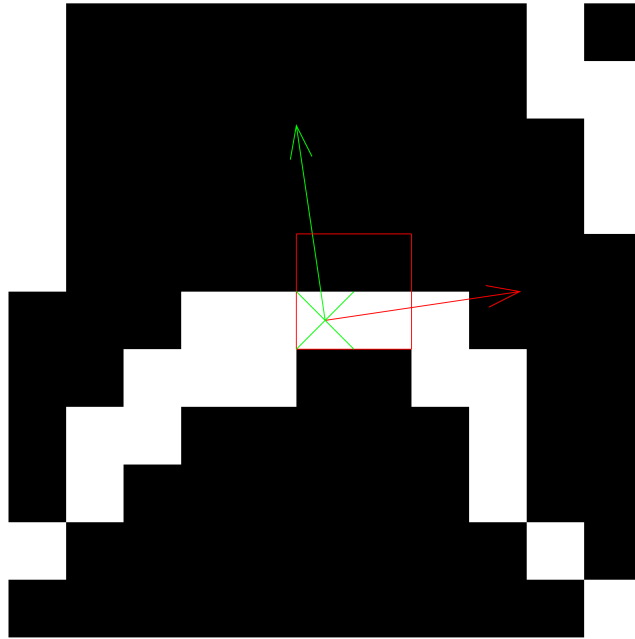


Fig. 2.12. Edge joining principle. White pixels denote Keylines, black pixels – non-Keylines, green x – currently processed pixel, green arrow – \vec{g} , red arrow – \vec{g} rotated by $\frac{\pi}{2}$ clockwise, red rectangle – edge joining candidates

2.4. Edge tracking

Goal of edge tracking is to find a 3D transformation (rototranslation – translation and rotation) that best describes transition between two consecutive frames. Previous frame Keylines are projected to 3D space, where they are rototranslated; then resulting points are back-projected to image plane (transformations are elaborated upon in Section 2.4.1). Reprojected Keylines are matched against Keylines of the next frame – see Sections 2.4.2 and 2.4.3. Minimization of the residual is performed using Levenberg-Marquardt algorithm. Similarly to [1], implementation is based on [53].

2.4.1. Warping function

2D Keyline \longleftrightarrow 3D point transformations are defined by (2.6) and (2.7). They can be derived from 1.2, assuming $f_x = f_y = f$ and $s = 0$. These assumptions were valid for tested datasets. It should be noted, however, that in case of smartphone cameras, **full pinhole model should be considered**, complicating these formulas. Once points have been projected

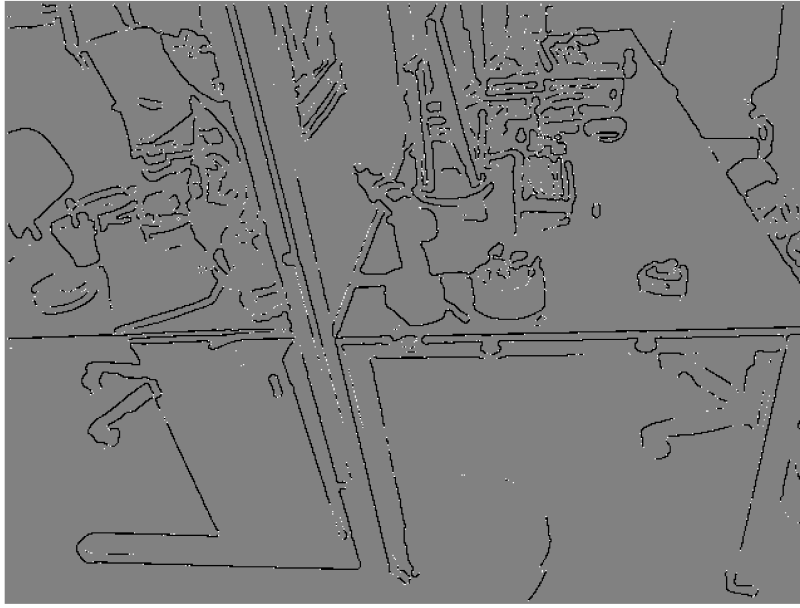


Fig. 2.13. Edge pruning result. Gray pixels – background; black – valid Key-lines; white – pruned Keylines

using γ , they can be rotated and translated in 3D space using (2.8). The rotation matrix has form $\mathbf{R} = \exp([\vec{\omega}]_s)$.

$$\gamma(h_p, \rho_p) = \left[\begin{array}{c} h_{px}, \quad h_{py}, \quad 1 \\ f\rho_p, \quad f\rho_p, \quad \rho_p \end{array} \right]^T : \mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}^3 \quad (2.6)$$

where:

γ – projection function.

$$\gamma^{-1}(Q) = \left[\left[\begin{array}{c} fQ_x \\ Q_z \end{array} \right]^T, \left[\begin{array}{c} fQ_y \\ Q_z \end{array} \right]^T, \frac{1}{Q_z} \right]^T : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \times \mathbb{R} \quad (2.7)$$

where:

γ^{-1} – back-projection function,

Q – 3D world point (3-vector).

$$\psi(\vec{v}, \vec{\omega}, Q) = \exp([\vec{\omega}]_s) Q + \vec{v} \quad (2.8)$$

where:

ψ – warping function,

\vec{v} – camera translation (3-vector),

$\vec{\omega}$ – camera azimuth rotation in axis-angle notation³ (3-vector),

\exp – matrix exponentiation function, which can be approximated using Euler-Rodrigues formula [11].

2.4.2. Auxiliary image

Auxiliary image is a lookup table that speeds up minimization. It is a matrix of size equal to the image. An entry in the matrix is nonzero if distance from its coordinates to closest Keyline's q is lower than a search radius r_{search} , usually equal to 5 pixels. Then such entry contains a reference to the found closest Keyline.

During minimization, Keylines from previous frame are projected to 3D space using depth information (ρ), rototranslated, and then projected back to image plane. Positions obtained from the back-projection are compared against the auxiliary image of next frame to check if such position corresponds to a Keyline (and to which).

Auxiliary image creation can be perceived as widening of an edge by “spanning“ its pixels along \vec{g} . A depiction is presented in Fig. 2.14a. Because pixel positions are discrete, **increments of this spanning should be lower than 1 px**. Otherwise, when \vec{g} is not perpendicular to pixel grid, edge widening process will occasionally skip over some pixels. This leads to coarseness and discontinuities (gaps) in auxiliary image (see Fig. 2.14b), which can later bias the minimization by producing false local minima.

Some gaps are unavoidable, however, because \vec{g} is available only in places where there is a Keyline. Even with subpixel position precision, there still is at most only one Keyline per pixel. If edge has any curvature, then at some radius auxiliary image will feature spikes. They are visible in both Fig. 2.14a and Fig. 2.14b.

³Axis-angle vector $\vec{\omega}$ describes right-hand rotation by $\|\vec{\omega}\|$ radians about $O\omega$ axis [4] [54].

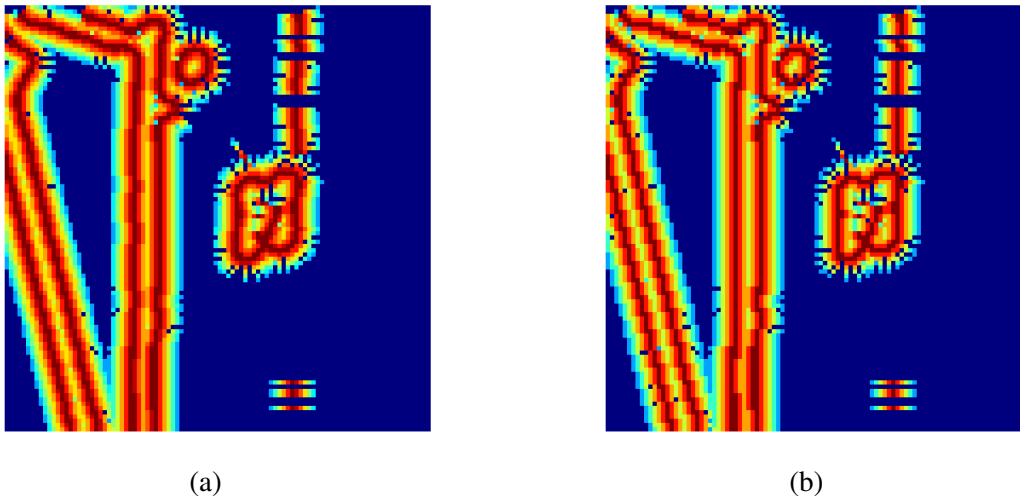


Fig. 2.14. The auxiliary images created with different step sizes. Pixel color encodes distance to nearest Keyline (the warmer, the closer). (a) step 0.5, (b) step 1

2.4.3. Keyline matching criteria

During each iteration of minimization, reprojected old Keylines and new Keylines are pre-matched, using auxiliary image. In order to determine validity and score of this match, following criteria are tested. Result of applying them on a frame is depicted in Fig. 2.15.

1. History – if k_p is lower than some constant threshold k_{thresh} (e.g. 2), then this Keyline is considered to be too uncertain (its history is too short) and it does not take part in minimization at all. This test is performed only after k_{thresh} frames have been wholly processed by the algorithm in order to allow system to initialize.
2. Depth uncertainty – Keylines where σ_{ρ_p} exceeds its 95. percentile are rejected. This parameter is estimated by Kalman filter during the mapping step.
3. Gradient similarity – score based on angle between \vec{g}_p and \vec{g}_c , as well as on their magnitudes, can not exceed a threshold.

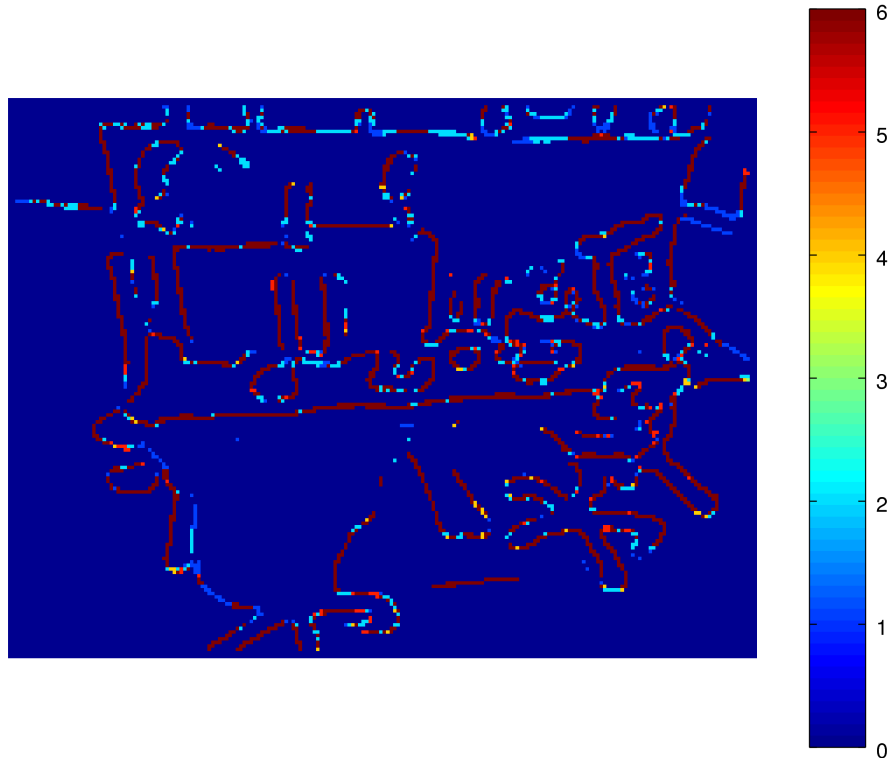


Fig. 2.15. Minimization tests results. Pixel colors: dark blue – not Keyline; blue – σ_ρ too high; cyan – too short history; teal – 2D reprojection lies outside the image; orange – no corresponding Keyline in next frame’s auxiliary image; red – gradients dissimilar; dark red – successful matching

2.4.4. Energy minimization

Each minimization iteration executes following steps:

- Using previously estimated Jacobian matrix \mathbf{J} and other information from previous iteration, propose new parameter vector in the 6-dimensional parameter space $[\vec{v}, \vec{\omega}]$.
- For each Keyline in previous frame:
 - Perform reprojection: $h_t = (\gamma^{-1} \circ \psi \circ \gamma)(h_p, \rho_p)$.
 - Find closest h_c that conforms to similarity criteria.
 - Calculate the residual.
- Calculate overall iteration score.

Minimized function takes form defined in (2.9). Distance between matched points is casted onto edge gradient, because “*location information is only present in direction [of the gradient]*” [1]. Sample residuals are depicted in Fig. 2.16 and Fig. 2.17. The matching function M is not differentiable, making it impossible to calculate Jacobian of ζ analytically. Therefore Jacobians were calculated using the same formulas as in Tarrío and Pedre’s implementation.

$$E = \zeta(\vec{v}, \vec{\omega}) = \sum_i^p \frac{w_i}{\sigma_{\rho_i}^2} \left[M_{h_t, h_c}((h_t) \bullet \vec{g}_c) \right]^2 \quad (2.9)$$

where:

w_i – weight (square of the Huber norm [55]),

h_t – transformed h_p , $h_t = \gamma^{-1}(\psi(\vec{v}, \vec{\omega}, \gamma(h_p, \rho_p)))$,

M – matching function, $M_{h_t}(\bullet) = \begin{cases} \bullet, & \exists h_c : h_t \text{ matches } h_c \\ r_{search}, & \text{otherwise} \end{cases}$.

Increment of the input vector is dictated by the Levenberg–Marquardt update equation [56]. One way of determining it is to use SVD on a 6x6 matrix Λ , as it is performed by Tarrío and Pedre. However, Λ is always a positive definite [53], meaning that **Cholesky factorization can be used**, instead of slower, but less constrictive SVD. However, size of the decomposed matrix is quite small, so computational gain is negligible.

This was tested in GNU Octave, using OpenCV [11] linear algebra routines, compiled to MEX files. Random values were chosen to populate variables in the LM equation and then it was solved 10^6 times, first using SVD, then Cholesky factorization. Average running times per iteration have been collected in Tab. 2.2.

Table 2.2. SVD and Cholesky running time comparison

Method	Average time
SVD	29 μ s
Cholesky	24 μ s

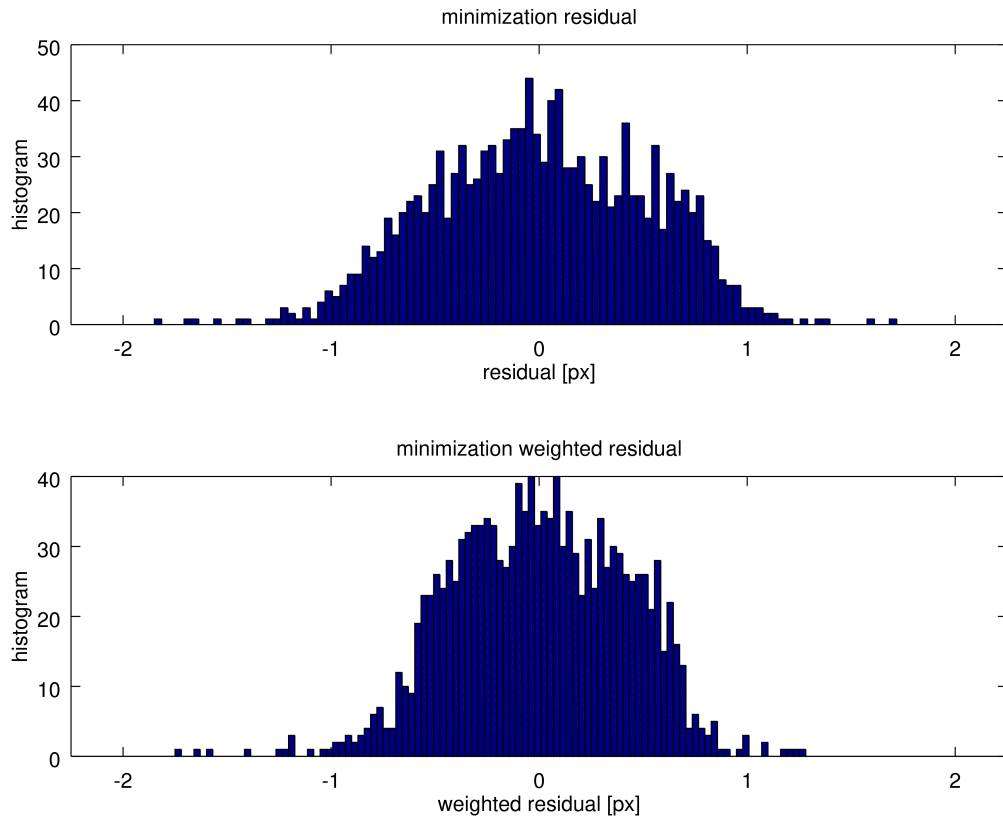


Fig. 2.16. Minimization residual after final iteration, with and without the weighting Huber norm. Negative values are present, because direction of displacement is taken into account

After optimal transformation parameters have been found, uncertainty of the result is estimated as covariance matrix in form $(\mathbf{J}^T \mathbf{J})^{-1}$. **If the matrix being converted is ill-conditioned, then it is assumed that minimization has failed.** For example, this can happen if no matches at all were found. Then all residuals will be equal and ζ will be completely flat locally. Usually minimization fails only if number of input Keylines is too low.

Moreover, after last minimization iteration, the m_{pf} field of previous Keylines is populated. For every Keyline that was successfully matched to a current Keyline, reference to this current Keyline is saved in the said field.

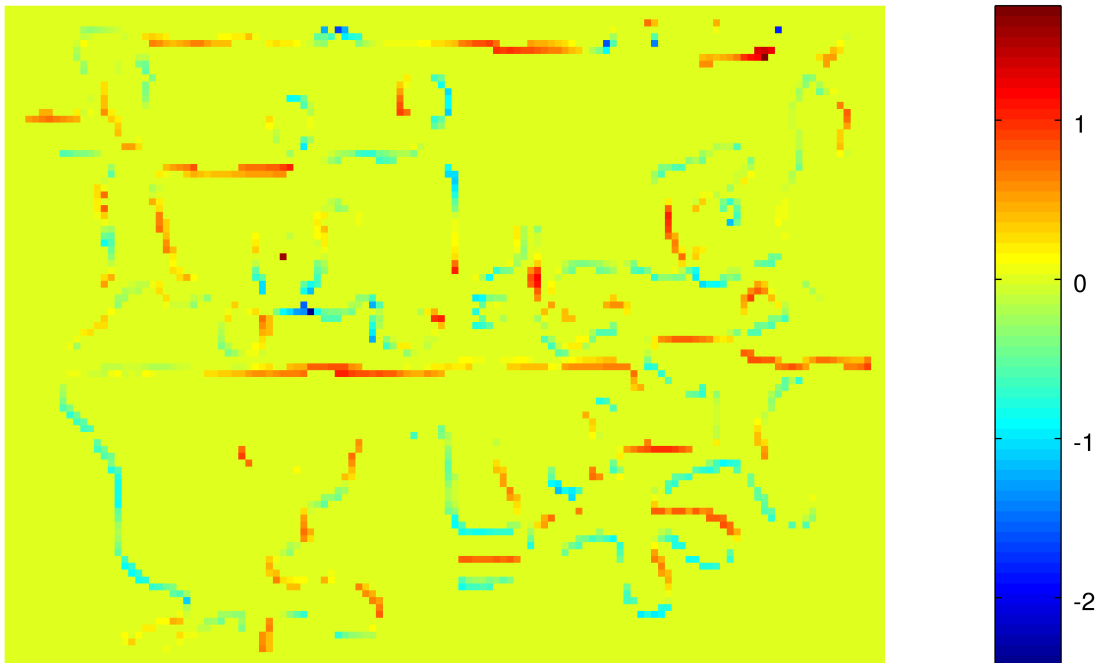


Fig. 2.17. Minimization residual after final iteration. Negative values are present, because direction of displacement is taken into account

2.4.5. Initial conditions

Levenberg-Marquardt algorithm is prone to falling to local minima. Repetitive patterns, such as fences, are especially challenging (see Fig. 2 in [57]). Generally, global minimization of arbitrary function is an intricate problem with no infallible solution. There has been proposed a plethora of heuristic methods: evolutionary algorithms, simulated annealing, physical simulations (Momentum), etc. However, they are unsuitable for real-time applications [19].

In case of LM, crucial issue is choice of initial condition vectors. Following [1], two initial conditions were tested, and calculations proceeded using better one (see Fig. 2.18). These vectors are: $\vec{0}$ and rototranslation of the previous frame. If linear and angular instantaneous velocities of the camera do not change rapidly, it is reasonable to assume that previous and current velocities are not far apart in the parameter space.

One exception is the special case when too few Keylines are matched and system decides to reinitialize. This usually happens when number of detected Keylines also drops – for instance

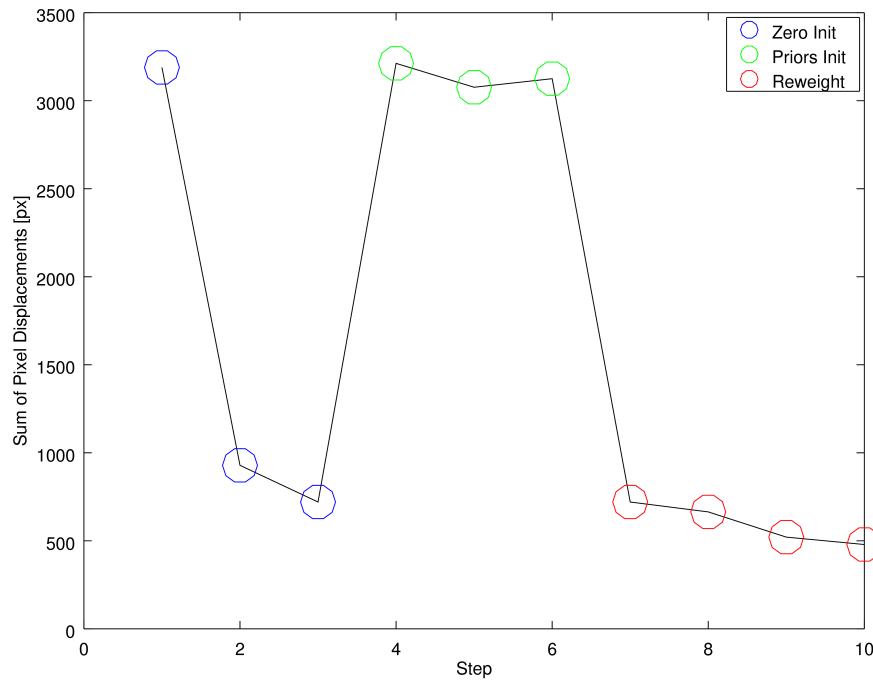


Fig. 2.18. Total energy minimization. Two initial conditions are tried, then better one is continued

in the `fr2_360_kidnap` sequence (TUM [45]). In the context of algorithm results postprocessing (i.e. fitting the estimated camera trajectory to ground truth data), it is better to assume no prior knowledge and overwrite the previous parameter vector with zeros.

During some tests, the prior rototranslation vector for the first frame in sequence was obtained from ground truth data in hope of accelerating algorithm convergence (and to assess whether it could retain good information that was fed to it). However, usually ground truth data uses other reference frame, as position of the laser positioning system is different from position of the camera itself.

2.5. Mapping

For obtained information to be useful in the next iteration of the algorithm, it needs to be preserved. Since this is not a full SLAM system, data needs to be updated and passed on to next array of Keylines. This is achieved by the mapping step: Keylines from previous iteration are matched to Keylines that are currently being processed. Data that is passed on between matched Keylines contains:

- estimated ρ and σ_ρ , that need to be refined with the Kalman filter,
- history (k),
- index of matching Keyline from the previous frame.

At this stage initial matching has already been performed during minimization. As it was noted in Section 1.5, it is quite coarse, however. In addition, there is another reason for such elaborate match searching scheme. After system has been initialized (i.e. after k_{thresh} frames have been processed), history test will become active in the minimizer. As far as the minimizer is concerned, only Keylines already having some matching history will have possibility of being matched again and propagated further. On the other hand, “new“ Keylines with no prior history will not have opportunity of gaining it, because they will be always skipped in the very first minimizer test and not matched. This is why additional matching is needed.

Finally, the mapping step contains depth information inter-frame processing procedures (regularization, Kalman filtering, scale correction).

2.5.1. Forward matching

Since calculated transformation between the two frames is supposed to be the optimal one, after applying it to 3D positions of previous Keylines and casting them into the image plane, their expected positions on the next frame will be obtained. This was already done at the minimization step; said Keylines were then paired with their corresponding match using property m_{pf} (see conclusion of Section 2.4.4). Forward matching simply uses this information and copies appropriate field values from previous Keylines to their new forward matched counterparts. If later a better match is found, already copied data will be simply overwritten. *Forward* refers to the fact that minimizer creates initial matches from previous frame to current.

2.5.2. Directed matching

Considering that there might have been some outliers that affected the quality calculated transformation and that potentially valid matches with no history need a chance to pick it up, it is essential to extend the list of matches to include Keylines that have not been back-projected perfectly.

So as to make finding candidates for matching easier, current Keyline array is reverse-rotated⁴ and casted onto the old edge map. This decreases the distance between corresponding points and reduces the problem to a pure translatory problem.

This approach can be compared to *mutual consistency check* in [27]. During minimization, previous Keylines were forward rotated onto current image plane and matching was performed. Now current Keylines are reverse rotated onto previous image plane in hopes of finding more matches.

Translation \vec{v} calculated during minimization, along with the reverse-rotated h_r , defines for each current Keyline a line. Were ρ_r known precisely, relationship between matching points would satisfy (2.10). As it is not, (2.10) defines a line.

$$h_p = h_r - \rho_r (f v_{r_{x,y}} - h_r v_{r_z}) \quad (2.10)$$

Because of assumption that $\rho_r > 0$, search line is reduced to a halfline. A possible match should lie on the halfline. In order to further constrain search area, maximum and minimum pixel displacements are estimated, leaving only a line segment (see Fig. 2.19). Keylines that lie within it are possible candidates for matches. *Directed* refers to probing along the direction of line segment.

Searching starts at a point that is estimated to be a most possible match. The procedure is performed checking pixels closer to segment ends, in an alternating manner. A segment of halfline If an inspected pixel a previous Keyline, it is considered a possible match, that still needs to be validated. Firstly, \vec{g}_p and \vec{g}_r are compared, analogously to the third keyline matching test of the minimizer (see Section 2.4.3). Secondly, potential match needs to conform to the motion model, defined in (2.11).

$$\left| \frac{\|h_p - h_r\|}{\|f v_{r_{x,y}} - h_r v_{r_z}\|} - \rho_r \right| < \tau(\sigma_{\rho_r}) \quad (2.11)$$

where:

\vec{v}_r – translation vector, $\vec{v}_r = (\exp([\vec{\omega}]_s))^T \vec{v}$,

τ – uncertainty estimation function that takes into account σ_{ρ_r} .

⁴Rotated using rotation opposite to rotation $\vec{\omega}$ that was obtained during minimization.



Fig. 2.19. Directed matching along halfline segment. Orange pixels denote current Keylines, rotated to previous frame by \mathbf{R}^T ; cyan – previous Keylines; dark red – position of a current and a previous Keyline. Green and red arrows span over the line segment

This provides additional outlier rejection scheme – any match that is not compatible with model is rejected. A Keyline that moves in 3D differently than rest of the scene most likely belongs to edge of an object that moves independently, as alluded in Section 1.5. This is particularly visible on TUM fr2_desk_with_person dataset, where objects present in the scene were shuffled around. As it can be seen in Fig. 2.20, Keylines belonging to a moving person have their history repeatedly reset. Outliers are filtered even if camera itself moves – as long as their motion is not consistent with the scene.

After forward matching and directed matching, the number of valid matches must exceed previously defined threshold. Threshold value of 500 Keylines has proved to be a good figure during tests, regardless of image scale. If threshold is not met, algorithm resets, as there is not enough Keylines with established history to base future transformation calculations on. A valid matching results is depicted in Fig. 2.21.

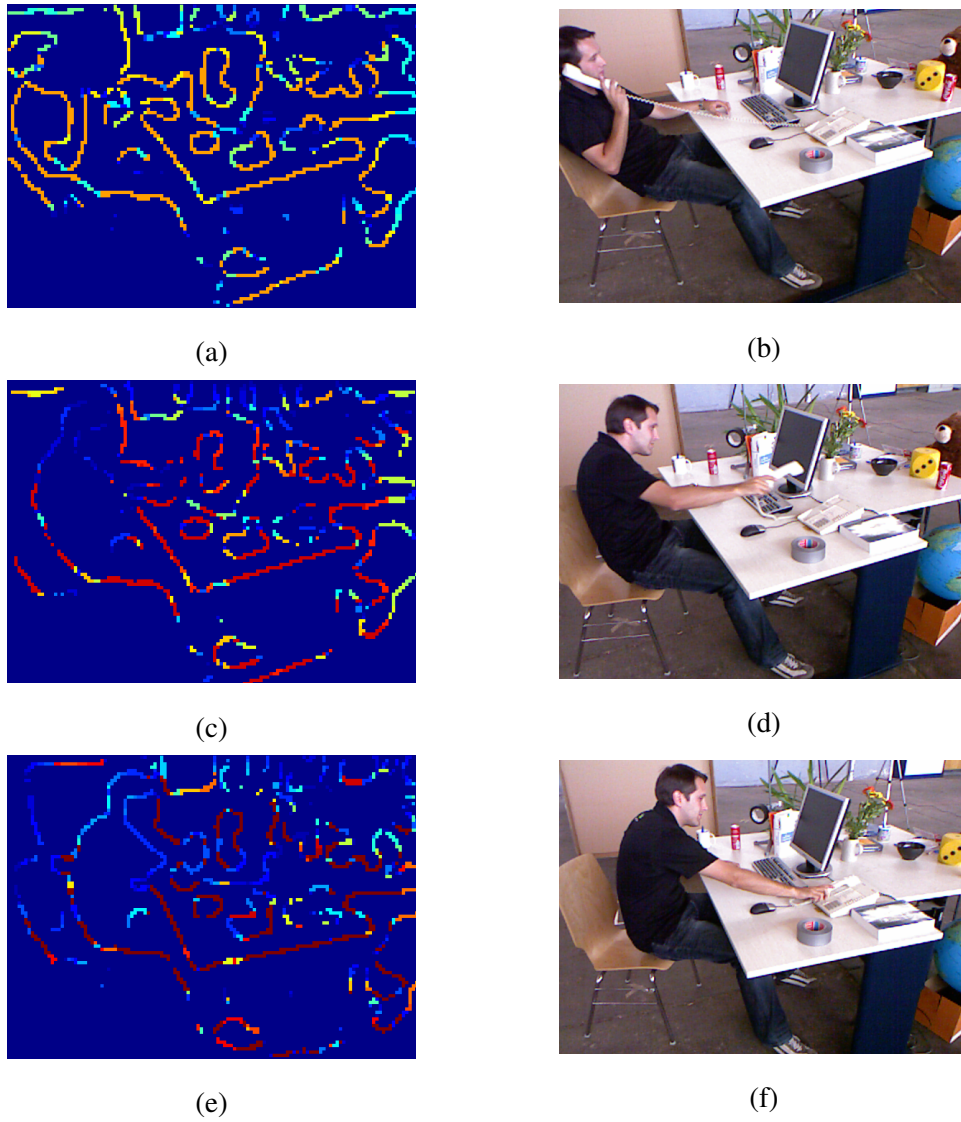


Fig. 2.20. Outlier rejection example on the TUM [45] dataset. Keyline color on (a), (c) and (e) denotes Keyline history k (the warmer, the greater)

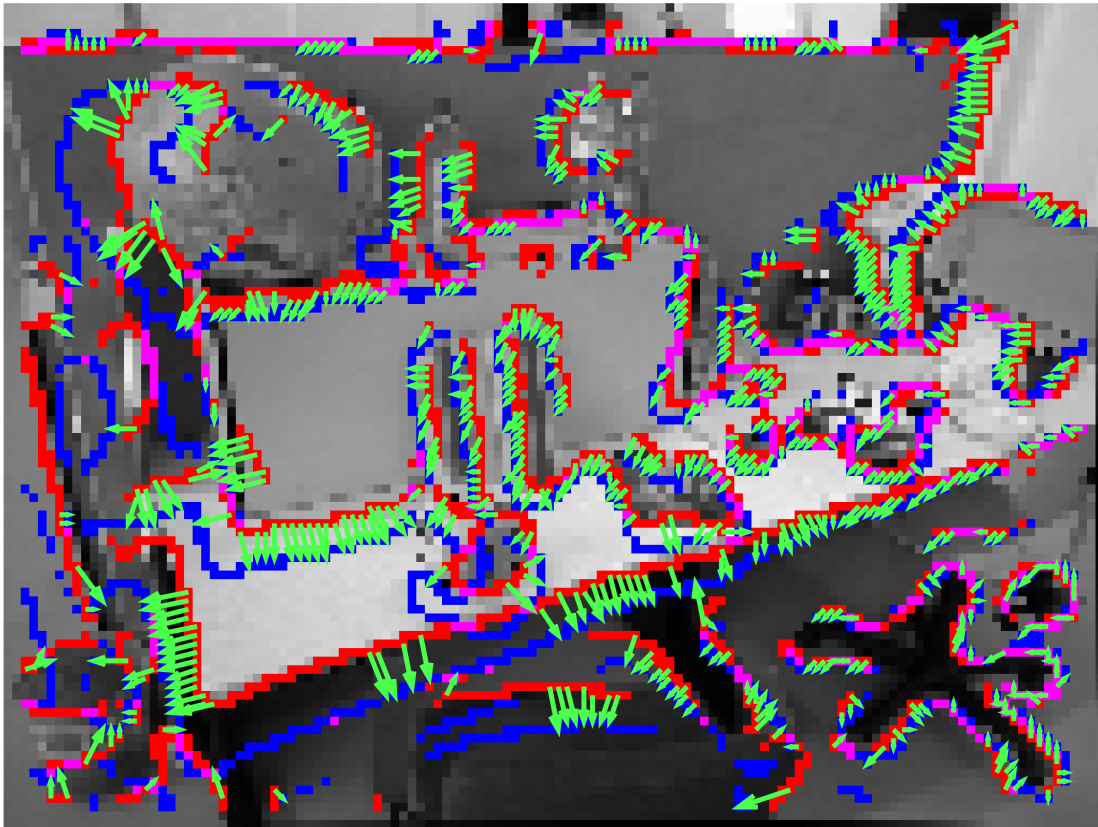


Fig. 2.21. Matches laid over an image from the TUM [45] dataset. Red pixels denote Keylines in this frame; blue – Keylines in next frame (not transformed in any way); pink – positions occupied by a Keyline in both frames, regardless if these Keylines are related; green vectors – directed matches between Keylines. Not matched Keylines are still visible

2.5.3. Regularization

Regularization is based on the assumption that Keylines located near each other on the image are also close to one another in 3D, meaning that their ρ are similar. As edges of real-life objects are seldom jagged, difference in depth between adjoining Keylines can be smoothed out. This is done by taking a weighted mean of ρ and σ_ρ of directly joined Keylines (see Section 2.3.5). Weights take into account σ_ρ and a simple angle-based gradient similarity measure. To be regularized, a Keyline needs to have 2 neighbors (this means excluding ends of edges).

However, the aforementioned assumption is not always true. This is why additional tests on neighboring pixels are needed. Firstly, depth of the neighbors must be comparable, taking into

account their uncertainties. Then angle between gradients of neighboring Keylines is checked for similarity. These tests are usually passed; most of rejected Keylines are corners.

During tests some datasets exhibited salt-and-pepper noise in estimated depth maps; e.g. single Keylines were predicted to be extremely distanced from their neighbors). Such noise had the tendency to be propagated by regularization, instead of being filtered. For such rare scenarios a **median filter was used** before the normal regularization procedure. For each Keyline, its depth was reestimated as median of its immediate neighbors, removing single wrongly predicted depths. In normal scenarios, median filter does not visibly affect algorithm outcome.

2.5.4. Depth reestimation

Depth can not be simply copied to current Keylines from corresponding previous ones, as camera has moved between frames, changing distance from the origin to 3D points. An EKF (Extended Kalman Filter) is separately used for each Keyline in estimate current depths; depth uncertainty is reestimated as well. Uncertainty of estimated translation and rotation, obtained in Section 2.4.4, is taken into account. Prediction and correction equations for ρ are (2.12) and (2.13), respectively

$$F(\rho_p, \vec{v}) = \left(\frac{1}{\frac{1}{\rho_p} + v_z} \right) N(1, Q_{rel}) + N(0, Q_{abs}) \quad (2.12)$$

where:

F – prediction function,

$N(\mu, \sigma)$ –normal-distributed random variable with mean μ and standard deviation σ ,

Q_{rel} – multiplicative noise standard deviation (constant),

Q_{abs} – additive noise standard deviation (constant).

$$H(\rho_c, \vec{v}) = \left(\left(f [v_x, v_y]^T - h_c v_z \right) \bullet \frac{\vec{g}_p}{\|\vec{g}_p\|} \right) \rho_c + N(0, 1) \quad (2.13)$$

where:

H – correction function.

Depth has been constrained in [1] between $\frac{1}{20}$ and 1000. However, it has been observed that Keylines too close to the camera can disturb the algorithm convergence. Much better results were obtained when minimal allowed depth was raised to 1. These values are relativ to initial depth in the first frame (mean value of the used distribution) and to other algorithm parameters.

2.5.5. Scale correction

Scale correction step proposed by Tarrío and Pedre does not resolve the scale ambiguity problem. It is argued in [1] that employed Kalman filter is biased, due to the fact that variance of velocity is much lower than σ_ρ . For each frame a single “shrinking factor“ Ξ can be calculated. Then it can be applied by Keylines by dividing ρ and σ_ρ by it.

Tests have concluded that Ξ oscillates too much while the system is initializing. **Therefore it is proposed not to estimate scale until system convergence** (usually a threshold of 20 frames suffices).

This does not remedy the fundamental problem of scale ambiguity in monocular systems. To fix this issue, dimensions of real-life objects present the scene need to be known a priori. Excellent example of such objects, with well defined shape and size, are **traffic signs and license plates**, ever-present in urban environments that the algorithm was originally envisioned for. Solutions for identifying those objects are:

- Haar feature-based cascade classifier combined with ORB (Oriented FAST and Rotated BRIEF) descriptor [11],
- machine learning methods: SVM (Support Vector Machine) and neural networks,
- rectangle detection by the means of Hough transform,
- sliding window applied over image saliency [58].

This however entails additional, heavy computing cost and would not fit the designated platform. Moreover, no out-of-the-box implementation was found, thus moving this problem out of the scope of this thesis.

2.6. Tests on artificial data

Along with data from KITTI and TUM datasets, some tests on generated data were performed, mainly for the edge detector. That way difficult edges (circles or sharp angles shaped like letters T, Y and L) could be examined – how they were thinned, joined, fragmented and how gradient of such edges behaved.

2D transformations are a special case of 3D ones, so they were used for tests of edge tracker and the mapper – 2D images were simply moved sideways or rotated. It was tested how these transformations would be recovered, however lack of depth data induced a lot of false positives.

Rendering a full 3D sequence using e.g. a game engine was also considered, but due to antialiasing, too simple lighting techniques, texturing simplifications (bump-mapping) and lack of raytracing, obtained results could be unreliable. Therefore instead popular datasets with well-defined ground truth data were chosen.

2.7. Trajectory fitting

In order to assess algorithm accuracy as it processes new frames, estimated camera trajectory is systematically compared with ground truth data after each frame. An optimal isotropic scaling and Euclidean transformation between two trajectories is searched for. Obtained parameters do not influence future algorithm iterations, as in real world applications such data is simply unavailable. They are calculated only for visualization purposes. These visualizations are used throughout Chapter 3.

In [59] it is described how to find an optimal (in the least square sense) rotation and translation between two trajectories 3D using SVD. Scaling factor can be determined using `fminsearch`, a black-box minimization routine available in the GNU Octave environment. Usually first 5 trajectory samples are not taken into account, as they are very noisy because of system being not yet initialized. Generally if system does converge at all, then some level of conformance with ground truth is achieved only after just a few frames.

3. Discussion of results

This chapter focuses on discussing overall algorithm results obtained from longer sequences. Tested sequences come from TUM [45] and KITTI [46] public datasets. Results include comparison of x , y and z components of the estimated camera trajectory with ground truth, as well as numeric quantities (Table 3.1). Some erroneous outcomes are also presented, with brief commentary.

3.1. Experimental setup

All of the tested sequences include ground truth (either laser positioning system in case of TUM, or Differential GPS in case of KITTI), which is essential for debugging and validation purposes. Also, these datasets are commonly used for visual odometry systems evaluation, making it possible to compare proposed algorithm with others. Intrinsic camera parameters were reported by dataset authors.

As the main goal of the algorithm is to estimate transformation between images, most care was put into making sure that the system correctly calculates position of the camera in respect to ground truth. Obtained trajectory was fitted onto ground truth trajectory using method explained in Section 2.7.

Algorithm itself was implemented in the GNU Octave environment. The OpenCV [11] library was also used; it was compiled to MEX files, that can be executed by Octave. Choice of programming language enabled fast prototyping and easy visualization, but prevented the algorithm from achieving real-time performance. Based on results in [1] it is estimated that implementation in a compiled language such as C++ would accomplish such goal even on a consumer-grade smartphone.

In order to speed up computations, images were first scaled down. This removed some false edges, while retaining only the strongest ones, thus greatly reducing the number of Keylines. In [1] it was already shown that algorithm run time depends linearly on Keyline number. The biggest downscaling factor was reduction of size by 80%, resulting in an 128 by 96 pixels wide image. Although such images are too small to be easily interpreted by humans, algorithm results were still satisfactory.

Comparison of time needed to process one frame with two different downscaling factors is featured in Fig. 3.1. For these particular images and scales, larger scale resulted in 1.81 more processing time per frame, on average. Measured time included creation of some visualization files, but this factor was insignificant next to the most time-consuming step, i.e. the minimization.

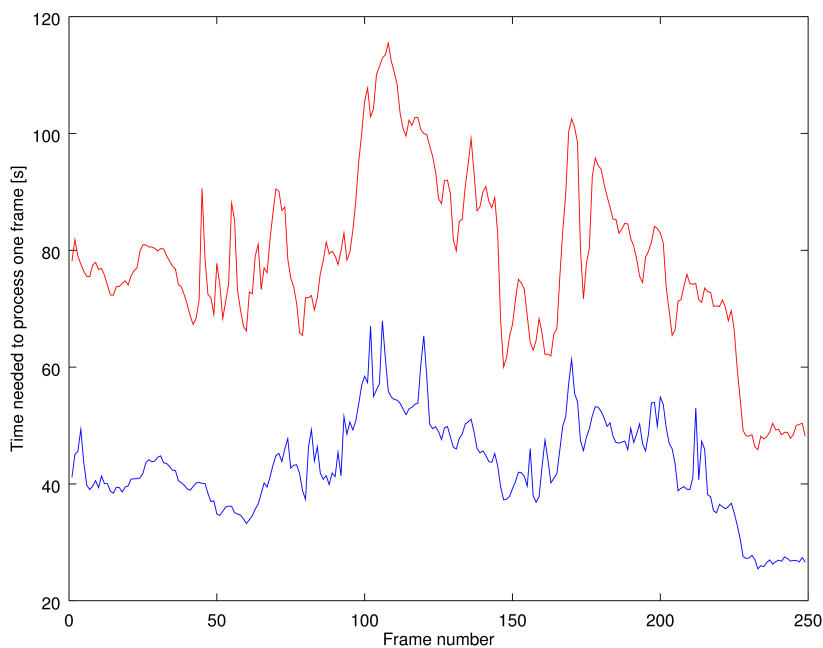


Fig. 3.1. Single frame processing time for TUM fr3_teddy sequence. Red line corresponds to scale of $\frac{1}{3}$ of original image; blue line – $\frac{1}{5}$

A few system parameters had to be adjusted between TUM and KITTI datasets, but there was no need of fine tuning the configuration for every sequence separately.

3.2. Trajectory comparison

Presented results consists mainly from the following sequences. Selected values have been collected in Table 3.1 - first two were used only for visualization as discussed in Section 2.7, last one was obtained from evaluation script *evaluate_ate* available in TUM evaluation benchmark [45]. It should be noted that these numbers are scores for only parts of said sequences.

- TUM fr1_xyz - only translatory motions are present here, as stated by creators of this dataset "*it's mainly for debugging purposes*",
- TUM fr2_desk - slow sweep around an office desk,
- TUM fr2_desk_with_person - same as above, but present person moves the objects around, creating outliers,
- TUM fr2_pioneer_SLAM - robot with a mounted camera is trying to self-localize and map the room,
- TUM fr3_long_office_household - slow sweep around two desks,
- TUM fr3_teddy - slow sweep around a big plush teddy bear,
- KITTI 01 - video registered by cameras¹ mounted in front of a car.

Overall, the algorithm performed well in scenarios where camera underwent complex movement, instead of simple linear translations. This is due to the fact that in the former case disparities between frames generally make it easier to recover the egomotion, independently from employed odometry algorithm. For example, a view from top of a trajectory is presented in Fig. 3.2. Despite the low number of frame, complex motion has been recovered relatively closely. Even in a dataset containing outliers (independently moving objects), odometry was correct (see Fig. 3.3).

Sometimes frames did not contain enough Keylines and system had to be reinitialized. In case of TUM fr3_teddy sequence there are 4 cases of reinitialization visible in Fig. 3.4: frames 271 & 272, 280, 386 & 387 and 394. System has recovered from three out of four of them, but

¹Output from only one camera was used for algorithm evaluation, as it is a monocular system.

Table 3.1. Algorithm results for selected parts of sequences

Sequence	Average drift ^a [$\frac{m}{s}$]	Scale	ATE ^b [m]
TUM fr3_long_office_household	3.6	1.6	0.302
TUM fr3_teddy (long subsequence)	1.8	1.1	0.287
TUM fr3_teddy (short subsequence)	0.6	1.5	0.175
TUM fr2_desk_with_person	0.4	1.05	0.133
TUM fr2_desk_with_person	0.1	1.69	0.066

^a RMSE between fitted trajectory points, multiplied by the frame rate (FPS)

^b Absolute Trajectory Error – RMSE in meters after trajectory alignment

the third² was fatal in a sense that scale and starting rototranslation were lost, and hence the trajectory as a whole could not be well fitted with ground truth (Fig. 3.5). However, partitioned sections can and are fitted quite well, as shown in Fig. 3.6 for the first part, and in Fig. 3.7 for the second. Such points could be potentially detected in post-processing, as they are a sharp spots on an otherwise smooth 3D curve.

This particular reinitialization is especially interesting, because of a depth estimation artifact. Movement in this sequence is a circular, around the teddy bear – in every frame camera moves and rotates a bit, so as to always have the teddy bear in the middle of the picture frame. This causes the toy’s edges to be more or less in the same spot, while edges of the environment are shifted a bit more. This in turn can lead to the system estimating that since the edges of this object, actually lying the closest in 3D, don’t move a lot, they have to be very far away, and the rest of picture is closer. Example of such bad initialization is presented in Fig 3.8. It is worth mentioning that in such cases estimated trajectory, after being reinitialized, also did fit the ground truth, despite having calculated wrong Keyline depths.

Places where rototranslation is lost due to reinitialization prevent usage of SLAM loop closure methods, as the curves are very far from closing. For instance, in Fig. 3.9, an expected loop and obtained trajectory are compared.

²There were simply not enough edges (objects) present on the photo. Paired with motion blur, this resulted in low Keyline number, no matter the image scale and gradient detection threshold.

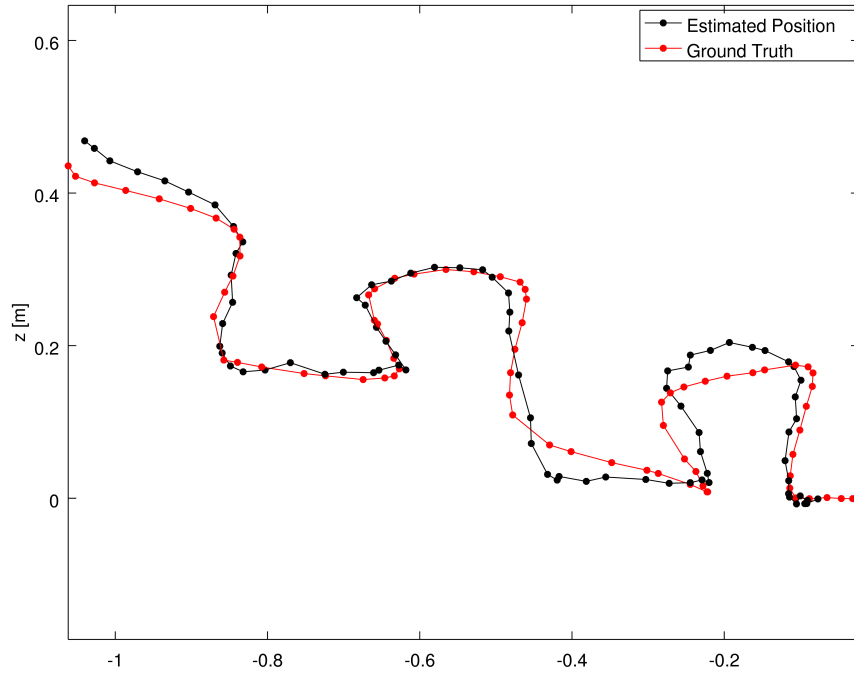


Fig. 3.2. Example trajectory comparison between obtained results and ground truth from dataset TUM fr2_desk_with_person, projected on the $y = 0$ plane, as camera height was the least significant motion component.

The KITTI dataset was far more challenging than TUM. It very often happens that through many consecutive frames, one half of images systematically lack Keylines, affecting the algorithm. As camera was mounted on a car, individual displacements are more rapid. It is even possible that rolling shutter problem is present, although this was not tested. Another issue is that when car is moving forward, many Keylines are present in the middle of the field of view. They undergo little to no displacement between frames, therefore they might bias the minimization and prevent other Keylines from being matched (in the described scenario edges moderately close to image border contain most information about motion of the camera, due to visible disparity). A trajectory estimated for one of KITTI sequences is depicted in Fig 3.10. These results are rather disheartening, as they show that algorithm needs to be improved before it can be used in urban scenarios for which it was originally planned in this thesis.

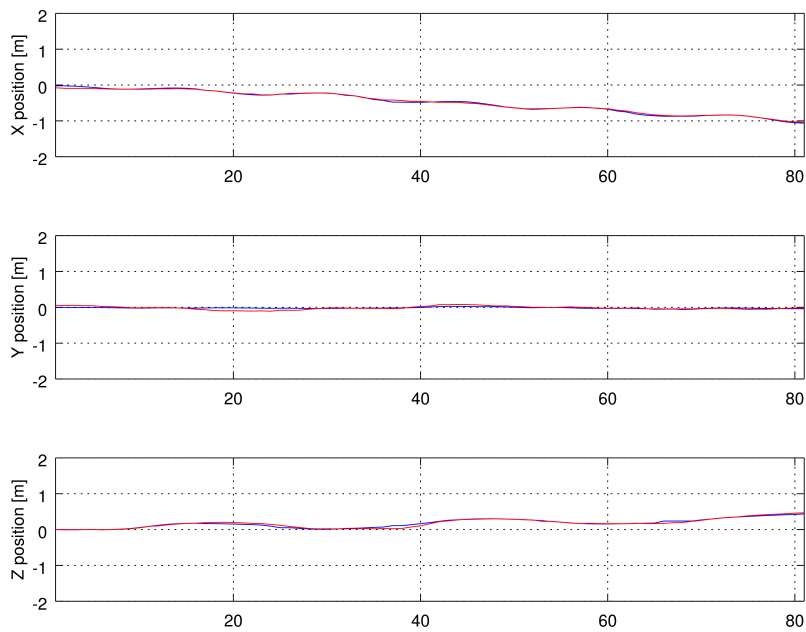


Fig. 3.3. Example trajectory components comparison between obtained results and ground truth from dataset TUM fr2_desk_with_person. Red line marks estimated position; blue – ground truth

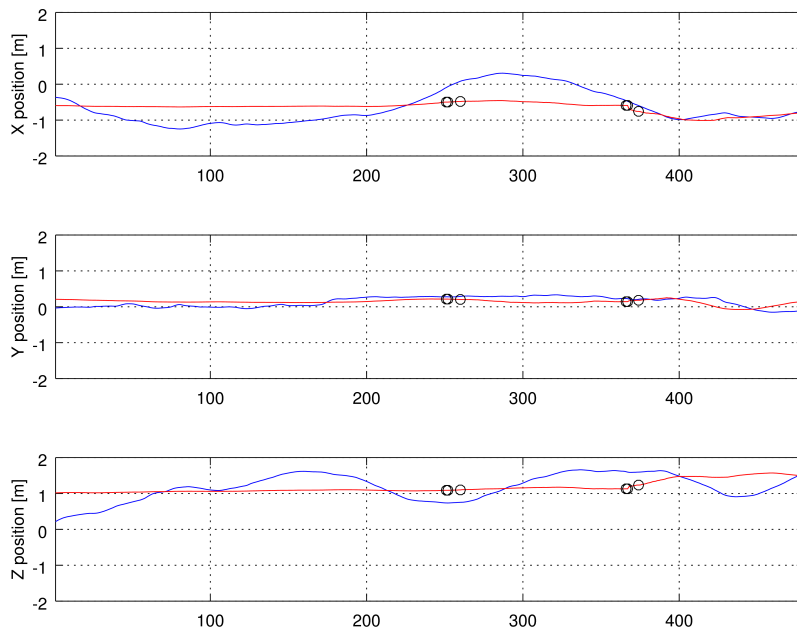


Fig. 3.4. Whole trajectory with denoted cases of reinitialization from dataset TUM fr3_teddy. Red line marks estimated position; blue – ground truth; black circles – reinitialization

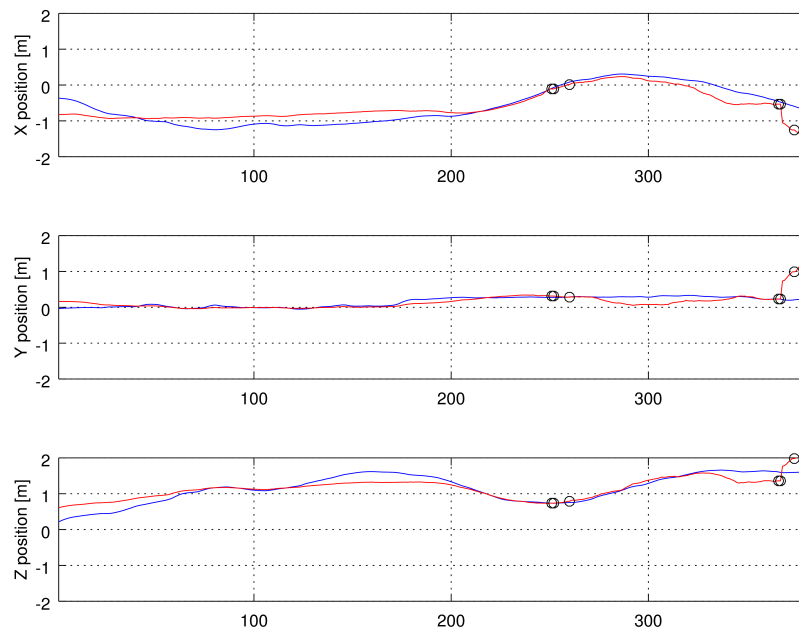


Fig. 3.5. Fitted trajectory showing fatal reinitialization problem from dataset TUM fr3_teddy. Red line marks estimated position; blue – ground truth; black circles – reinitialization

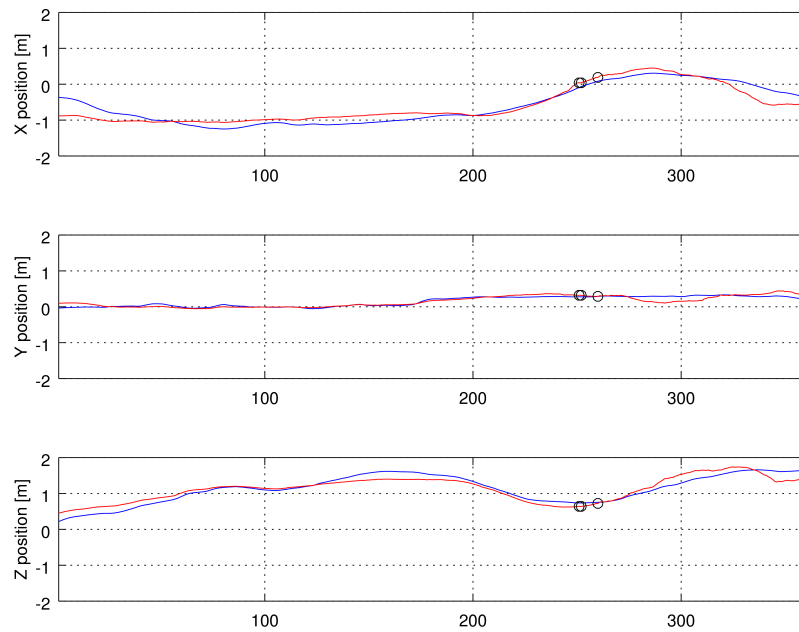


Fig. 3.6. First part of the sequence fitted to ground truth from dataset TUM fr3_teddy. Red line marks estimated position; blue – ground truth; black circles – reinitialization

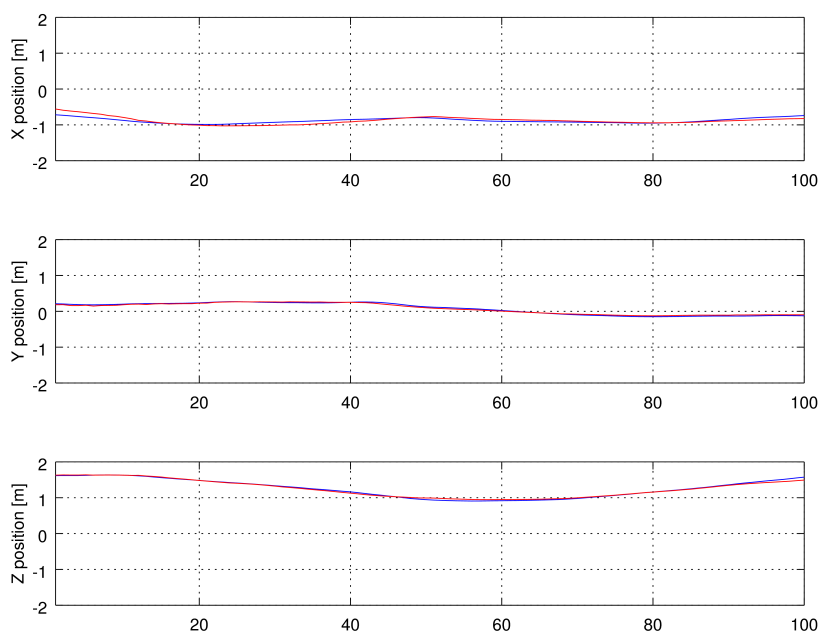


Fig. 3.7. Second part of the sequence fitted to ground truth from dataset TUM fr3_teddy. Red line marks estimated position; blue – ground truth; black circles – reinitialization

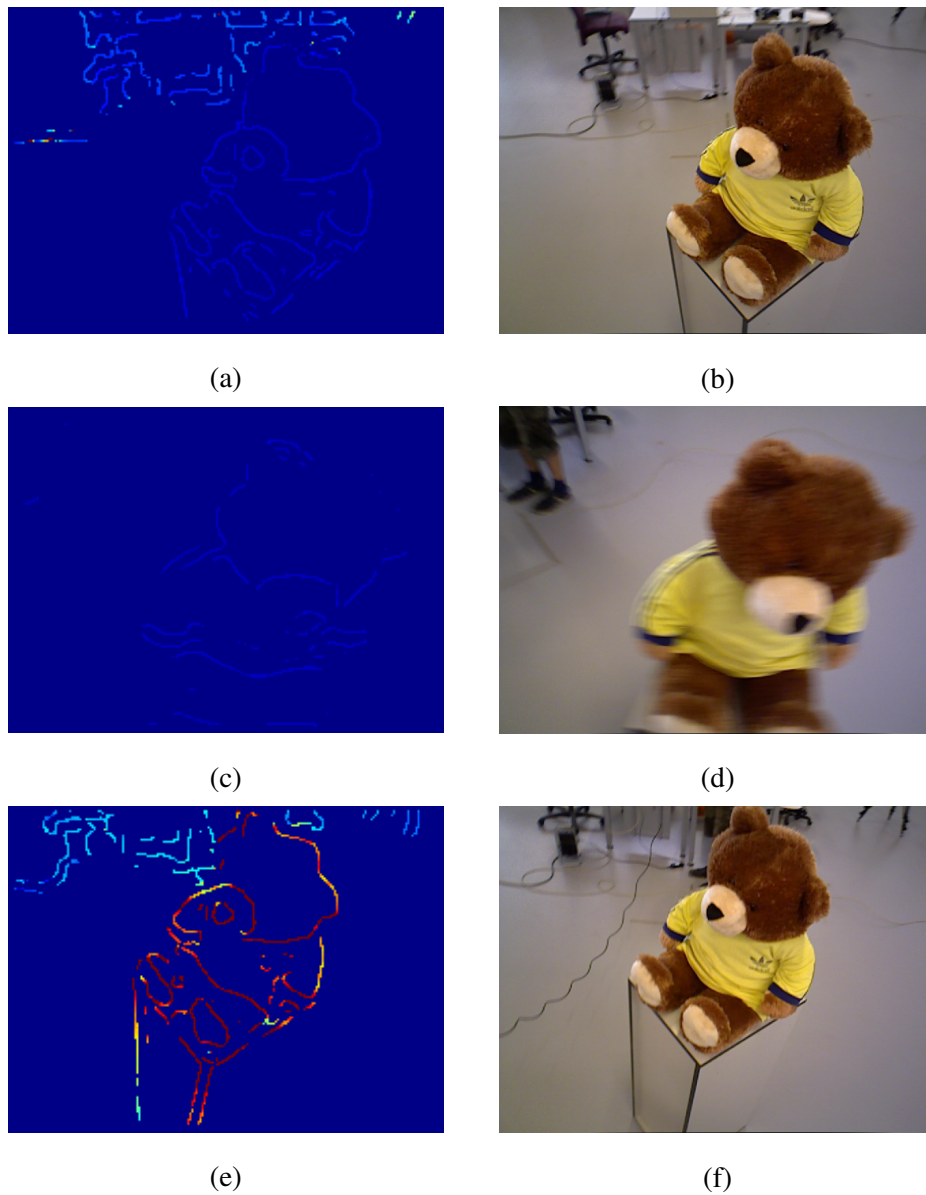


Fig. 3.8. An example of invalid depth estimation after system reinitialization. Warm colors denote large edge distance from the camera. (a), (b) Well-estimated depth, as background is warmer (cyan) and foreground is cooler (blue), (c), (d) Depth is reset with random values due to motion blur, (e), (f) Invalid depth after few frames – depth of the bear is estimated to be larger than the objects that are situated behind it in reality

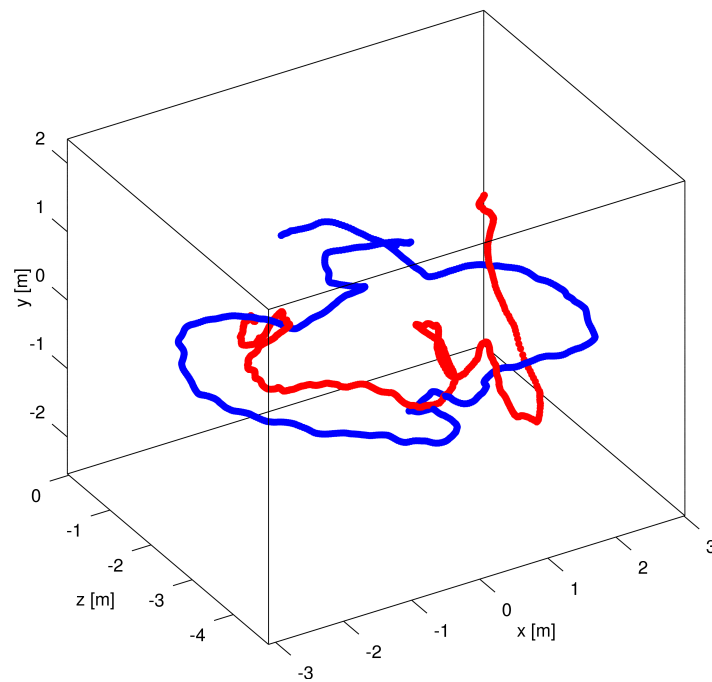


Fig. 3.9. 3D trajectory obtained from TUM fr3_long_office_household, which features a closed loop. Red points marks estimated position; blue – ground truth

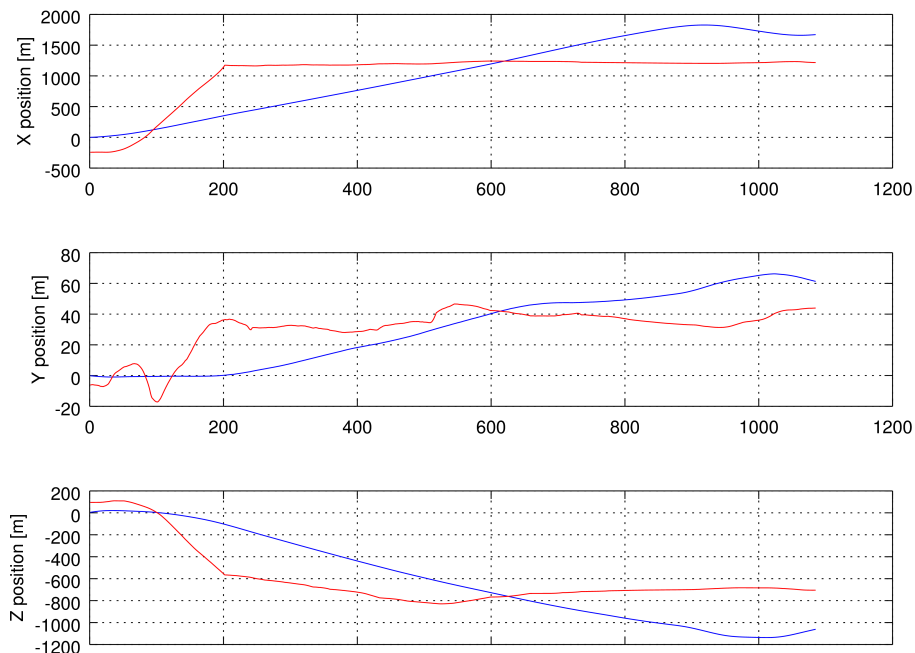


Fig. 3.10. Estimated trajectory of KITTI 01 sequence, fitted to ground truth. Red line marks estimated position; blue – ground truth. Due to large distance traveled, individual subplots have different scales

4. Conclusion

Overall, presented algorithm continues the novel take on lightweight and pure visual odometry from [1]. An easier way of implementing VO would have been the classic approach: extracting feature points using one of available feature detectors, matching them and then estimating the rototranslation; possibly including IMU and/or input from the other camera along the way. However, a lot of research by the scientific community has already been performed in this area (but of course existing solutions are still being perfected). This was covered in Chapter 1. Studies described in this thesis aimed to explore an approach that was off the beaten track and could operate in real time on mobile devices, instead of high-end PCs.

Results are promising, especially in cases of complicated motion, which was primary problem on assumed designation of implementation of said algorithm. Trajectories fit nicely onto the ground truth to some degree, which means that instantaneous azimuths are being estimated adequately. They are crucial, because absolute camera position was originally supposed to be acquired from a GPS unit, and camera orientation – from odometry. Algorithm needs to be robustified against challenges of urban scenes, though. Presented implementation is publicly available via a Git repository: <https://github.com/kaszczesny/robustified-rototranslation>, so that it can be independently verified, or perhaps further developed.

Additions to the original implementation [1] that were described in Chapter 3 include:

- removal of artificial border edges possibly created during image rectification (applied before any edge detection substep),
- one more edge detection test, that removes the need of having to later handle division by zero (or by numbers close to zero),
- simpler, but a little bit more effective edge detection threshold control mechanism (instead of hysteresis),

- depth initialization with normal-distributed random numbers in the first frame of the sequence, instead of a constant,
- creation of (marginally) more accurate lookup table for minimization,
- minor speed optimization – usage of Cholesky decomposition instead of SVD, based on the observation that decomposed matrix is positive definite [53],
- optional median filter applied to edge depths before further regularization,
- proposition of estimated depth bounds that perform better.

A handful of possible ways to further improve presented solution have been addressed also throughout the Chapter 3:

- incorporation of fuzzy logic into edge detection,
- feature-based depth initialization (deemed unnecessary),
- usage of full pinhole camera model in projection and back-projection functions,
- scale ambiguity resolution by using a priori knowledge of traffic signs and license plates sizes.

Some more general enhancements can also be proposed:

1. Instead of direct Jacobian calculation like Tarrío and Pedre, a black-box approach can be taken and Jacobians could be calculated numerically, e.g. using the Secant version of Levenberg-Marquardt algorithm [53]. Alternatively, an entirely different minimization algorithm could be picked, but this might affect time performance.
2. Many algorithm steps are performed independently for each Keyline. This suggests that they could be parallelized in the GPU (Graphics Processing Unit), taking performance to a whole new level. Nowadays GPUs are present not only in PCs, but also in mobile devices.
3. Algorithm is configured with roughly 30 parameters, that affect final outcomes in varying degrees. Most of their values have been simply acquired from [1]. An extensive search for their optimal values, performed over large datasets, could be undertaken.
4. As far as visualizations are concerned, another color map should be used. Many of the figures presented in Chapter 2 (and all images that were created automatically by the

implementation) used the default `jet` colormap, or colors were chosen manually. Authors are aware that `jet` should not be used to represent dense fields [60]. Figures did depict only individual edges, but still their visual appeal could be enhanced.

5. Keyline matching criteria (especially during the minimization step) could be dynamically changed, so that distant, but well-matched Keylines do not bias the outcome as it was observed when KITTI [46] dataset was used.
6. Obtained trajectory can be Kalman-filtered, e.g. as in [61].
7. System could be transformed to a SLAM one. As long as only a few of best features would be remembered, real time would not be hindered [25]. As it is not expected to form closed loops in real life navigation scenarios, saved features could be instead used to tackle occlusions.
8. GPS information could be incorporated, as it was originally intended; e.g. by taking approach similar to [23]. It could, over larger time scale, alleviate the problem of error accumulation when trajectory rapidly changes direction due to system reinitialization (and error accumulation in general).

Finally, the algorithm should be reimplemented in a more efficient programming language, such as C++. Time complexity, as well as behavior in real life scenarios, could be then verified.

Bibliography

- [1] Juan Jose Tarrío and Sol Pedre. “Realtime edge-based visual odometry for a monocular camera”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 702–710.
- [2] *Pokémon GO*. URL: <https://pokemongolive.com/en> (visited on 2017-08-29).
- [3] *ARCore*. *Augmented reality at Android scale*. URL: <https://www.blog.google/products/google-vr/arcore-augmented-reality-android-scale/> (visited on 2017-08-31).
- [4] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [5] Krzysztof Szczesny. “Analysis of Algorithms for Geometric Distortion Correction of Camera Lens”. Engineering Diploma Thesis. AGH University of Science and Technology, 2016.
- [6] Thomas Stehle et al. “Camera calibration for fish-eye lenses in endoscopy with an application to 3d reconstruction”. In: *Biomedical Imaging: From Nano to Macro, 2007. ISBI 2007. 4th IEEE International Symposium on*. IEEE. 2007, pp. 1176–1179.
- [7] Janne Heikkilä and Olli Silven. “Calibration procedure for short focal length off-the-shelf CCD cameras”. In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. Vol. 1. IEEE. 1996, pp. 166–170.
- [8] John G Fryer and Duane C Brown. “Lens distortion for close-range photogrammetry”. In: *Photogrammetric engineering and remote sensing* 52.1 (1986), pp. 51–58.
- [9] Zhengyou Zhang. “Flexible camera calibration by viewing a plane from unknown orientations”. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 1. Ieee. 1999, pp. 666–673.
- [10] Stephen DiVerdi and Jonathan T Barron. “Geometric calibration for mobile, stereo, autofocus cameras”. In: *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on*. IEEE. 2016, pp. 1–8.

- [11] *OpenCV. Program documentation*. Version 3.1. URL: <http://docs.opencv.org/3.1.0/> (visited on 2017-08-29).
- [12] Andrew W Fitzgibbon. “Simultaneous linear estimation of multiple view geometry and lens distortion”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001.
- [13] R Grompone Von Gioi et al. “Towards high-precision lens distortion correction”. In: *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE. 2010, pp. 4237–4240.
- [14] Richard Hartley and Sing Bing Kang. “Parameter-free radial distortion correction with center of distortion estimation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.8 (2007), pp. 1309–1321.
- [15] Frederic Devernay and Olivier Faugeras. “Straight lines have to be straight”. In: *Machine vision and applications* 13.1 (2001), pp. 14–24.
- [16] Boguslaw Cyganek and J Paul Siebert. *An introduction to 3D computer vision techniques and algorithms*. John Wiley & Sons, 2011.
- [17] Zhengyou Zhang. “Determining the epipolar geometry and its uncertainty: A review”. In: *International journal of computer vision* 27.2 (1998), pp. 161–195.
- [18] Dan Pojar, Pangu Jeong, and Sergiu Nedevschi. “Improving localization accuracy based on lightweight visual odometry”. In: *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE. 2010, pp. 641–646.
- [19] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.
- [20] Christian Dornhege and Alexander Kleiner. “Visual odometry for tracked vehicles”. In: (2006).
- [21] Hongshan Yu et al. “An improved visual odometry optimization algorithm based on Kinect camera”. In: *Chinese Automation Congress (CAC), 2013*. IEEE. 2013, pp. 691–696.
- [22] Kouros Khoshelham and Sander Oude Elberink. “Accuracy and resolution of kinect depth data for indoor mapping applications”. In: *Sensors* 12.2 (2012), pp. 1437–1454.
- [23] Ignacio Parra Alonso et al. “Accurate global localization using visual odometry and digital maps on urban environments”. In: *IEEE Transactions on Intelligent Transportation Systems* 13.4 (2012), pp. 1535–1545.

- [24] Maciej Polanczyk et al. “The application of Kalman filter in visual odometry for eliminating direction drift”. In: *Signals and Electronic Systems (ICSES), 2010 International Conference on*. IEEE. 2010, pp. 131–134.
- [25] Andrew J Davison et al. “MonoSLAM: Real-time single camera SLAM”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.
- [26] David Nistér, Oleg Naroditsky, and James Bergen. “Visual odometry”. In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. Vol. 1. Ieee. 2004, pp. I–I.
- [27] Yong Ren et al. “A stereo visual odometry based on SURF feature and three consecutive frames”. In: *Smart Cities Conference (ISC2), 2015 IEEE First International*. IEEE. 2015, pp. 1–5.
- [28] Felix Woelk and Reinhard Koch. “Fast monocular bayesian detection of independently moving objects by a moving observer”. In: *Joint Pattern Recognition Symposium*. Springer. 2004, pp. 27–35.
- [29] João Paulo Costeira and Takeo Kanade. “A multibody factorization method for independently moving objects”. In: *International Journal of Computer Vision* 29.3 (1998), pp. 159–179.
- [30] Yanpeng Cao, Peter Cook, and Alasdair Renfrew. “Vehicle ego-motion estimation by using pulse-coupled neural network”. In: *Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*. IEEE. 2007, pp. 185–191.
- [31] Michal Irani, Benny Rousso, and Shmuel Peleg. *Recovery of ego-motion using image stabilization*. Hebrew University of Jerusalem. Leibniz Center for Research in Computer Science. Department of Computer Science, 1993.
- [32] Jianbo Shi et al. “Good features to track”. In: *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on*. IEEE. 1994, pp. 593–600.
- [33] Christoph Feichtenhofer and Axel Pinz. “Spatio-temporal Good Features to Track”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 246–253.
- [34] Johannes Gräter, Tobias Schwarze, and Martin Lauer. “Robust scale estimation for monocular visual odometry using structure from motion and vanishing points”. In: *Intelligent Vehicles Symposium (IV), 2015 IEEE*. IEEE. 2015, pp. 475–480.
- [35] Xiaojing Song, Lakmal D Seneviratne, and Kaspar Althofer. “A Kalman filter-integrated optical flow method for velocity sensing of mobile robots”. In: *IEEE/ASME Transactions on Mechatronics* 16.3 (2011), pp. 551–563.

- [36] Chen Xiao et al. “A novel approach to improve the precision of monocular visual odometry”. In: *Information and Automation, 2015 IEEE International Conference on*. IEEE. 2015, pp. 392–397.
- [37] Etienne Mouragnon et al. “Real time localization and 3d reconstruction”. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2006, pp. 363–370.
- [38] Jianjun Gui, Dongbing Gu, and Huosheng Hu. “Robust direct visual inertial odometry via entropy-based relative pose estimation”. In: *Mechatronics and Automation (ICMA), 2015 IEEE International Conference on*. IEEE. 2015, pp. 887–892.
- [39] Rafid Siddiqui and Siamak Khatibi. “Robust visual odometry estimation of road vehicle from dominant surfaces for large-scale mapping”. In: *IET Intelligent Transport Systems* 9.3 (2014), pp. 314–322.
- [40] Jakob Engel, Jurgen Sturm, and Daniel Cremers. “Semi-dense visual odometry for a monocular camera”. In: *Proceedings of the IEEE international conference on computer vision*. 2013, pp. 1449–1456.
- [41] Chris Harris and Mike Stephens. “A combined corner and edge detector.” In: *Alvey vision conference*. Vol. 15. 50. Manchester, UK. 1988, pp. 10–5244.
- [42] Javier Civera, Andrew J Davison, and JM Martinez Montiel. “Inverse depth parametrization for monocular SLAM”. In: *IEEE transactions on robotics* 24.5 (2008), pp. 932–945.
- [43] Joao P Barreto, Rahul Swaminathan, and Jose Roquette. “Non parametric distortion correction in endoscopic medical images”. In: *3DTV Conference, 2007*. IEEE. 2007, pp. 1–4.
- [44] Shichao Yang and Sebastian Scherer. “Direct Monocular Odometry Using Points and Lines”. In: *arXiv preprint arXiv:1703.06380* (2017).
- [45] J. Sturm et al. “A Benchmark for the Evaluation of RGB-D SLAM Systems”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. Oct. 2012.
- [46] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.
- [47] Richard I Hartley. “In defense of the eight-point algorithm”. In: *IEEE Transactions on pattern analysis and machine intelligence* 19.6 (1997), pp. 580–593.
- [48] John Canny. “A computational approach to edge detection”. In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

- [49] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. “Machine vision, vol. 5”. In: *McGrawHill New York* (1995).
- [50] Anna Fabijańska. “Subpixel Edge Detection in Blurry and Noisy Images”. In: *International Journal of Computer Science and Applications* 12.2 (2015), pp. 1–19.
- [51] Frédéric Devernay. “A non-maxima suppression method for edge detection with sub-pixel accuracy”. PhD thesis. INRIA, 1995.
- [52] David G Lowe. “Object recognition from local scale-invariant features”. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.
- [53] Kaj Madsen, Hans Bruun Nielsen, and Ole Tingleff. “Methods for non-linear least squares problems”. In: (2004).
- [54] Jose-Luis Blanco. “A tutorial on se (3) transformation parameterizations and on-manifold optimization”. In: *University of Malaga, Tech. Rep 3* (2010).
- [55] Peter J Huber et al. “Robust estimation of a location parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101.
- [56] William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.
- [57] Ignacio Parra, Miguel Angel Sotelo, and Ljubo Vlacic. “Robust visual odometry for complex urban environments”. In: *Intelligent Vehicles Symposium, 2008 IEEE*. IEEE. 2008, pp. 440–445.
- [58] Kai-Hsiang Lin, Hao Tang, and Thomas S Huang. “Robust license plate detection using image saliency”. In: *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE. 2010, pp. 3945–3948.
- [59] K Somani Arun, Thomas S Huang, and Steven D Blostein. “Least-squares fitting of two 3-D point sets”. In: *IEEE Transactions on pattern analysis and machine intelligence* 5 (1987), pp. 698–700.
- [60] David Borland and Russell M Taylor Ii. “Rainbow color map (still) considered harmful”. In: *IEEE computer graphics and applications* 27.2 (2007).
- [61] Hai-Gen Min et al. “Visual odometry for on-road vehicles based on trifocal tensor”. In: *Smart Cities Conference (ISC2), 2015 IEEE First International*. IEEE. 2015, pp. 1–5.