UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS no. 358

Robust point tracking for visual navigation

Ante Trbojević

Zagreb, July 2012.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

I would like to thank my supervisors Siniša Segvić and François Chaumette, who suggested the topic, for their support, technical assistance and advises. Also, special thanks to Fabien Spindler for technical support, implementation advises and patience during the experimenting with the developed software.

This work has been done during my research internship in Lagadic group (INRIA, Rennes, France).

CONTENTS

1. Introduction

The goal of this master thesis is to design a software that will be able to autonomously navigate a robot from an initial location to a desired location in outdoor environment. This task can be performed with two steps: *mapping* and *navigation*. The *mapping* is a learning process where the robot tries to learn a path on which it is going to operate autonomously later during the *navigation* phase. The learning phase is an offline process during which the robot is usually guided by human control. The *navigation* phase has to be a real-time process, so that robot can autonomously move through the pre-learned path.

Today, there are two main approaches how to accomplish visual navigation of the robot using computer vision: the *model-based* approach and the *appearance-based* approach

In the *model-based* approach the environment is represented by a 3D model. Simply, during the learning, geometrical information such as lines[8] or planes [7] is extracted from environment as the robot travels along desired path. Using such features a 3D model can be created describing the working environment (world) where the robot is operating. During the navigation phase, the localization is done by matching the learned model with the current local model reconstructed from currently acquired data from sensors.

The *appearance-based* approach splits the path to the several smaller paths that are connected in a directed chained graph. Each node in the graph represents the location where information, i.e. description of environment, has been obtained from data sensors and stored in a visual memory. So, a link between nodes gives a possibility for the robot to navigate autonomously between these two nodes. During the navigation, the goal is to navigate the robot from the initial node to the adjacent node. Information stored in visual memory for those two nodes is used for localization, cf. Figure 1.1. Knowing the difference between description of environment in the current view (acquired from camera during the navigation) and the reference view (the one stored in the nodes during the learning) it is possible to do the localization of the robot. When a

reference node is reached context is switched to the next pair of adjacent nodes, i.e. the context for visual navigation is replaced with the context stored in visual memory for the next adjacent pair of nodes.

The concept of splitting path in several smallest is described in [18]. Using this concept an framework for autonomous navigation of robot has been designed in [19]. The learned path in [19] is organized as a direct chained graph, where each node is called *key-image* and a link between two key-images represent a *local world* where visual navigation can be easily and robustly performed, cf. Figure 1.2.

Figure 1.1: Visual memory is organized as ordered set of key-images $\{I_0, \ldots, I_n\}$. Each key-image contains features that can be used for visual navigation during the navigation phase.

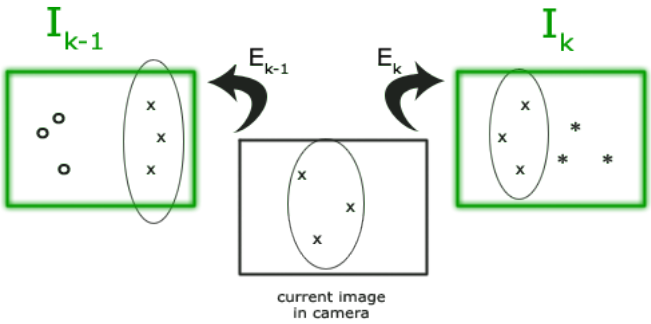


Figure 1.2: Local world k is consisted of two adjacent key-images (I_{k-1}, I_k) with correspondent set of features (marked as '$\times$' in the images). Those features, presented in both key-images, are projected in the current camera view and used for the visual navigation. E_{k-1} is a geometry between the current view and the preceding key-image view, while E_k is a geometry between the current view and the following key-image view.

The developed system in this master thesis relies on the proposed system in [19]. The design concept is the same as in [19]. The tracker with the affine model is replaced with the tracker with the translational model, as also MLESAC with RANSAC (RANSAC + LMedS). As well, the five point relative pose algorithm [17] is not used, instead the simple eight point algorithm is used. The feature prediction and the smooth transition are also avoided in this system.

This paper is organized as follows. Assumptions and information about used hardware are discussed in Chapter 2. Chapter 3 provides the description of photometric tracker for robust point tracking and description of additional improvements of tracker. Details about geometry reconstruction from two views are described in Chapter 4. Information about the learning process can be found in Chapter 5, while the navigation phase is described in Chapter 6. Results are provided in Chapter 7 with examples, images and videos of the learning and the navigation. Finally, the conclusion with discussion about advantages and disadvantages of the implemented solution, possible future improvements and applications is provided in Chapter 8.

2. Assumptions

The proposed system is designed for outdoor environment navigation. The robot used for testing is equipped with a single perspective camera which is mounted and fixed on the front panel of the mobile robotic car. This sensor is the only available sensor for gathering data and captures cca. 15 fps. Therefore, the developed system relies only on images acquired from the camera, other sensor are not used. In fact, the robot is equipped with obstacle avoidance hardware, so that robot automatically avoids obstacles. Obstacle avoidance has been implemented in low-level and access to the system, as also to the obstacle avoidance hardware, is not available. The robot motion is controlled by visual navigation, but if the obstacle avoidance system detects an obstacle then circumvention is performed by the obstacle avoidance system and all motion commands of the visual navigation are ignored.

The robot is moving forward with the approximately constant speed and only one angle of rotation can be used for the manipulation. Therefore, the robot has only 1 DOF (degree-of-freedom), the rotation angle that enables robot to turn left or right, i.e. the yaw angle.

In order to decrease error of input data in images, such as distortion of camera image, camera calibration has been performed. Four intrinsic linear parameters, i.e. focal length in terms of pixels (α_x, α_y) and principal point (u_0, v_0) , and one radial distortion parameter k_{du} are determined. The radial distortion parameter k_{du} is used for correction of tracked points in image before the points are forwarded to the two-view geometry reconstruction procedure. The parameter k_{ud} is used for opposite direction, i.e. for transforming point from undistorted coordinates to distorted. This parameter is also needed because of determining the positions of features from visual memory during the navigation phase (the projection of features).

3. Point feature tracking

Several algorithms for tracking point features have been examined. All tested algorithms are photometric trackers based on Kanade-Lucas-Tomasi Feature Point Tracker. The first version of tracker is described in [16] and a more sophisticated version has been proposed in [21]. Finally, the whole process, together with discussion which features are the most appropriate for tracking, is explained in [20].

The OpenCV's tracker, Birchfield's implementation and the modification of Birchfield's tracker described in [9] have been evaluated during the learning (mapping) phase.

The implementation of KLT tracker in OpenCV library is based on [20], details about the implementation can be found in [4]. The Birchfield's tracker [1], unlike the OpenCV's tracker, contains affine consistency check [2] and insensitivity to the lighting. The modificitation described in [9] contains isotropic scaling and affine contrast compensation (without the lighting insensitivity component). The OpenCV implementation is cca. 30% computationally faster than the Birchfield's tracker, while the the modification of Birchfield's implementation [9] is the slowest one, cca. 50% slower than the OpenCV's tracker.

Using the tracker with the *affine consistency check* a large number of bad feature tracks can be eliminated from feature set as it's shown in [19], it is especially useful for detecting feature drifts. As the OpenCV does not contain the affine model of tracker, therefore the affine consistency check can not be easily implemented. This deficiency has been overcome with additionally filtering of tracked features like *similarity consistency check*, *translational consistency check* and *large movement elimination*.

3.1. Similarity consistency check

Similarity consistency check compares the descriptor of feature in the current image with the descriptor of same feature in the image where it has been detected (where

[1] The source code and details about the implementation: http://www.ces.clemson.edu/ stb/klt/

the tracking of feature has been initialized). A feature is removed from the tracking feature set, if similarity error between those two descriptors is large. The BRIEF (Binary Robust Independent Elementary Features) descriptor [5] has been chosen as feature descriptor, mainly because it can be fast computed and because of invariance to brightness and blurring [14]. The only disadvantage is variation to scale. On the contrary SURF descriptor [1] which is invariant to brightness, rotation and scale has not been chosen because the computation of SURF descriptor is slightly too slow for the navigation phase, as also the invariance to rotation is not desirable.

3.1.1. BRIEF descriptor

Binary Robust Independent Elementary Features (BRIEF) is an efficient feature point descriptor designed for fast creation and comparison. BRIEF algorithm computes binary strings from image patches, this binary strings can be easily compared using Hamming distance.

Each bit in a binary string represents a comparison between two points inside a patch. Bit is set to value 1 if first point has a higher intensity then other point. The comparison between two points $\mathbf{x}$ and $\mathbf{y}$ inside patch $\mathbf{p}$ of size $S \times S$ is defined as *test* τ

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases}, \tag{3.1}$$

where $\mathbf{p}(\mathbf{x})$ is the pixel intensity in a smoothed version of $\mathbf{p}$ at $\mathbf{x} = (u, v)^T$. If the binary string has n bits, then n $(\mathbf{x}, \mathbf{y})$ - locations have to be picked and tested, so BRIEF descriptor of patch (feature) $\mathbf{p}$ is defined as

$$f_n(\mathbf{p}) := \sum_{1 \le i \le n} 2^{i-1} \cdot \tau(\mathbf{p}; \mathbf{x}, \mathbf{y}). \tag{3.2}$$

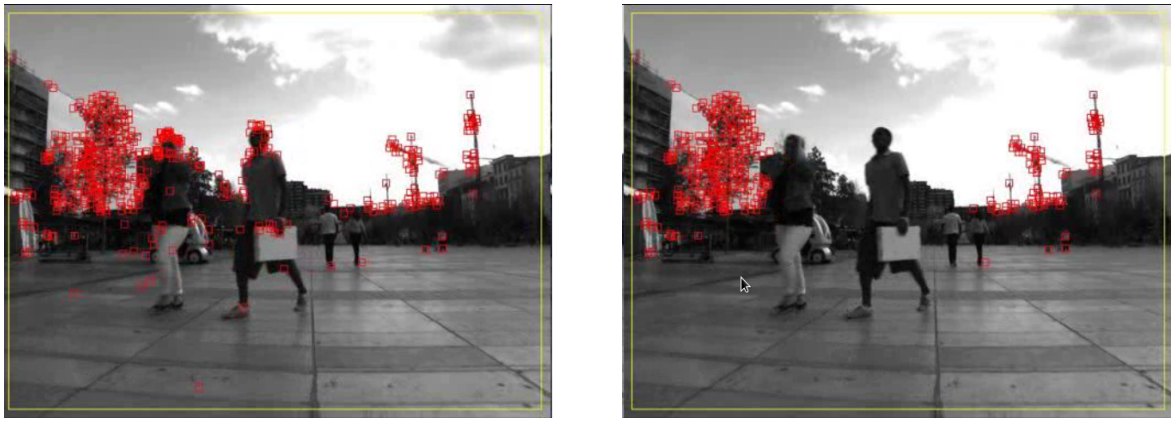
The selection of n pairs $(\mathbf{x}, \mathbf{y})$ depends on the Gaussian distribution, i.e. they are sampled from an isotropic Gaussian distribution distributed at the patch center with variance $\frac{1}{25}S^2$.

In order to decrease noise sensitivity, the whole image is pre-smoothed with Gaussian smoothing before the calculation of descriptor. Size of Gaussian kernel is 9×9 and variance σ is set to value 2.

Two BRIEF descriptors can be easily compared by calculating the Hamming distance between those two descriptors. Hamming distance can be easily and fast performed with XOR operation on modern CPUs.

3.2. Large movement elimination

Features detected on moving objects like cars or persons represent another set of undesirable features. During the tracking those features usually are successfully tracked, i.e. they are moving together with the object, but they should be considered as outliers during the essential matrix calculation. Similarity consistency check can not remove such features, because descriptors of the reference patch and patch in the current image are similar. So, a simple solution named *large movement elimination*, has been implemented with assumption that major number of features have similar displacement (euclidean distance between position of feature in a previous frame and position in a current frame). Features that are far from the camera have the smallest displacement, as feature is closer to the camera, the displacement is larger. Features tracked on the moving object usually have larger displacement then nearest feature to the camera, although that depends on the direction, speed and distance of the moving object. Generally, satisfactory results have been obtained with eliminating the features with displacement much greater than the average displacement of tracked feature set. Thus, after the new positions of features have been obtained using the tracker's translational model, the average displacement is calculated and for each feature is checked if its displacement is not too far from the average. This method successfully removes features on fast moving objects and objects that are moving orthogonal to the direction of camera moving (cf. Figure 3.1).



(a) Without large movement elimination

(b) With large movement elimination

Figure 3.1: An example of the large movement elimination method. With the large movement elimination features are not tracked on persons who are crossing in the front of camera.

3.3. Translational consistency check

Translational consistency check has been also implemented. The approach is similar to the affine consistency check [2], however the affine model of tracking is not available in the OpenCV library, thus the translational mapping is used instead of the affine mapping. Figure 3.2 describes the translational consistency check. A feature is tracked

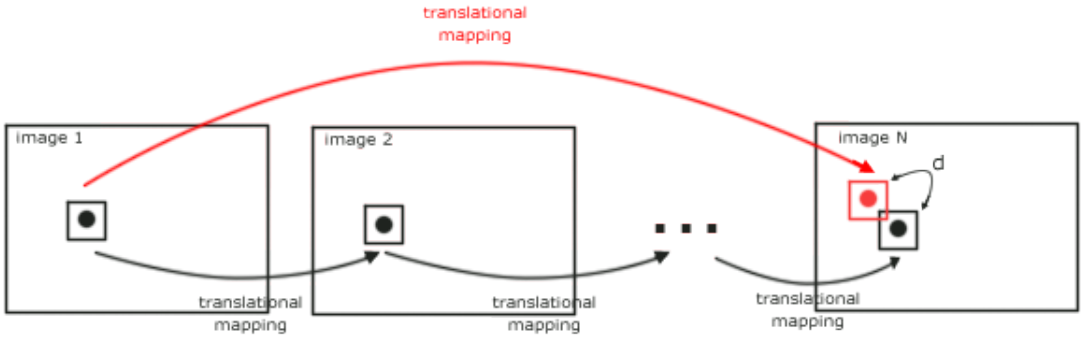


Figure 3.2: Translational consistency check.

in the video sequence frame-to-frame, i.e. for each frame a tracked patch of feature in the previous image is used as a reference template for tracking in the current image. After this step translational consistency check is called, i.e. appearance of feature in first image is used as a reference template and such template is searched in the current image. If the position of feature obtained by translational mapping from the first image is not too far from the position of feature obtained by tracking from the previous image, then the feature is not removed from the feature set. Using this method bad tracks, e.g. feature drifting can be easily detected and such features can be removed from the tracker's feature set.

4. Estimation of the essential matrix

The essential matrix can be easily estimated from a set of correspondences in two camera views with the 8-point algorithm [12]. In order to successfully estimate the essential matrix, the set of feature correspondences has to contain at least eight feature point correspondences that are not co-planar. As it is mentioned in Chapter 2, before any estimation all points are transformed in the calibrated context, i.e. the normalization is performed [11]. Since all correspondences need to satisfy the condition defined by equation:

$$q'Eq = 0, \tag{4.1}$$

where E is the essential matrix 3×3, and q is a 2D point in first view and q' is a correspondent 2D point in second view (points are represented in homogeneous coordinates). The essential matrix is calculated from the homogeneous linear system that is formed from equations for all correspondences. The homogeneous linear system is solved using Singular Value Decomposition [13].

4.1. Elimination of outliers

The main disadvantage of the eight point algorithm is the influence of outlier correspondences and co-planar points that can produce degenerate solution. Outliers may have a large impact on the accuracy of the estimated essential matrix. They can produce the completely failed essential matrix that does not represent the actual geometry between two views. Outliers are produced during the tracking, usually because of a temporary occlusion or a tracking drift (a feature gradually drifts away from the correct position). The improvements of tracker described in Chapter 3 do not guarantee the elimination of all outliers, thus the correspondence set has to be additionally filtered with an iterative method for parameter estimation.

Iterative methods for parameter estimation, like RANSAC and MLESAC have been used for outliers elimination. These methods can eliminate outliers, so the essential

matrix can be estimated only on a set of inliers using the 8-point algorithm [12]. The OpenCV library contains an implementation for the essential matrix estimation which uses RANSAC for the elimination of outliers. Additionally, a custom implementation of MLESAC, based on the paper [22], has been implemented. The MLESAC implementation uses the 7-point algorithm for calculating the hypothesis, i.e. the essential matrix. To determine how feature fits to the hypothesis (the essential matrix) the Sampson error [12] has been used as the error function. Torr and Zisserman [22] propose the reprojection error as the error function. This error function is an overkill for performances, since it requires recovering the camera motion (extraction of the rotation matrix and the translation vector from the essential matrix) in each iteration. The Sampson error (the algebraic approximation of reprojection error) is much less computationally demanding.

A technique for outlier rejection also has been implemented based on the combination of the RANSAC and LMedS (Least Median of Squares). The correspondent set of features in two-views is forwarded into the RANSAC algorithm. The inliers set obtained using RANSAC is passed then in the LMedS algorithm. The output of LMedS is considered as final set of inliers and the 8-point algorithm is performed on such set. This combination showed the most robust results during the learning (mapping) phase, cf.. Section 7.1. During the navigation it is not used because it is slightly too slow for the real-time application.

4.2. Decomposition of essential matrix

A camera motion can be extracted from the essential matrix. Single camera provides 6 DOF of the camera motion (three rotational angles and three components of translation), but only 5 DOF can be estimated. The remaining degree is the scale of the translation. Camera pose can be expressed using the 3×4 camera matrix P which describes the mapping of a pinhole camera from 3D point Q in the world to the correspondent 2D point q in a camera image [12], c. f. Eq. 4.2.

$$q = PQ \tag{4.2}$$

If the coordinate system of the 3D world is centered at the camera coordinate system then mapping is represented with $\mathbf{P} = [\mathbf{I}|\mathbf{0}]$, where $\mathbf{I}$ is a 3×3 identity matrix. If camera coordinate system (x, y, z) and world coordinate system (x', y', z') are not aligned then $\mathbf{P} = [\mathbf{R}|\mathbf{t}]$ represents relation between those two coordinate systems,

where $\mathbf{R}$ is the 3×3 rotation matrix and $\mathbf{t}$ the translational vector.

The camera pose in the current view relevant to coordinate system of a camera pose in the last detected key image k can be expressed with $\mathbf{P_c} = [\mathbf{R_k}|\mathbf{t_k}]$. Rotation matrix $\mathbf{R_k}$ and translational vector $\mathbf{t_k}$ are extracted from essential matrix as it is described in [12], [17]. The decomposition of essential matrix produces four possible solution, the solution where points are in front of camera is a valid one. To determine if a point is in front of camera the triangulation is performed as it is described in [17].

5. Creation of a topological-metric environment map

Creation of a topological-metric environment map (hereinafter "mapping") is a learning phase during which the main goal is to learn a path, so that the robot can autonomously move through the learned environment. The learned path is then used as a reference for the navigation phase.

During the mapping phase features are extracted using the Harris corner detector [10]. In the first frame detector is called. The 2D positions of features together with the image are saved in a visual memory as the first key-image. These features then have been tracked through the video sequence using Lucas-Kanade tracker with the translational model [21]. In each frame two-view geometry is reconstructed between the camera pose in the current frame and the camera pose in the last key-image. If the quality of reconstructed two-view geometry is not acceptable, the image in the current frame together with the feature positions in the image are saved in the visual memory as a new key-image.

This process is repeated in a loop. From an input, i.e. learning video sequence, key-images are extracted. Those key-images are used as reference images during the navigation phase, together with the extracted features in the mapping phase that are used for visual servoing of the robot.

5.1. Scale reconstruction

Translational vector $\mathbf{t}$ from an essential matrix is known only up to scale, it shows only direction of camera motion, thus without knowing the scale it's not possible to determine the exactly camera position. Scale is estimated following the method described in [19], i.e. by requiring that the depth of all 3D points in the previous local world [1]

[1] A local world is space between two adjacent key-images. In this space it is possible to navigate the robot using the correspondent set of features that are visible in both key-images.

and the current local world is the same. In the first local world, it is not possible to determine the scale, obviously there is no previous local world, so scale reconstruction is performed from the second local world.

5.2. Key-image detection

Determining when to save an image as a key-image in visual memory during the mapping is a crucial step. Quality of extracted information from key-images have a significant influence on the quality of the navigation and finally on the visual servoing. During the mapping phase, features are detected using the Harris-corners detector (maximum 850 features), then tracked in each frame. Two-view geometry is reconstructed between the current image and the last detected key-image. If the quality of two-view geometry between those two images is not satisfactory, the current image together with the tracked features is saved in visual memory as a new key-image. In each frame three values are monitored to determine the quality of the reconstructed two-view geometry.

During the tracking, at each frame, two-view geometry is estimated using feature correspondences between the current image and the last detected key-image, i.e. the essential matrix is estimated. Using the reconstructed geometry, the reprojection error is calculated as it is shown in Eq. (5.1).

$$reprojectionError(t) = \frac{\sqrt{\sum_i d(q_i, \hat{q}_i)^2 + d(q'_i, \hat{q'_i})^2}}{\text{number of tracked features in frame } t}, \tag{5.1}$$

In Eq. (5.1) $d(q_i, \hat{q}_i)$ is euclidean distance between points q_i and $\hat{q}_i$. Vector q_i is a 2D point in the first camera view, while q'_i is a correspondent 2D point in the second camera view. Points $\hat{q}_i$ and $\hat{q'_i}$ are projected points with projection matrix on first view and second view, respectively.

A new key-image is taken if less then 75% of features are tracked (live) cf. Eq. (5.6) and if the standard deviation of reprojection error is above threshold `maxStDev` (cf. Eq. (5.5)) or if the ratio between reprojection error in the current frame and the previous frame is above threshold `ratioCurrPrev` (cf. Eq. (5.3)). Also if the number of tracked features is below threshold `minLiveFeatures` (cf. Eq. (5.7)) key-image is detected and saved. Finally, the formulation of condition for detecting key images is expressed with condition Eq. (5.2).

$$[(cond1 \vee cond2 \vee cond3) \wedge cond4] \vee cond5 \tag{5.2}$$

$$cond1 := [\frac{(stdev(reprojectionError(t))}{stdev(reprojectionError(t-1))} > ratioCurrPrev] \tag{5.3}$$

$$cond2 := [stdev(reprojectionError(t)) > maxStDev] \tag{5.4}$$

$$cond3 := [\frac{scale(t)}{scale(t-1)} < 0.8] \tag{5.5}$$

$$cond4 := [\frac{n(t)}{N(t)} < ratioLiveFeatures] \tag{5.6}$$

$$cond5 := [n(t) < minLiveFeatures] \tag{5.7}$$

6. Visual navigation

6.1. Initial localization

When the robot is turned on it's necessary to determine a rough location of the robot in the global space, i.e. between which two key-images the robot is located. To perform this task a current image in camera is compared with all key-images in the visual memory. By finding the most similar key-image it is possible to determine between which two key-images the robot is located. When the nearest preceding and following key-image are found, then fine local localization is performed to determine the exact location of robot.

6.1.1. Rough global localization

The content-based image retrieval system is used to find the most similar key-image to the current image acquired with the camera. More information about the used image retrieval system can be found in [3]. During the mapping phase, after all key-images have been extracted, a content-base image retrieval tree is constructed on all key-images. Construction of the CBIR tree can take several minutes, even hours, i.e. the construction time depends on the number of key-images. The offline construction of the tree enables a fast querying in the navigation phase. Thus, during the initialization in the navigation phase, the content-based image retrieval system is queried with the current image to find the most similar key-image in visual memory.

By knowing the most most similar key-image I_k it is still unknown in which local world the current image is located. The current image could be in the local world (I_{k-1}, I_k) or in the local world (I_k, I_{k+1}). To disambiguate between those two local worlds it is checked if the current image is in front or behind of the camera view in the key-image. The disambiguation is achieved by calculating the geometry between the current image and the key-image. When the geometry is known, then the translation vector is also known. This vector shows in which direction is the camera pose in

the current image (reference to the key-image coordinate system). So by calculating the angle ϕ between the camera direction in the key-image and the translation vector of camera motion in the current image (reference to the key-image) it is possible to determine if the current image is in front or behind of the key-image. If the angle ϕ is less or equal than 90° then the current image is in front of camera view in the key-image (cf. Figure 6.1).

The geometry is estimated using feature correspondences between two images. Feature correspodence are obtained using the wide-baseline matcher described in [3]. In both images features (DoG, MSER) are extracted, then they are matched and correspodent features are used for the geometry estimation.

Whereby the image retrieval system returns a list of results for a query ordered by likelihood. The selection of the most similar key-image is improved by simply comparing the reprojection error of reconstructed geometry with the predefined threshold. The key-image obtained as result of image retrieval query is accepted only if the reprojection error is not larger then the threshold, otherwise the process is repeated with the next best key-image in the list. If all key-images are rejected, then the one with the smallest reprojection error is picked as the initial key-image.

6.1.2. Fine local localization

The exact location is found by performing wide-baseline matching [3] between the current image I_t, the preceding key-image K_k and the following key-image K_{k+1}. Features are extracted from images using DoG and MSER extractors [3]. Matched features are used for estimation of essential matrix between the current image and the preceding key-image $\mathbf{E}_{t:k} = [\mathbf{R}_{t:k}|\mathbf{t}_{t:k}]$, as also between the current image and the following key-image $\mathbf{E}_{t:k+1} = [\mathbf{R}_{t:k+1}|\mathbf{t}_{t:k+1}]$. Correspondent matches in all three images are used to estimate scales of translational vectors $\mathbf{t}_{t:k}$ and $\mathbf{t}_{t:k+1}$. This is performed by requiring that points have the same depth in the world represented with $\mathbf{E}_{t:k}$ or the world with $\mathbf{E}_{t:k+1}$. The estimated scales $s_{t:k}$ and $s_{t:k+1}$ have to be adjusted with the current local world (represented with K_k and K_{k+1}). If $\mathbf{P}_{k+1} = [\mathbf{R}_{k+1}|\mathbf{t}_{k+1}]$ is a camera pose in the key-image K_{k+1} reference to the camera pose in the key-image K_k, then scale can be adjusted with a scale factor represented with Eq. (6.1).

$$s_{t:k:k+1} = \frac{|s_{t:k} \cdot \mathbf{t}_{t:k} + (-1) \cdot s_{t:k+1} \cdot \mathbf{t}_{t:k+1}|}{|\mathbf{t}_{k+1}|} \tag{6.1}$$

Finally, the scaled translational vectors are

$$\begin{aligned} \mathbf{t}_{t:k} &:= s_{t:k:k+1} \cdot s_{t:k} \cdot \mathbf{t}_{t:k} \\ \mathbf{t}_{t:k+1} &:= s_{t:k:k+1} \cdot s_{t:k+1} \cdot \mathbf{t}_{t:k+1}. \end{aligned} \tag{6.2}$$

6.2. Projecting feature positions

After the geometry is successfully reconstructed the features from the current local world in visual memory need to be located in the current image I_t. Features saved in visual memory are represented as 3D points in the coordinate frame of the following key-image K_{k+1}. If camera pose relevant to the camera pose in the key-image K_{k+1} is known those features can be easily projected in the current image. As it is mentioned in section 6.1.2 two two-view geometries are reconstructed, one between I_t and K_k and other between I_t and K_{k+1}. The better one of those two is selected. Using the better geometry features are projected in the current image. The next step is the refining of feature position, since there is no guarantee that reconstructed geometry is the actually one that represents the geometry between views. The refining step is performed with tracker by minimizing the residual between the projected feature and the reference appearance in key-image. References are taken from the key-image which produces a better geometry.

Additionally, the whole refined feature set is filtered with *similarity consistency check* to eliminate invisible features. This can happen if for example during the mapping features had been detected on a car and during the navigation phase the car is not in that place anymore, so these features have to be eliminated. This is performed using tracker's *similarity consistency check*, i.e. by comparing BRIEF descriptor (cf. Section 3.1.1) of the feature in the current image with BRIEF descriptor of the reference appearance in the key-image.

6.3. Tracking

After the features from visual memory have been found, those features are tracked frame-to-frame using the point feature tracker, cf. Section 3, with enabled *large movement elimination*. The *translational consistency check* and the *similarity consistency check* are disabled because they may eliminate too much inliers during the navigation. This is not a problem for the mapping phase because new features are detected if the number of correspondences is less than Eq. (5.6), but in the navigation if there is less

then eight correspondences, then the essential matrix can not be computed. Thus, the navigation would fail.

However, the situation with too few points for the essential matrix calculation can not be discarded, there is a possibility of occurring such exception during the navigation. Thus, if the navigation fails because of too few features, the *similarity error threshold* for *similarity consistency check* is scaled by factor two and features from visual memory are again projected and refined on the current image, as it is described in Section 6.2, but using the geometry from the previous frame and scaled *similarity error threshold*.

After the tracking, two geometries (the geometry between the current image and the preceding key-image and the geometry between the current image and the following key-image) are refreshed (re-estimated) with the new positions of features obtained by the tracker. Better geometry of those two is used as representative one for a localization.

6.4. Visual servoing

Robot manipulation has been performed in a visual servoing processing loop [6] using correspondences between features in the current image acquired in real-time and features in the following key-image from visual memory.

The steering angle is calculated as it is shown in Eq. (6.3).

$$\phi = -\lambda(\bar{x_t} - \bar{x_{i+1}}) \tag{6.3}$$

Where $(x_t, y_t) \in X_t$ are feature coordinates in the current image, $(x_{i+1}, y_{i+1}) \in X_{i+1}$ are coordinates of features in the following key-image and $\lambda \in \mathbb{R}^+$ is a predefined gain.

6.5. Transition

For the navigation phase, decision when to switch the local world has significant influence on the quality of navigation. Two approaches are implemented and tested: *the geometric method* and *the second moment order method*.

6.5.1. The geometric method

The geometric method relies on the reconstructed geometry. After the geometry is reconstructed it is possible to determine the position of camera in the current view relative to following key-image. Transition occurs if the camera pose is in front of camera view in the following key-image K_{k+1}. It is calculated by measuring the angle ϕ between the vector of camera direction in the following key-image K_{k+1}(cf. the green vector in Figure 6.1) and the vector that connects points $p(K_{k+1})$ and $p(I_t)$ (cf. the black vector in Figure 6.1). Where $p(K_{k+1})$ is a geometric 2D position of camera pose in current image I_t and $p(I_t)$ is a geometric 2D position of camera pose in the key-image K_{k+1}. Transition is performed only if the angle ϕ is not above 90°.

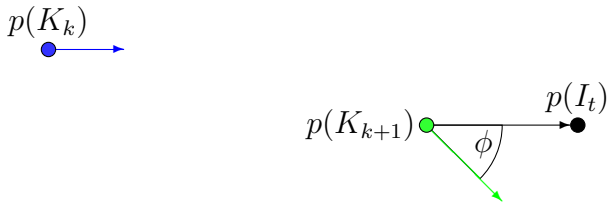


Figure 6.1: An example of the detection of transition. The blue vector shows the direction of camera in the preceding key-image K_k, while the green vector shows the direction of camera in the following key-image. Function $p(\cdot)$ returns the 2D position in the map from the bird's-eye view. The black vector connects 2D position of the camera pose in the current image with the camera pose in the following key-image. The transition is performed, since the angle ϕ between the green vector and the black vector has value of 45°, i.e. less than or equal to 90°.

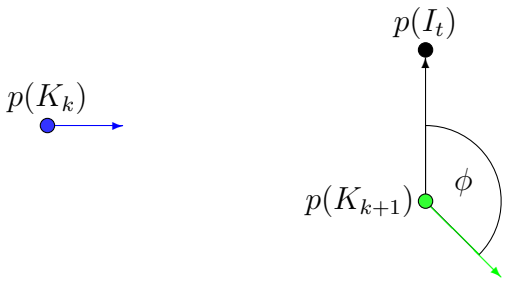


Figure 6.2: The blue vector shows the direction of camera in the preceding key-image K_k, while the green vector shows the direction of camera in the following key-image. Function $p(\cdot)$ returns the 2D position in the map from the bird's-eye view. The black vector connects 2D position of the camera pose in the current image with the camera pose in the following key-image. The transition is not performed, since the angle ϕ between the green vector and the black vector has value of 135°, i.e. greater than 90°.

6.5.2. The second moment order method

The second moment order method does not need any information about geometry between two views. It relies only on 2D positions of features in current view and the following key-image. Thus, the transition can be performed by measuring the error between second moment of features in the current image and second moment of correspondent features in the following key-image. The moment of feature set $S_{features}(I)$ in image I is defined as Eq. 6.4, where N is the number of features $|S_{features}(I)|$ and $(\bar{x}, \bar{y})$ is the center of gravity, then the second moment is Eq. 6.5. Finally, the error is defined as it is shown in 6.6. The transition is performed if $m_{error}(I_t, K_{k+1})$ is near zero.

$$M_{ij}(S_{features}(I)) = \sum_{k=1}^{N} (x_k - \bar{x})^i (y_k - \bar{y})^j \tag{6.4}$$

$$m_{2nd}(S_{features}(I)) = M_{20}(S_{features}(I)) + M_{02}(S_{features}(I)) \tag{6.5}$$

$$m_{error}(I_t, K_{k+1}) = \frac{[m_{2nd}(S_{features}(I_t)) - m_{2nd}(S_{features}(K_{k+1}))]^2}{N^2} \tag{6.6}$$

7. Results

7.1. Mapping

Six test examples have been ran to compare and evaluate different outlier rejection techniques during the mapping phase. Maps in all examples are learned on the same video sequence `garage/2011.12.02-15h/aller`. The video sequence has been chosen because of presence of three road curves, lighting changes and presence of other urban situations. The groundtruth of the scene is depicted in Figure 7.1. It is important to observe the behaviour of mapping algorithm in road curves, because the robot should not have problems with turnings. For each example, a topological map of robot position at each frame (Figures 7.2a, 7.2b, 7.2c, 7.5a, 7.5b, 7.5c), a graph that shows the percentage of detected outliers (Figures 7.3a, 7.3b, 7.3c, 7.6a, 7.6b, 7.6c) and a graph with number of frames between two key-images (Figures 7.4a, 7.4b, 7.4c, 7.7a, 7.7b, 7.7c) are presented.

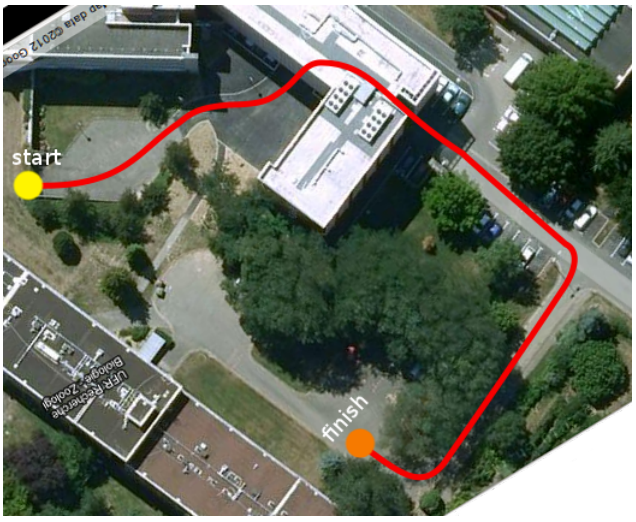


Figure 7.1: The groundtruth topological map of the video sequence `garage/2011.12.02-15h/aller`

Each example can be easily executed within the executable file. All what is needed

is to set the parameter `-a` to `test` and the parameter `-e` to the number of desired example in the configuration file.

Thresholds $ratioCurrPrev$ Eq. (5.3) and $ratioLiveFeatures$ Eq. (5.6) are set to 1.25 and 0.75, respectively. In examples 1,2 and 3 threshold $minLiveFeatures$ Eq. (5.7) has value 400, while in examples 4,5 and 6 the threshold is set to 60.

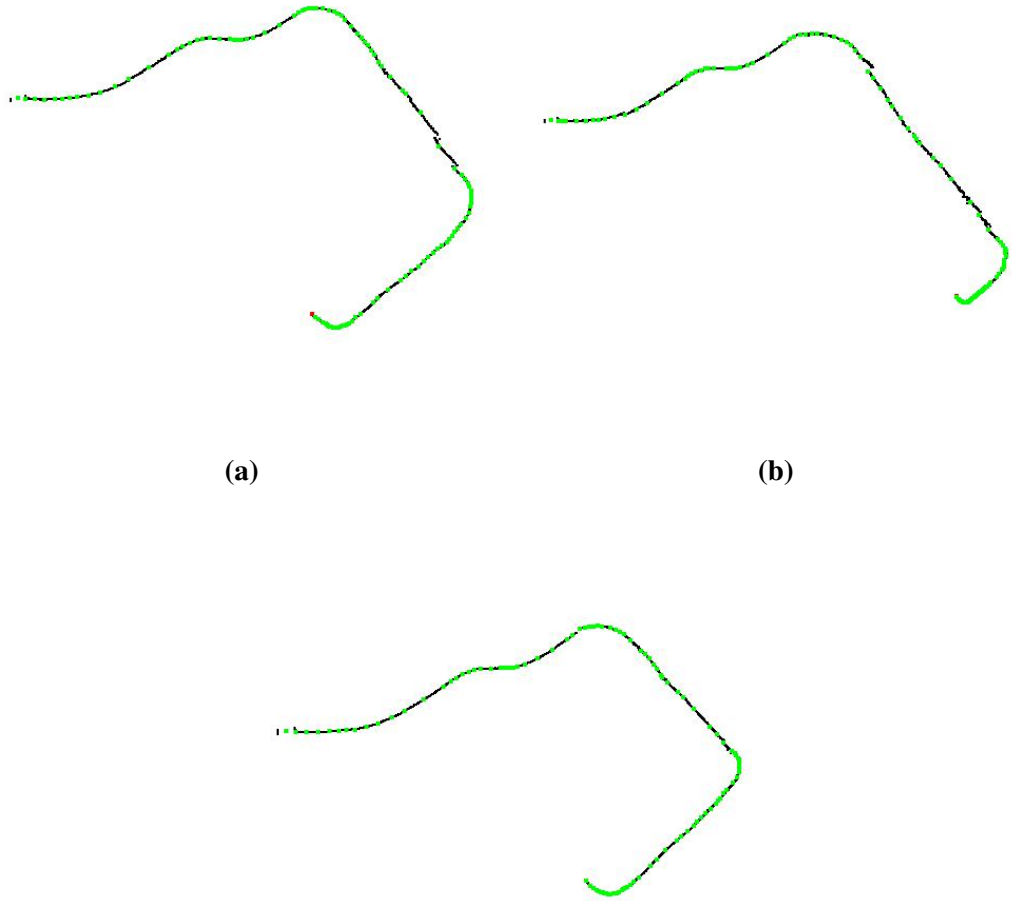
$$rOutliers(t) = \frac{\text{number of outliers in frame } t \text{ detected with algorithm for outlier elimination}}{\text{number of tracked features in frame } t}, \tag{7.1}$$

A topological map shows bird's-eye view 2D position of the robot in each frame. The green point represents the position of key-image.

In graphs (Figure 7.3a, Figure 7.3b, Figure 7.3c, Figure 7.6a, Figure 7.6b, Figure 7.6c), for each frame value of Eq. (7.1) is plotted. In the upper right corner in each graph the number of detected key-images is noted. The x-axis in the graphs represents the number of frame and the y-axis represents the observed value.

In Figures 7.4a, 7.4b, 7.4c, 7.7a, 7.7b and 7.7c are represented graphs that present the number of processed frames between two adjacent key-images, i.e. how much frames has been processed until a new key-image is detected. The blue dashed line represents the average of processed frames until the key-image detection.

Examples 1, 2, 3: minLiveFeatures = 400



(a) **(b)**

(c)

Figure 7.2: The three subfigures show topological maps constructed during the learning on the sequence `garage/2011.12.02-15h/aller` ($minLiveFeatures = 400$). The subfigure (a) shows the topological map after the learning has been performed with RANSAC, while in the subfigure (b) MLESAC has been used. In the subfigure (c) RANSAC + LMedS has been used for the outlier rejection. The reconstructed path in the subfigure (a) is similar to the groundtruth depicted in Figure 7.1, but because of the path disruptions it is not good as the reconstructed path in the subfigure (c).

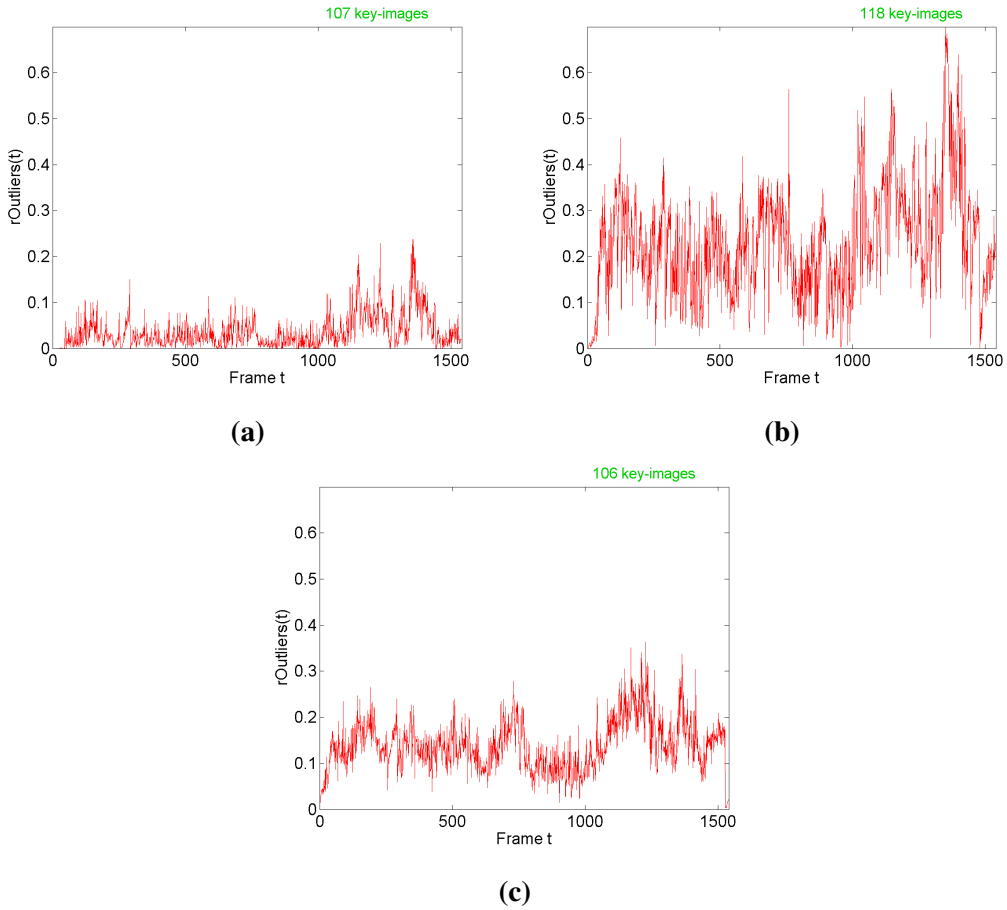
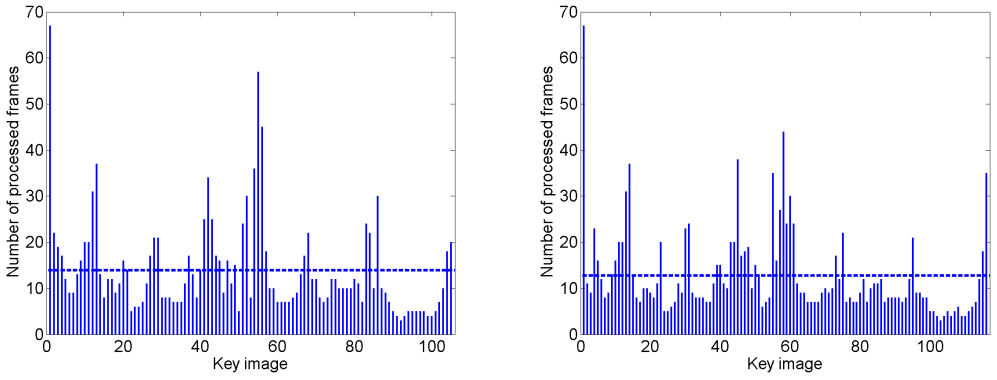
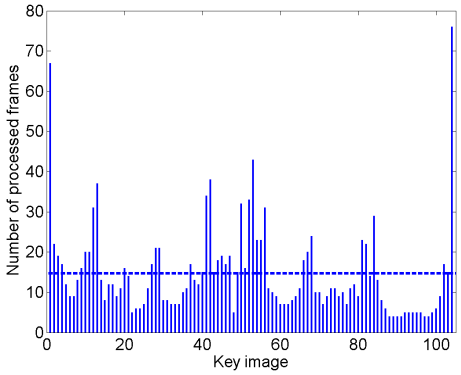


Figure 7.3: The graphs (a-c) display value of Eq. (7.1) in each frame of the learning sequence `garage/2011.12.02-15h/aller` ($minLiveFeatures = 400$). In the subfigure (a) RANSAC has been used for the outlier rejection, while in the subfigure (b) this task has been done by MLESAC and in the subfigure (c) RANSAC + LMedS has been used. It can be noticed how MLESAC (b) rejects too much features, in frames 1300 - 1400 more than 60% of features have been rejected. Also it produces 118 key-images, while RANSAC and RANSAC + LMedS produce 106 - 107 key-images for the same path.



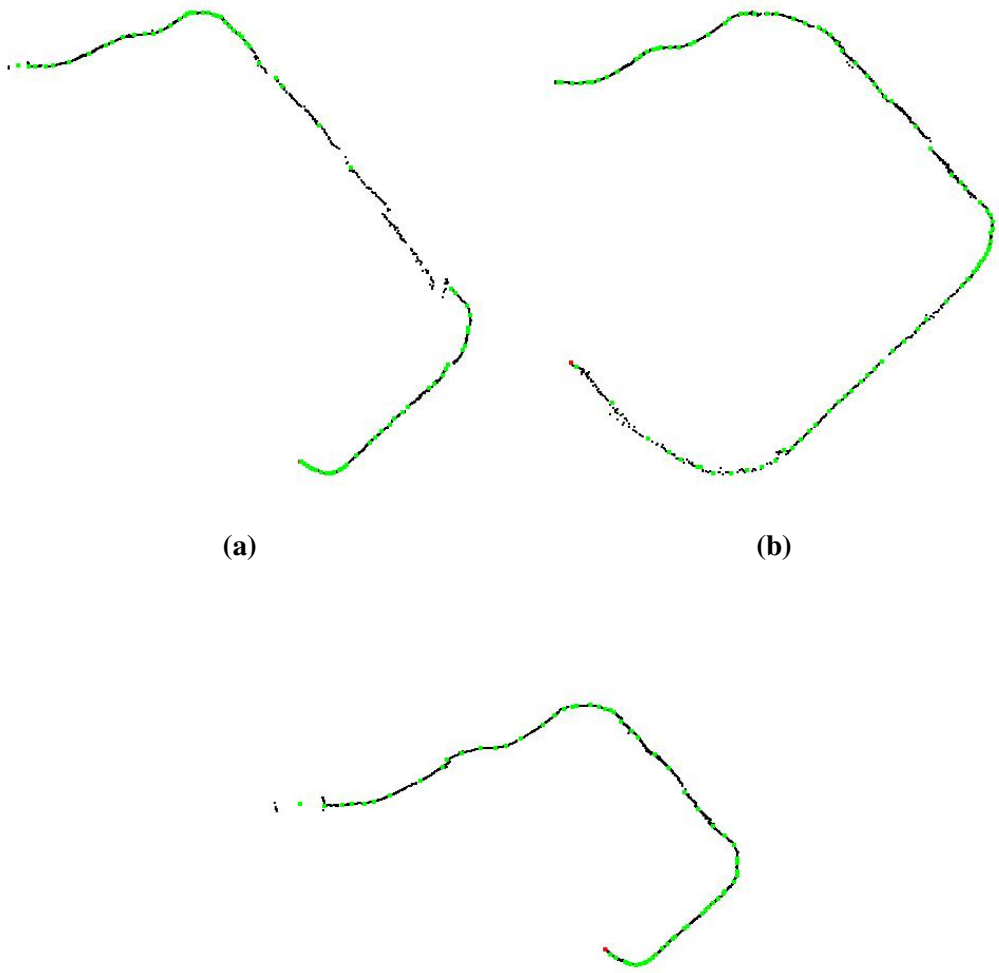
(a) **(b)**



(c)

Figure 7.4: The subfigures show how many frames is processed before the new key-image is detected ($minLiveFeatures = 400$). In the subfigure (a) RANSAC has been used, while in the subfigure (b) MLESAC has been used. Finally in the subfigure (c) RANSAC + LMedS has been used as the outlier rejection technique. In all subfigures the horizontal dashed line shows the average number of processed frames for the key-image detection. In the subfigure (a) it is 13, in (b) 12 and in (c) 14.

Examples 4, 5, 6: minLiveFeatures = 60



(a) **(b)**

(c)

Figure 7.5: The three subfigures show topological maps constructed during the learning on the sequence `garage/2011.12.02-15h/aller`. The threshold $minLiveFeatures$ is set to value 60. The subfigure (a) shows the topological map after the learning has been performed with RANSAC, while in the subfigure (b) MLESAC has been used. In the subfigure (c) RANSAC + LMedS has been used for the outlier rejection. The reconstructed path in the subfigure (c) is the most similar to the groundtruth depicted in Figure 7.1. The reconstructed paths in (a) and (b) do not represent the actual path with the satisfactory level of quality.

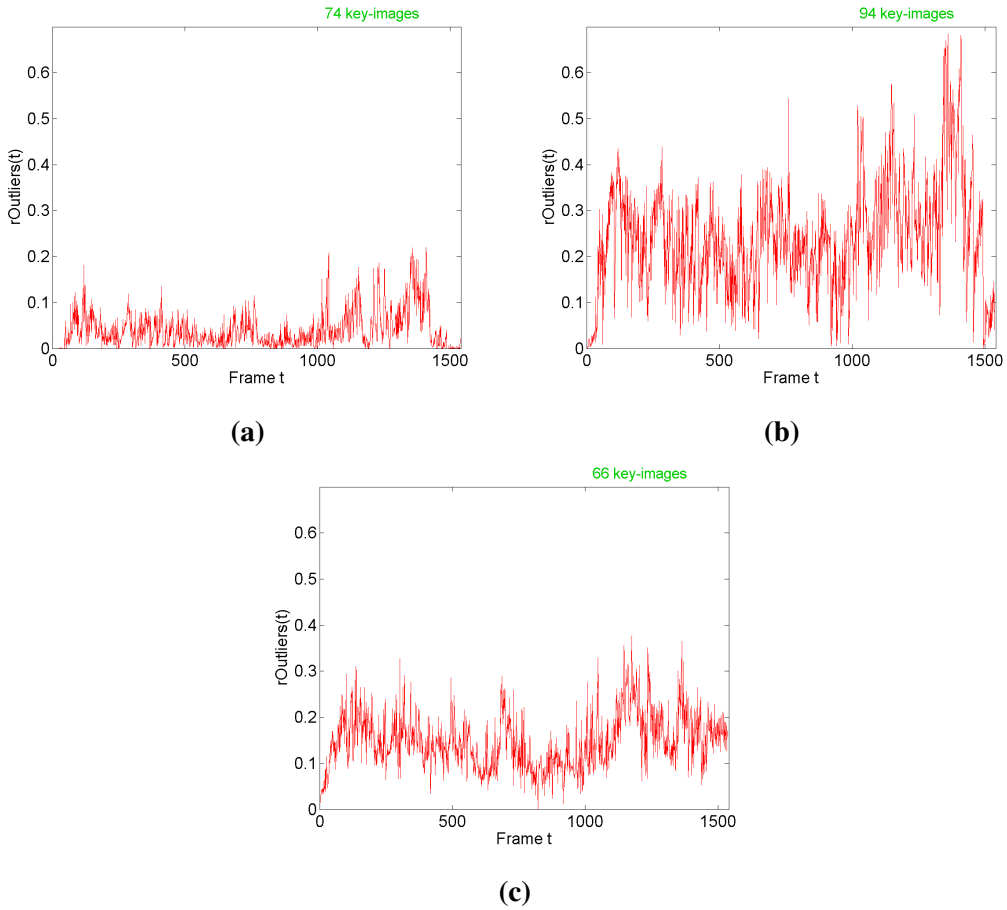


Figure 7.6: The graphs (a-c) display value of Eq. (7.1) in each frame of the learning sequence `garage/2011.12.02-15h/aller`. The threshold $minLiveFeatures$ is set to value 60. In the subfigure (a) RANSAC has been used for the outlier rejection, while in the subfigure (b) this task has been done by MLESAC and in the subfigure (c) RANSAC + LMedS has been used. It can be noticed how MLESAC (b) rejects too much features, in frames 1300 - 1400 more than 60% of features have been rejected. Also it produces 94 key-images, while RANSAC and RANSAC + LMedS produce 74 and 66 key-images, respectively.

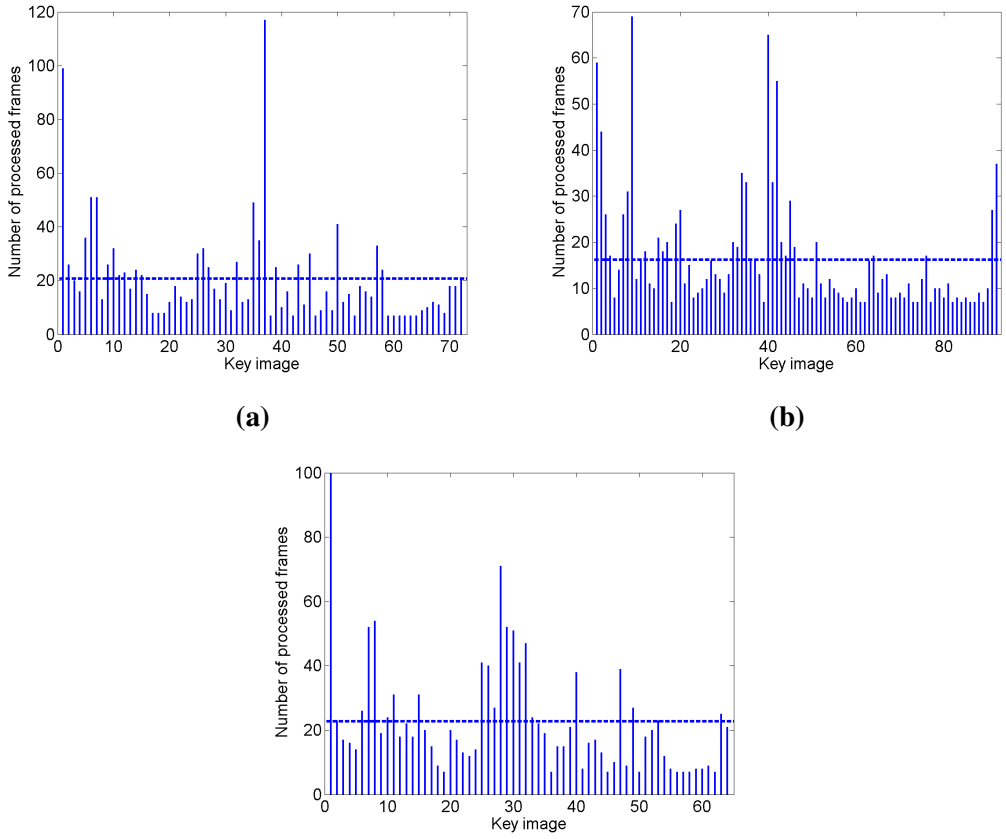


Figure 7.7: The subfigures show how many frames is processed before the new key-image is detected. The threshold $minLiveFeatures$ is set to value 60. In the subfigure (a) RANSAC has been used, while in the subfigure (b) MLESAC has been used. Finally in the subfigure (c) RANSAC + LMedS has been used as the outlier rejection technique. In all subfigures the horizontal dashed line shows the average number of processed frames for the key-image detection. In the subfigure (a) it is 20, in (b) 17 and in (c) 22.

The graphs (Figure 7.3 and Figure 7.6) are demonstrating that the combination of RANSAC and LMedS produces the best topological map (comparing with the groundtruth depicted in Figure 7.1). The RANSAC simply does not eliminate all outliers, as can be seen from Figures 7.3a and 7.6a at average 10% of features has been detected as outliers, while the MLESAC obviously eliminates some inliers because in some cases, for example near frames 1300 - 1400 in Figure 7.3b and 7.6b, it eliminates more than 60% of features, while the RANSAC + LMeds in the same span of frames eliminates from 20 - 30% features. As can be seen by comparing Figure 7.2c and 7.2b with the groundtruth depicted in Figure 7.1, as also Figure 7.5c and 7.5b, RANSAC + LMedS produces a better topological map than MLESAC. The MLESAC also produces more key-images, for example if $minLiveFeatures$ is set to 400 then it produces 118 of them, if $minLiveFeatures$ is set to 60 then it produces 94 key-images, while the RANSAC + LMedS in first case produces 106 and in the other case 66 of them, which is approximately the same if only RANSAC is used.

The best results are obtained when $minLiveFeatures$ is set to 400. This can be seen by comparing the topological maps in Figure 7.2 and Figure 7.5 with the groundtruth depicted in Figure 7.1. It should be taken in consideration that threshold $minLiveFeatures = 60$ produces less key-images, at average every 22^{nd} frame is a new key-image(cf. Figure 7.7), while with the value 400 every 14^{th} frame is a new key-image (cf. Figure 7.4). This difference is not too large and the quality of the topological map is more important than number of key-images, so the threshold $minLiveFeatures = 400$ has been chosen as optimal.

Also experiments with the navigation showed that setting the value of threshold $minLiveFeatures$ to 400 produces a way better navigation, so this threshold has been chosen as optimal. Larger values of $minLiveFeatures$ produce too much key-images, while with smaller values the navigation is not satisfactory.

7.2. Navigation

The navigation phase has been performed on the video sequence `garage/2011.12.05-14h/aller`, while the mapping has been performed on the video sequence `garage/2011.12.02-15h/aller` acquired three days earlier. In the navigation sequence some cars are moved (cf. Figure 7.9), some are absent while they have been there during the learning (cf. Figure 7.8). New cars appeared. As also significantly changes in lighting can be noticed. These all differences between the learning and the navigation sequence are interest dataset for testing the developed solution.

(a) **(b)**

Figure 7.8: The small white van is absent in the navigation sequence `garage/2011.12.05-14h/aller` at frame #355 (b), while it is present in the learning sequence `garage/2011.12.02-15h/aller` at frame #386 (a).

(a) **(b)**

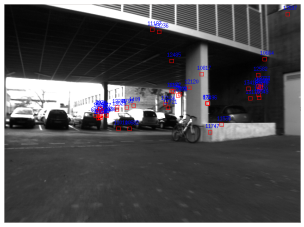
Figure 7.9: The large white van is relocated in the navigation sequence (b). Also shadows are different in the learning sequence `garage/2011.12.02-15h/aller` at frame #575 (a) and in the navigation sequence `garage/2011.12.05-14h/aller` at frame #547 (b).

7.2.1. Transition

Experiments with second moment of features as tool to determine the transition showed that this method is not adequate. Simply, it does not work in environment where few distant points are present in the current image (points that are distant from the camera in the 3D world), for example when a wall is in front of the camera.

Figure 7.10 shows the environment where the transition has failed. As can be seen in Figure 7.11 the error measure mu_{error}, cf. Eq. (6.6), converge until 14th key-image, after which the measure diverge and never falls under 0.05 (threshold for detecting transition). In the graph after frame #266 mu_{error} has value much greater than one. In the graph the red line detonates the threshold for detecting transition, the blue curve detonates the mu_{error} value and the green dashed vertical line detonates the frame

(a) The preceding key-image



(b) Current image in the camera

(c) The following key-image

Figure 7.10: An example of missed transition near frame #266. The current image (b) is not located in the current local world defined with the preceding key-image (a) and the following key-image (c).

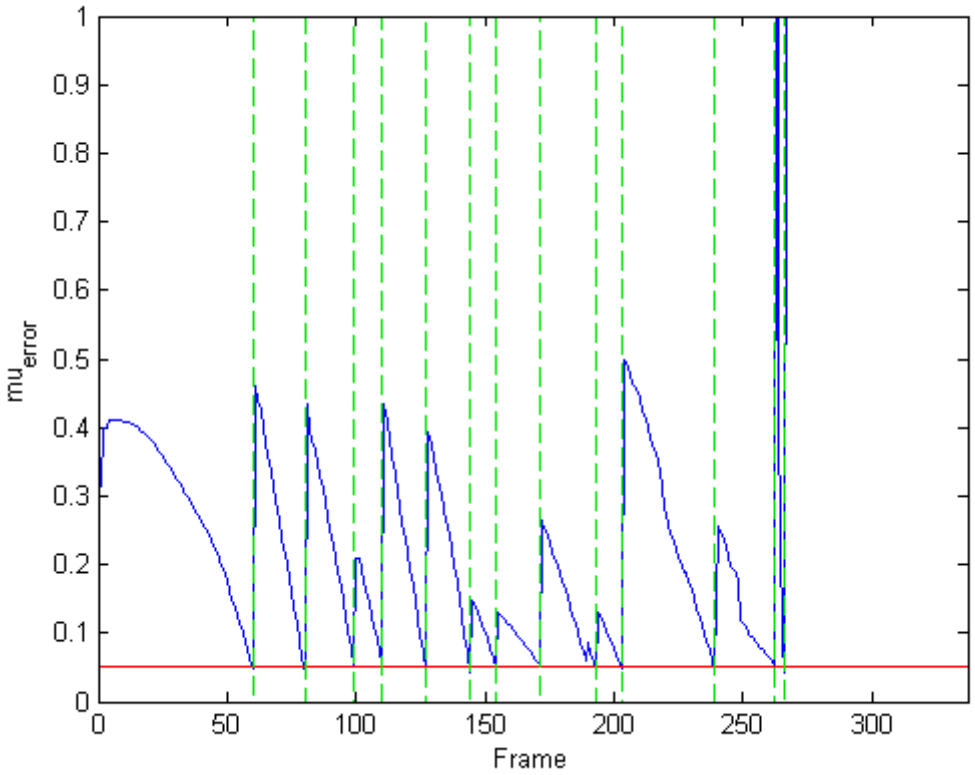
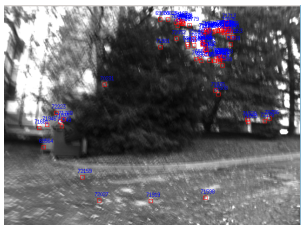
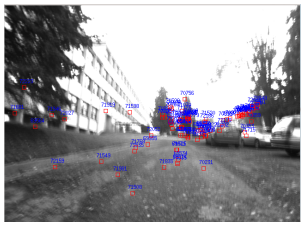


Figure 7.11: Graph with plotted mu_{error} in each frame. After frame #266 mu_{error} does not converge and never falls under the value of threshold for detecting the transition.

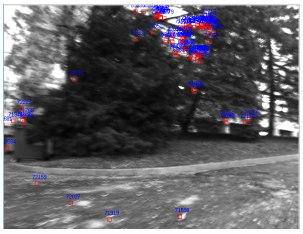
where transition has been performed.



(a) The preceding key-image



(b) Current image in the camera



(c) The following key-image

Figure 7.12: An example of missed transition near frame #1350 in the navigation sequence `garage/2011.12.05-14h/aller`. The current image (b) is not located in the current local world defined with the preceding key-image (a) and the following key-image (c).

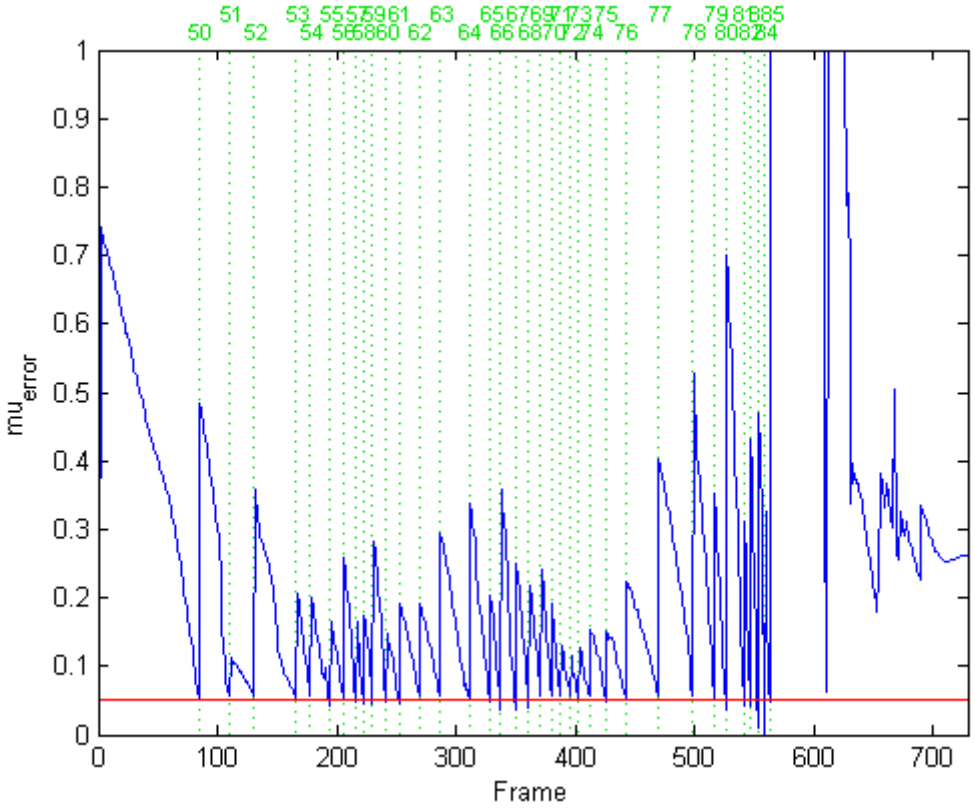


Figure 7.13: Graph with plotted mu_{error}. After frame #1350 mu_{error} does not converge and never falls under the value of threshold for detecting the transition.

The second example of failed transition can be noticed around frame #1350 (cf. Figure 7.13 and Figure 7.12). The problem is same as in previous example, simply features are too close to the camera, so mu_{error} diverges, cf. Figure 7.13. Note that navigation

has been started from the key-image #49.

As the *second moment of feature set method* did not produce expected behaviour, the geometric method has been chosen for the transition determination, because such problems have not been noticed using the topological method.

7.3. A navigation experiment

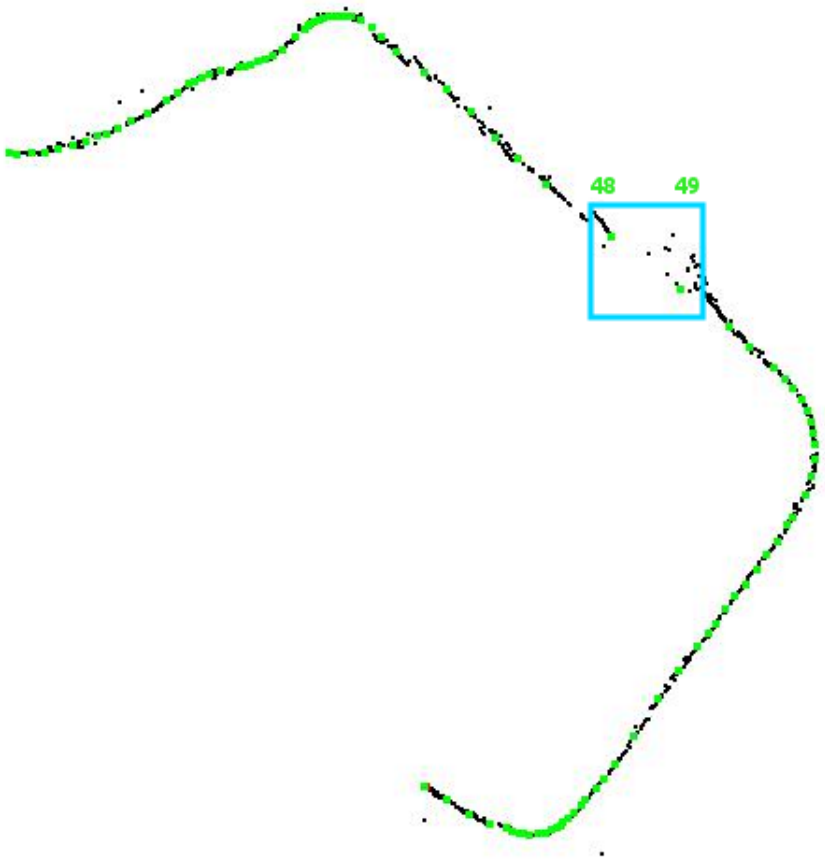
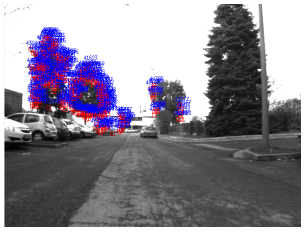


Figure 7.14: The topological map of reconstructed after the navigation. The mapping has been performed on the video sequence `garage/2011.12.02-15h/aller`, while the navigation has been performed on the sequence `garage/2011.12.05-14h/aller`. The key-image positions (the green dots) have been loaded from the visual memory. The black dots show the reconstructed positions of the robot during the navigation. In the local world between the key-image #48 and #49 (the blue rectangle), some difficulties with the navigation can be noticed.

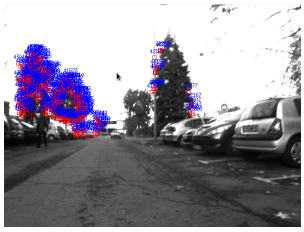
Figure 7.14 shows 2D positions of robot at each frame during the navigation. The

2D positions have been reconstructed from the calculated essential matrix and they represent the quality of robot localization. The green points represent the location of key-images obtained during the learning.

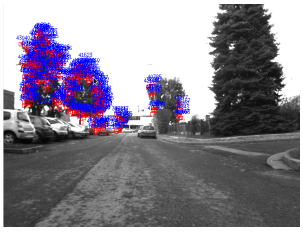
As can be seen in Figure 7.14, the reconstructed path is satisfactory, so navigation is performed well. Between key-images #48 and #49 (the blue rectangle in Figure 7.14) the navigation has not been performed with the satisfactory level. Namely, just after the transition from the local world #47 - #48 in the local world #48 - #49 the robot has turned intensively right, so large number of features drifted from the correct location, which had the effect on the reconstructed geometry and finally on the feature projection and the refining step during the transition. The badly reconstructed geometry, later in the next frame, caused again another transition, as can been seen in Figure 7.15. However, although the transition is performed too early, large number features are correctly projected in the current image and the navigation has been continued correctly, that can be noticed in Figure 7.14 (the final key-image has been successfully reached).



(a) The preceding key-image (#50)



(b) The current image



(c) The following key-image (#51)

Figure 7.15: In the local world defined with the key-images #48 and #49 the current image (b) (frame #849 in `garage/2011.12.02-15h/aller`) performed the transition. This transition produced a bad geometry, so features have been incorrectly projected, which resulted with another transition. The result of these transitions has been the incorrect local world defined with key-images #50 (a) and #51 (c), while they should be #48 and #49.

In the frame #1360 (Figures 7.16, 7.17 and 7.18) during the navigation 12 features have been successfully projected in the image, but RANSAC found 5 outliers and essential matrix could not be calculated. Two features from the orange rectangle and three features from the green rectangle in Figure 7.17 are considered as outliers. So, error recovery has been performed as it is described in Section 6.3. Features from the visual memory are projected and refined again using the geometry from the previous frame. As can be seen from Figure 7.14 the recovery has been performed successfully and the final key-image has been reached.

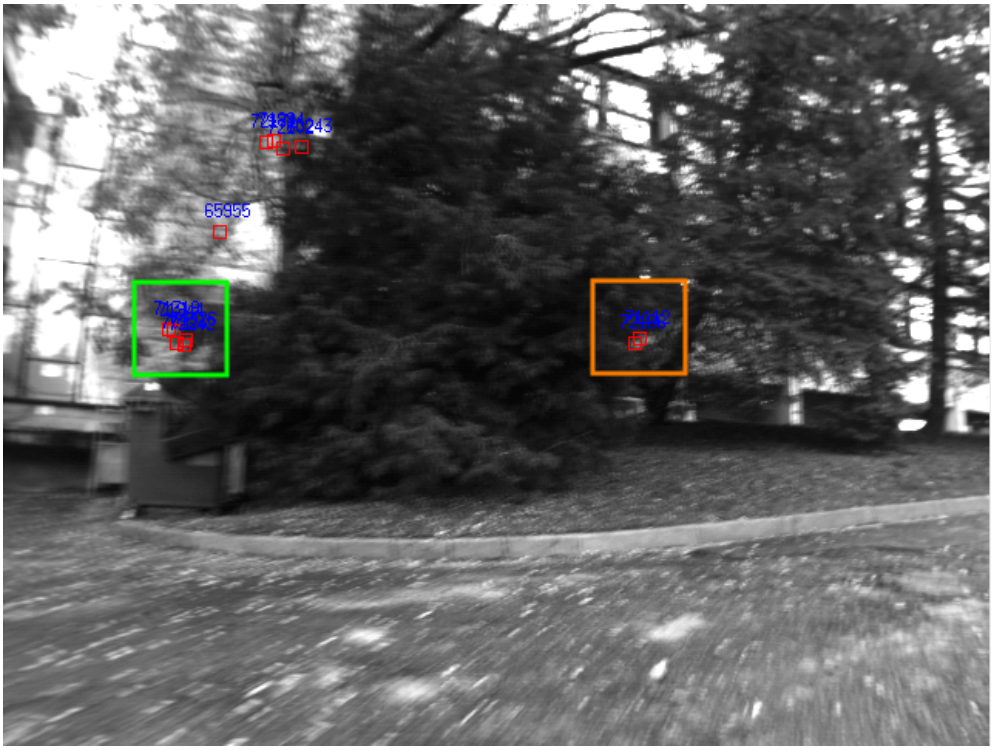


Figure 7.16: The preceding key-image (#84)

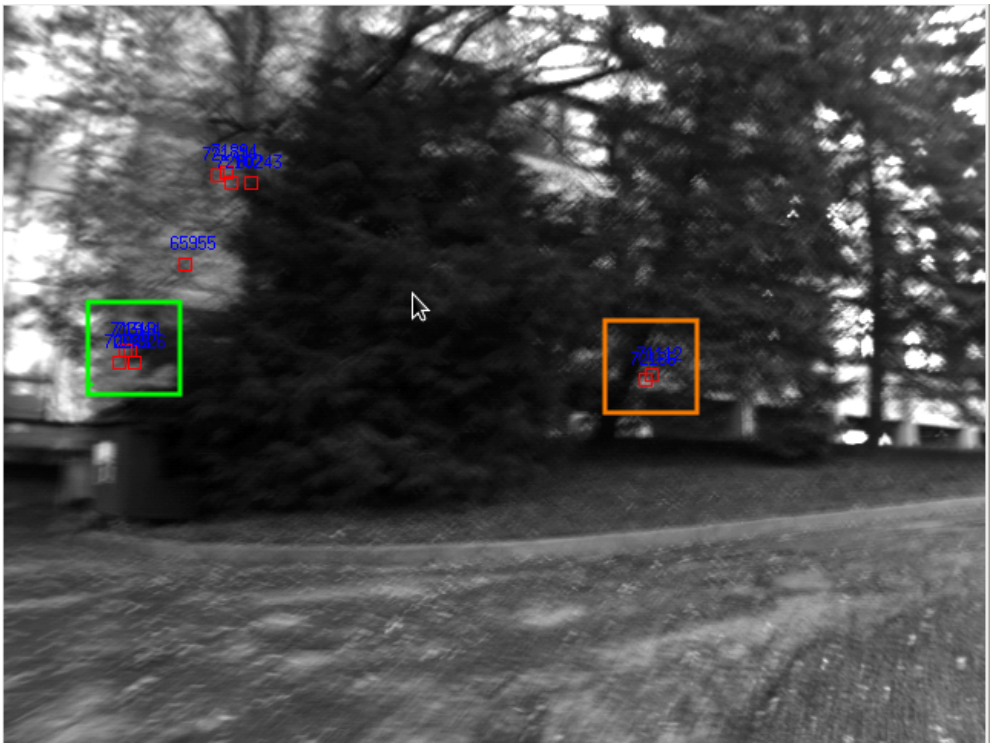


Figure 7.17: The current image

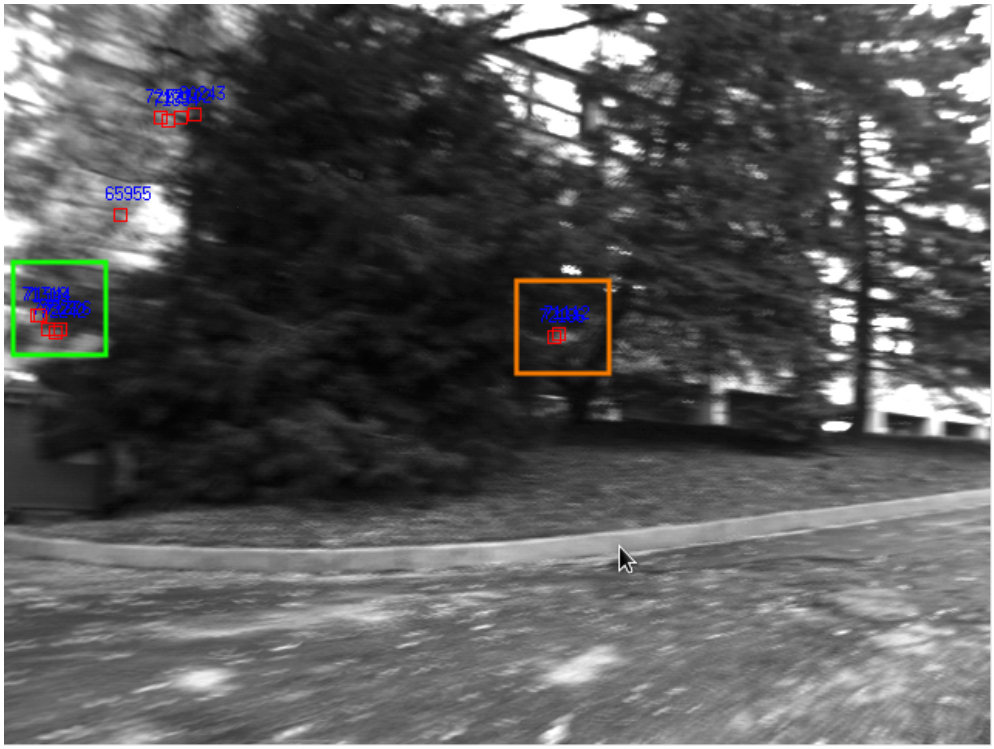


Figure 7.18: The following key-image (#85)

7.4. Testing on the robot

The autonomous navigation is demonstrated in video `nav-06-21.avi`. The learning has been performed on the video sequence that has been acquired two hours before the navigation. As the video demonstrates, the robot successfully navigated autonomously and reached the final position.

The second experiment involves more complex situations. Two learning sequences have been acquired, one `garage/2012.06.27-18h/aller` and the other `garage/2012.06.27-18h/retour` during the returning into the garage. They have been acquired on 27^{th} June 2012 at 7PM, when the parking lot was almost empty.

The topological map after the performed learning on the video sequence `garage/2012.06.27-18h/retour` is showed in Figure 7.19.

Changing light conditions, cf. Figure 7.20, produced the space between key-images #4 and #5 (the blue rectangle Figure 7.20). Namely, in frame #58 sun rays have been hidden by trees and then in next frame #59, as the robot moves forward, the sun rays suddenly appear on the image, cf. 7.20. Almost all features drifted from the correct location, which had the impact on the geometry estimation. Note that the *large movement elimination* method is not helpful in such situations, because almost all features

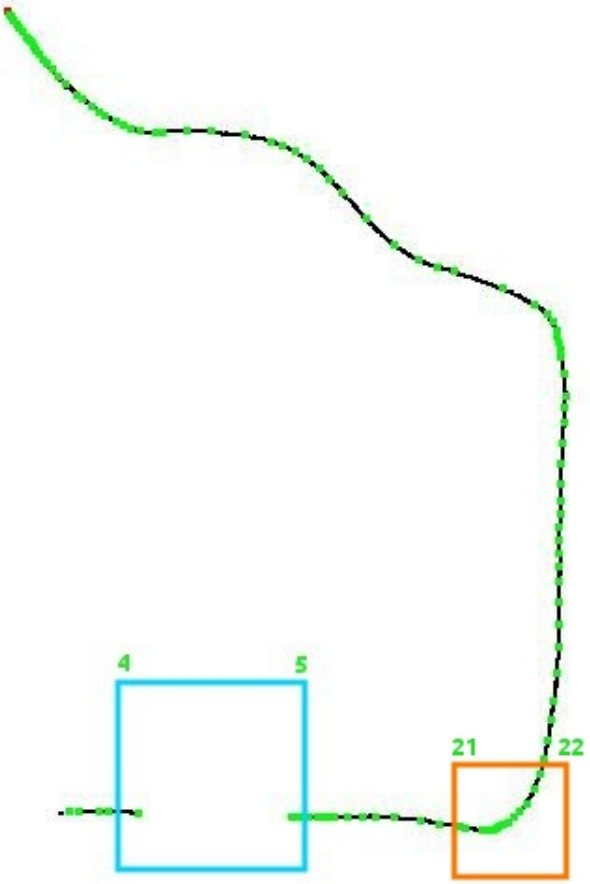


Figure 7.19: The reconstructed topological map of the learning video sequence `garage/2012.06.27-18h/retour`. Between key-images #4 and #5 (the blue rectangle) sunlight produced the bad reconstruction of the path.

in the image drifted.

Another problematic situation was passing of car just in front of the camera during the mapping phase, cf. 7.21. The *large movement elimination* method successfully eliminated features on the car (cf. 7.21), so this event had not influence on the geometry, as can be seen from Figure 7.19 (the orange rectangle).

The navigation experiments have been performed the next day around noon, when the parking lot was full of cars. As can be seen in video `nav-06-28-12h-failed.avi` the robot navigated autonomously fifty meters. Under the building, where the lighting was bad, the emergency system for instant stopping has been used, because the robot suddenly turned right and almost hit the parked car. Failed navigation can be justified with excessive changes in the environment, cf. Figure 7.22 and Figure 7.23.

(a) Frame 58

(b) Frame 59

Figure 7.20: Intensive change of light between two successive frames (#58 (a) and #59 (b)) in the learning sequence `garage/2012.06.27-18h/retour`. This event produced the space between key-images #4 and #5 depicted in the blue rectangle in Figure 7.19.

.

(a) Frame 352

(b) Frame 361

Figure 7.21: The passing of car in the learning sequence `garage/2012.06.27-18h/retour` between frames #352 - #361. Features have not been tracked on the car. As can be seen in (a) features located on the top of the car have been rejected (b). These features have been rejected with the large movement elimination method. The passing of car had not influence on the geometry as can be seen from the reconstructed path in the orange rectangle in Figure 7.19. Namely, in the orange rectangle there are not present any anomalies.

Simply, the current appearance of the environment is too different from the learned one, so implemented solution had difficulties with the navigation.

In order to justify this thesis, the experiment has been repeated but from the other initial location, where changes in the environment appearance have not been so intensive. The robot has been turned on at location where the parking lot is not visible (near key-image #59). As can be seen in video `nav-06-28-12h-short.avi` the robot

(a) The learning sequence

(b) The navigation sequence

Figure 7.22: The environment appearance has been changed between the learning sequence `garage/2012.06.27-18h/retour` at frame #694 (a) acquired on 27^{th} June at 7PM and the navigation `nav-06-28-12h-failed.avi` (b) that has been performed on 28^{th} June at 12PM.

(a) Learning

(b) Navigation

Figure 7.23: The environment appearance between the learning sequence `garage/2012.06.27-18h/retour` at frame #764 (a) acquired on 27^{th} June at 7PM and the navigation `nav-06-28-12h-failed.avi` (b) that has been performed on 28^{th} June at 12PM.

has been successfully navigated.

The same day at 5PM the third experiment has been performed using the same learned map database (learned on the video sequence `garage/2012.06.27-18h/retour`). As can be seen in `nav-06-28-17h-long.avi` the robot has been successfully navigated on the whole path and autonomously parked in the garage. In order to achieve the parking in the garage, the obstacle avoidance system had to be turned off in the key-images that represent entering in the garage, because the system does not allow entering in the garage. Note that the navigation is performed successfully at

5PM, but not successfully at the noon. The reason is that the environment appearance at 5PM has been similar to the learned environment appearance, mainly because of the absence of the cars in the parking lot and sun orientation.

8. Conclusion

The implemented system for autonomous robot navigation supported autonomous navigation in the real experiments with a mobile robotic car. The experiments showed that the system is robust and able to navigate the robot even in the environment appearance changed from the learned one. The improvements of tracker, such as similarity consistency check, translational consistency check, and large movement elimination showed how simple additions can improve the performance of translational tracker. Thus, the affine model tracker, the 5-pt relative pose algorithm and MLESAC used in [19] can be replaced with simpler algorithms such as the translational model tracker, the 8-pt relative pose algorithm and RANSAC. Please note that tracker in [19] uses the translational model of tracker for frame-to-frame tracking, but the isotropic scaling and the affine photometry have been used for the feature refining. The implemented solution produces twice more key-images than the solution described in [19]. That is not commendable because with fewer key-images it is easier to find the nearest key-image using the image retrieval during the initialization. The exact comparison with the solution described in [19] has not been performed due to the lack of time, but it would be an interesting direction for future work.

In future, some improvements can be implemented as well. For example in the implemented solution only features from visual memory are tracked during the navigation. Better results could be provided with periodically detecting new features in the current image. Thus, the features from visual memory and the newly detected features could be tracked and used for the reconstruction of geometry. This approach could significantly improve the performance, because the geometry could be estimated more accurately.

Also it would be interesting to completely replace the tracker with translational model with the tracker with affine model and use it even for frame-to-frame tracking. The CPU implementations are not interesting because they are too slow for real-time application, but a GPGPU implementation [15] could offer a real-time execution. The LMedS algorithm also can be easily implemented on the GPU, using such implemen-

tation it is expected that the combination of RANSAC and LMedS could be used in real-time applications, such as the navigation phase.

As the implemented solution is designed for outdoor navigation, the efficacy of the navigation depends on time of day when the navigation is performed. The lighting conditions around noon and evening are not at all the similar. Thus, several map databases could be created, one map database could be created at morning, one at noon and one at evening. During the initialization the system chooses the map database with the maximal number of inliers in the initial image and that map database is used for further navigation.

This solution could have application in the autonomous parking of vehicles and as a taxi service. Simply, the driver drives himself somewhere near the garage, leave the car and car autonomously parks itself in the garage. As a taxi service, the solution can be used in small cars for carrying objects around a site or at a university campus.

BIBLIOGRAPHY

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[2] S. Birchfield. Chapter 8: Affine consistency check of features, May 2007. URL http://www.ces.clemson.edu/~stb/klt/user/chpt8.html.

[3] P. Bosilj. Localization of autonomous robot in visual memory. Master's thesis, University of Zagreb, Croatia, 2012.

[4] Jean-Yves Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm, 2000. URL http://robots.stanford.edu/cs223b04/algo_tracking.pdf.

[5] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. U *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15560-X, 978-3-642-15560-4.

[6] F. Chaumette and S. Hutchinson. Visual servo control, part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 13:82–90, 2006.

[7] H. Zhang D. Cobzas and M. Jagersand. Image-based localization with depth-enhanced image map. *IEEE Inter. Conf. on Robotics and Automation*, 78:1570–1575, 2003.

[8] N. X. Dao, B. J. You, S. R. Oh, and M. Hwangbo. Visual Self-localization for indoor mobile robots using natural lines. U *Intelligent Robots and Systems*, pages 1252–1255, 2003.

[9] A. Diosi, S. Segvic, A. Remazeilles, and F. Chaumette. Experimental evaluation of autonomous driving based on visual memory and image based visual servoing.

IEEE Trans. on Intelligent Transportation Systems, 12(3):870–883, September 2011.

[10] Chris Harris and Mike Stephens. *A combined corner and edge detector*, svezak 15, pages 147–151. Manchester, UK, 1988.

[11] R. I. Hartley. In defence of the 8-point algorithm. U *Proceedings of the Fifth International Conference on Computer Vision*, ICCV '95, pages 1064–, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7042-8.

[12] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, second edition, 2003.

[13] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990. ISBN 0521386322.

[14] I. Khvedchenia. Feature descriptor comparison report, August 2011. URL http://computer-vision-talks.com/2011/08/feature-descriptor-comparison-report/.

[15] Junsik Kim, Myung Hwangbo, and Takeo Kanade. Realtime affine-photometric klt feature tracker on gpu in cuda framework. U *Workshop on Embedded Computer Vision (ECV), 2009 (held in conjunction with ICCV)*, October 2009.

[16] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. U *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[17] David Nistér. An efficient solution to the five-point relative pose problem. U *CVPR (2)'03*, pages 195–202, 2003.

[18] Anthony Remazeilles and François Chaumette. Image-based robot navigation from an image memory. *Robot. Auton. Syst.*, 55(4):345–356, April 2007. ISSN 0921-8890.

[19] S. Segvic, A. Remazeilles, A. Diosi, and F. Chaumette. A mapping and localization framework for scalable appearance-based navigation. U *Computer Vision and Image Understanding*, pages 172–187, 2009.

[20] Jianbo Shi and Carlo Tomasi. Good features to track. U *Book*, pages 593–600, 1994.

[21] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[22] P. H. S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78: 2000, 2000.

Robusno praćenje značajki s primjenom u vizualnoj navigaciji

Sažetak

Diplomski rad opisuje razvijeni softver koji je sposoban robusno pratiti točke od interesa u video sekvenci. Praćene točke su korištene za rekonstukciju geometrije dvaju pogleda i za vizualno upravljanje mobilnim robotom. Svrha razvijenog rješenja je autonomna navigacija robota u vanjskom okolišu. Cijeli proces od faze učenja i kreiranja vizualne memorije do faze navigacije koja koristi podatke unaprijed naučene iz vizualne memorije objašnjen je i opisan u ovom radu. Rezultati pokazuju da implementirano rješenje ispunjava svrhu i sposobno je robusno voditi robota kroz unaprijed naučen okoliš, čak i ako su nastupile manje promijene okoliša poslije učenja.

Ključne riječi: računalni vid, autonomna navigacija robota, vizualna navigacija, geometrija dvaju pogleda, praćenje točaka, KLT, RANSAC, MLESAC, LMedS, BRIEF, provjera konzistencije, esencijalna matrica, outlieri

Robust point tracking for visual navigation

Abstract

This master thesis describes the developed software able to track points of interest in a video (image) sequence. Tracked points are used for two-view geometry reconstruction and visual servoing of the mobile robot. The purpose of the developed software is autonomous navigation of robot in outdoor environments. The whole process from the learning phase and the creation of visual memory to the navigation phase that uses the data from the visual memory is explained and described. The results show that the implemented software supports autonomous navigation and that it is robust and resistant to the small changes in environment that may occur after the mapping.

Keywords: computer vision, autonomus robot navigation, two-view geometry, visual navigation, point tracking, tracker, KLT, RANSAC, MLESAC, LMedS, BRIEF, consistency check, outliers, essential matrix

APPENDIX

To compile the attached software to this master thesis, it's needed to install OpenCV 2.3.1 library and ViSP library [1], also for compiling CMake is necessary. A C++ compiler need to have support for C++11 standard. The software is compatible with Linux and OpenCV 2.3.1, but not with OpenCV version 2.4.1.

Instruction for compiling: in desired build folder call `ccmake <path to the source code directory>`.

An example of running executable: `./arns-main-cargs.config`, where the parameter `-c` is the path to a configuration file.

In the configuration file, parameters are `-a, -f, -m, -i, -n, -e, -k, -o`. After each parameter in a new line there is space reserved for the value of parameter, e.g. the parameter `-f` has value: `/udd/atrbojev/soft/data/ifsic1`. All relative paths are relative to the path where configuration file is located.

Figure 8.1 shows an example of configuration file, Figure 8.2 shows what file should contain for mapping and what for navigation c. f. Figure 8.3. Additional information about the format of configuration file can be found in `README.txt` which is attached to the source code.

[1] The library can be downloaded from: http://www.irisa.fr/lagadic/visp/visp.html

```
-a
map
-f
/udd/atrbojev/soft/data/ifsic1
-m
./map
-e
pgm
-i
1400
-n
700
-k
camera.xml
```

Figure 8.1: An example of configuration file.

```
-a
map
-f
<path to the input video sequence>
-m
<path where the map database will be stored>
-e
<extension of images in input video sequence>
-k
<path to the camera configuration file>
```

Figure 8.2: An example of configuration file to perform mapping.

```
-a
nav
-f
<path to the input video sequence>
-m
<path to the map database>
-e
<extension of images in input video sequence>
-k
<path to the camera configuration file>
```

Figure 8.3: An example of configuration file to perform navigation.

Table 8.1: Description of parameters

-a	name of algorithm
-f	path to a video sequence
-m	path to a folder where map will be saved or loaded if navigation is performed
-i	index of first frame
-n	number of frames that will be processed
-e	extension of images in video sequence
-k	path to camera configuration file
-o	index of initial key-image (for navigation)
-ct	create the image retrieval tree after the learning