Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Master's Thesis

# Life-long Visual Localization of a Mobile Robot in Changing Environments

*Bc. Kristýna Kumpánová*

Supervisor: Ing. Erik Derner

Study Programme: Cybernetics and Robotics, Master

Field of Study: Systems and Control

May 24, 2018

# MASTER'S THESIS ASSIGNMENT

## I. Personal and study details

| | | | |
|---|---|---|---|
| Student's name: | **Kumpánová Kristýna** | Personal ID number: | **406278** |
| Faculty / Institute: | **Faculty of Electrical Engineering** | | |
| Department / Institute: | **Department of Control Engineering** | | |
| Study program: | **Cybernetics and Robotics** | | |
| Branch of study: | **Systems and Control** | | |

## II. Master's thesis details

Master's thesis title in English:

**Life-long Visual Localization of a Mobile Robot in Changing Environments**

Master's thesis title in Czech:

**Dlouhodobá vizuální lokalizace mobilního robotu v proměnném prostředí**

Guidelines:

The goal of the thesis is to design and implement a method for visual localization of a mobile robot in a changing environment and evaluate the method on a real mobile robot.
The assignment comprises the following tasks:
1. Prepare a framework for the robot localization task using the widely-used Robot Operating System (ROS).
2. Set up a suitable scene for the experiment, modelling the partially changing environment.
3. Record data sets of images with known locations. To determine the ground truth location of the robot, use an existing implementation of SLAM or a similar method, e.g., the gmapping package in ROS.
4. Survey the state-of-the-art image descriptors and choose a suitable one for the problem.
5. Design, implement and evaluate a method to detect changes in the scenes and update the stored information used to localize the robot.

Bibliography / sources:

[1] Tomáš Krajník, Pablo Cristóforis, Keerthy Kusumam, Peer Neubert, and Tom Duckett. Image Features for Visual Teach-and-Repeat Navigation in Changing Environments. In Robotics and Autonomous Systems 88 (2017), pages 127-141, Elsevier, 2017.
[2] Daniel L. Silver, Qiang Yang, and Lianghao Li. Lifelong Machine Learning Systems: Beyond Learning Algorithms. In AAAI Spring Symposium: Lifelong Machine Learning, pages 49-55, Elsevier, 2013.
[3] Cristina Romero-González, Jesus Martínez-Gómez, Ismael García-Varea, and Luis Rodríguez-Ruiz. On Robot Indoor Scene Classification Based on Descriptor Quality and Efficiency. In Expert Systems With Applications 79 (2017), pages 181-193, 2017.
[4] Relja Arandjelović, Petr Gronát, Akihiko Torii, Tomáš Pajdla and Josef Šivic. NetVLAD: CNN Architecture for Weakly Supervised Place Recognition. arXiv:1511.07247, 2015.
[5] Zhiyuan Chen and Bing Liu. Lifelong Machine Learning. Morgan & Claypool Publishers, 2016.

Name and workplace of master's thesis supervisor:

**Ing. Erik Derner, Department of Control Engineering, FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **30.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

| | | |
|---|---|---|
| Ing. Erik Derner | prof. Ing. Michael Šebek, DrSc. | prof. Ing. Pavel Ripka, CSc. |
| Supervisor's signature | Head of department's signature | Dean's signature |

## III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

| | |
|---|---|
| . | |
| Date of assignment receipt | Student’s signature |

# Acknowledgements

I would like to thank the supervisor Ing. Erik Derner for the patient guidance, the smooth cooperation and the invaluable feedback.

## Declaration

# Abstract

This thesis focuses on the development of methods suitable for the localization task in changing environments using the life-long learning approach. The methods implemented in this thesis are based on keypoint descriptor distances. The more significant is the change occurring in the scene, the lower weights are assigned to the descriptors in the changed area. The life-long learning aspect of the method consists in the weighting of the descriptors. The weights are being updated during the robot operation. This way the robot learns how to localize itself in a changing environment.

Three different methods of weighting were implemented: distance vector normalization, past distance subtraction and global reference value. Two approaches to localization were tested: matched keypoints and transformed keypoints. The methods were evaluated on experimental data that have been recorded in an arena constructed for this purpose. The project is implemented in C++ in the ROS environment.

# Abstrakt

Tato práce se zabývá vývojem metod, které jsou vhodné pro lokalizaci v dynamickém prostředí s využitím life-long learning přístupu. Metody implementované v této práci jsou založeny na vzdálenosti descriptorů významných bodů. Čím větší změna se ve scéně vyskytne, tím nižší váhy jsou deskriptorům v této oblasti přiřazeny. Life-long learning aspekt spočívá ve váhování deskriptorů. Váhy jsou aktualizovány při běhu robotu. Tímto způsobem se robot učí jak se lokalizovat v dynamickém prostředí.

V rámci této práce byly implementovány 3 metody váhování: normalizace vektoru vzdáleností, odčítání minulé vzdálenosti a globální referenční hodnota. Dále byly vyzkoušeny dva přístupy k lokalizaci: přiřazování významných bodů a transformování keypointů. Metody byly ohodnoceny na experimentálních datech, která byla získána nahrávním v aréně sestavené výhradně pro tento účel. Projekt je implementován v jazyce C++ v prostředí ROS.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Today, robots are widely used for minimization of human labor. The environment and the tasks, where the robots are mainly used these days, are structured and static. For example, manufacturing robots stay in one place and do the same job all over and over again. Their surrounding is adapted to the robot, so that, for example, there are no people and no other objects that could cause collision with the robot.

The most of the real-life environments are not static and can't be adapted to the robot. The robot needs to be able to navigate in this dynamic environment. However, even simple localization task is not that simple in the dynamic scene. The robot can remember there is a table with four chairs in the kitchen, but if somebody were to take the chairs out? How does the robot recognize it to be the same room? How can the "essence" of the particular room be defined to the robot?

There are various methods for path planning in the static environment. The robot observes the scene. It finds the path. It follows the path. If the area is known in advance and static, one could also program the robot to follow the pre-defined path, without searching for it. But what if there are people walking around the place? The robot can not blindly continue following the previously found path.

For the robot to be universal, it has to be aware of changes. If we want to have one universal robot and be able to deploy it in different buildings, or places in general, it needs to be able to adapt to the new environment and be able to function no matter what changes the environment presents to it.

The whole new area of development is human-robot cooperation. Robot has to be able to recognize which object is a human, has to pay attention to the human's movement, has to predict the human's trajectory and has to know how to behave safely and pleasantly around the human [1].

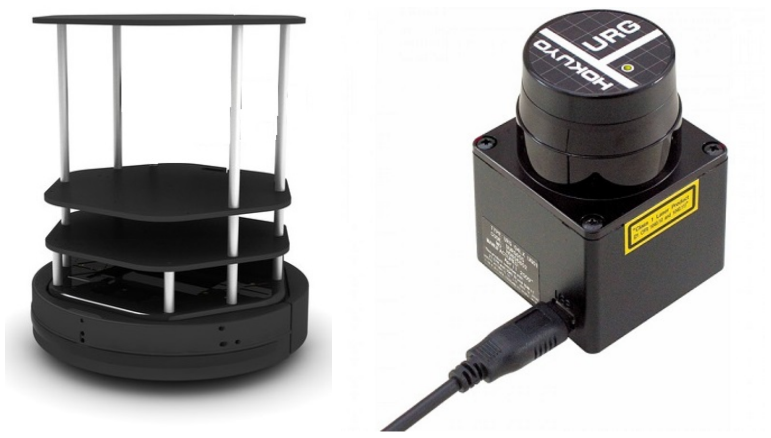


Figure 1.1: TurtleBot and the Hokuyo laser rangefinder.

Not just robots can benefit from algorithms detecting changes. More and more people depend on online maps and navigation systems for their car rides. Many have the unpleasant experience with the map being out-dated. Algorithms detecting the changes in the world and automatically updating the map would be a welcomed technology.

Other areas utilizing change detection algorithms are for example video surveillance [2], remote sensing [3] or medical diagnosis [4].

## 1.2 Problem Specification

This thesis focuses on the problem of mobile robot localization in a dynamic indoor environment. The idea is that a mobile robot will have the knowledge of its environment. A map of the building, or areas, where it can move around. The environment is said to be dynamic, that means the robot has to know its position in the map, even though its current measurements are inconsistent with the map.

One of the considered approaches might be following: First, the robot segments the image from the camera input. After it recognizes objects in the scene, it decides which objects are irrelevant for the task of localization. Typically objects that tend to move and change a lot. The robot compares its current measurement with the map while excluding the notoriously changing objects from this procedure. The robot learns which objects were moved or changed so that it can exclude them the next time. The learning which objects to exclude might be based on updating weights or model parameters;

In this thesis the chosen approach focuses on individual features in the scene, rather than whole objects, therefore no segmentation is needed. The methods are described in Chapter 4 in more detail.

The approach will be tested in a custom arena, that will consist of narrow corridors simulating the usual indoor environments.

For the implementation TurtleBot robot (see Figure 1.1) will be used. The robot has one RGB camera mounted in the front and the Hokuyo laser rangefinder. The laser will be used to obtain the ground truth map of the arena. Only the camera will be the source of information for the localization task.

## 1.3 Structure of this Thesis

In Chapter 3 the preparation of the framework is documented and software choices are explained. In Chapter 2 the theory is explained and the state of the art is discussed. In Chapter 4 a principle of implemented methods are explained. In Chapter 5 the experiment set up and recorded data are described. In Chapter 6 the results of the evaluation are presented and discussed. Chapter 7 sums up the results of this thesis and presents suggestions for improvement.

# Chapter 2

# State of the Art

## 2.1 Mobile Robot Design

One of the obvious applications of mobile robots is logistics in warehouses. Transporting material, documents or products is labor intensive, mundane work. There are companies focusing on producing robots for this exact application (inVia Robotics, Fetch Robotics, Locus and many others). There are also public competitions held to motivate academic robotics community come up with solution for this task (such as the Amazon Picking Challenge [5] within the RoboCup@Work competition).

Robots used for warehouse logistics usually use RGB-D camera and laser. The laser can scan the area in a high range, but only in one plane, therefore it might miss objects that can cause a collision with the robot. On the contrary, the camera can see the whole scene in front of the robot, but is inefficient in scanning distant objects.

In the Amazon Picking Challenge, teams were supposed to come up with a robot, which is able to pick objects of various shapes from the shelf. Commonly used manipulator solution are manipulators based on suction, two-finger gripper, 3-finger hand, spatula and custom gripper [5].

Standard solution of mobile robot design for warehouse logistics also contain RFID reader, which even organizers of RoboCup@Work plan to involve in competitions [6].

## 2.2 Mobile Robot Environment

When designing a mobile robot, it is crucial to determine the kind of environment where the robot is supposed to work. Not only mechanical solution needs to be adjusted to the environment, but software as well. Sensors and the algorithms used for indoor and outdoor environments will differ. A solution for static and dynamic environments will differ. A solution for environments with people will differ from the one without the constraints of

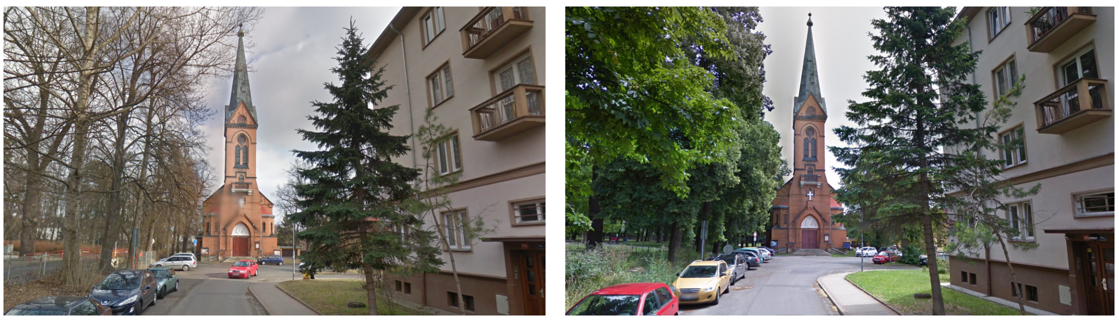


Figure 2.1: Seasonal change in the outdoor environment acquainted from Google Street View Time Machine.

being safe around people. If we desire to make a robot for one specific place, for example, a warehouse, we can also adjust the environment to the robot. Such as adding QR codes on the floor or the ceiling.

The outdoor environment changes differently than the indoor environment. For example in the outdoor environment, the season and the weather will have a significant impact on the appearance. The example of such change is depicted in Figure 2.1. In the outdoor environment, we can expect cars, people and animals to be constantly moving objects. Buildings are most stable objects but can change color, or there might be a poster glued to the wall. We can also expect changes in lighting throughout the day.

In the indoor environment, there are usually some static objects like tables and shelves, but chairs will get moved, doors will be opened and closed, boards on the wall will change the content, and possibly people will occupy the space and move around.

## 2.3 Change Detection

The change detection survey [7] gives a general idea of this research area. In order to compare two images for change detection, it is necessary to perform certain pre-processing steps. For example eliminate differences between images in illumination, sensor calibration or atmospheric absorption. In video surveillance application the camera might be fixed, but for other applications, it is needed to include registration in the pre-processing, and depending on the application, precise coordinate frame alignment might be needed.

There are many methods for dealing with illumination aspect of an image. The intensity of a pixel depends on the light source (illumination), surface reflectance and surface orientation [8]. With certain assumptions this problem can be tackled by intensity normalization, filtering the illumination component out of the image, illumination modeling, intensity transformations or averaging.

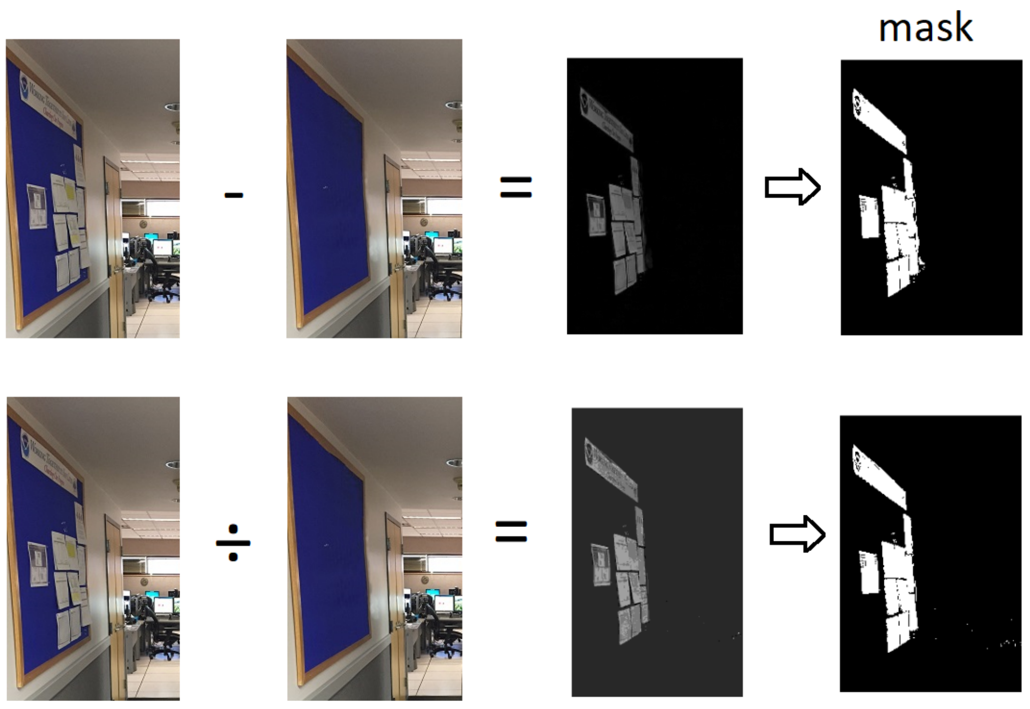


Figure 2.2: Difference and ratio images and the change masks obtained from them.

Likewise, there are many approaches to the change detection algorithm. The simplest approaches are image differencing and image ratioing, see Figure 2.2. In the image differencing method a signed difference image $D(x)$ is obtained using (2.1), where $I_i(x)$ is the intensity of the pixel $x$ in the image $i$. The image ratioing is faster method in which the ratio image $R(x)$ is obtained using (2.2). In both cases certain, experimentally found, threshold $\theta$ is applied to the difference/ratio image, to decide which pixels represent a change and thus create change mask $B(x)$ using (2.3).

$$D(x) = I_2(x) - I_1(x) \tag{2.1}$$

$$R(x) = \frac{I_2(x)}{I_1(x)} \tag{2.2}$$

$$B(x) = \begin{cases} 1 & \text{for } D(x) > \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Another approach is the hypothesis testing, where the image pair $(I_1(x)$,$I_2(x))$ are considered the random vector and their conditional joint probability density functions $p(I_1(x), I_2(x)|\mathcal{H}_0)$ and $p(I_1(x), I_2(x)|\mathcal{H}_1)$ provide the information about the most suitable hypothesis. The null hypothesis $\mathcal{H}_0$ represents the scenario where there was no change. The alternative hypothesis $\mathcal{H}_1$ represents the scenario where there has been a change. The decision made in an individual pixel is also based on the decision made in the neighboring pixels.

Combining the statistical and the simplest approach the significance test $S(x)$ can be used on difference image $D(x)$ to see how consistent it is with the null hypothesis, see (2.4), where $\tau$ is some significance threshold. Similarly can be used likelihood ratio test $L(x)$, as seen in (2.5). The threshold can be computed to produce desired false alarm rate and include costs associated with each decision.

$$S(x) = p(D(x)|\mathcal{H}_0) > \tau \tag{2.4}$$

$$L(x) = \frac{p(D(x)|\mathcal{H}_1)}{p(D(x)|\mathcal{H}_0)} \tag{2.5}$$

The next group of approaches to mention are predictive models. One of the ideas is to fit the intensity values of each block consisting of several pixels to a polynomial function of the pixel coordinates. The null hypothesis in this case is – the corresponding blocks are best fit by the same polynomial function. The alternative hypothesis is – the corresponding blocks are best fit by different functions.

Another predictive model is based on a Gaussian distribution of pixel intensity in time, and the decision is made by likelihood ratio testing. A similar approach uses Wiener filter to predict the intensity value of the pixel based on its previous values. In this case, if the prediction error is several times worse than expected, the pixel is classified as changed. And lastly, a neural network can be used for unsupervised learning of parameters of a non-linear predictor. Pixels, for which the predictor performs poorly, are classified as changed.

Mostly in the context of the video surveillance, another group of methods called Background Modeling is used. It is specific for the video surveillance applications that there is a large amount of video data available, containing images taken only few seconds apart. The whole sequence is typically used to make the decision. The camera might be fixed. Those are the constraints in which video surveillance applications and, for example, remote sensing applications differ. In the video surveillance application, it might be desired to implement some semantic interpretation of the data. For example, a person enters a room or a truck drives down a road.

The background is usually represented by a Gaussian distribution of the pixel intensity or by a mixture of Gaussian distributions if the background is dynamic (for example, there are swaying branches in the background). The foreground pixels lie some number of standard deviations from the mean of the background and are clustered into objects.

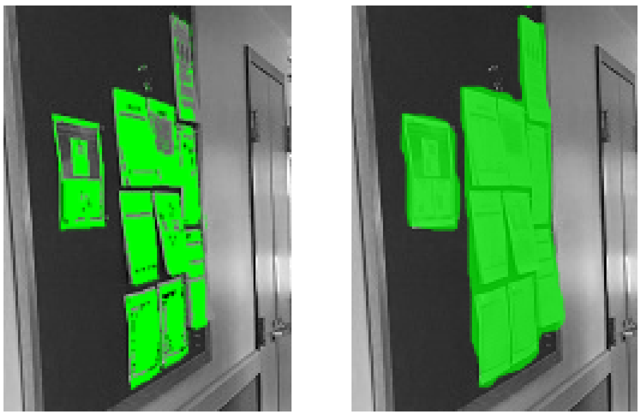


Figure 2.3: Example of the inconsistent mask (on left) and consistent mask (on right).

Whenever the decision is made for an individual pixel, the result is noisy. There might be few pixels representing a changed object, which has been classified as unchanged. To address this issue, the decision can be made for the whole area rather than individual pixels. This problem is referred to as the Change Mask Consistency, see Figure 2.3.

The simple approach is to use a median filter to remove the small groups of pixels with a different label than their neighbors. The unsupervised clustering of regions with homogeneous intensity can be used to detect objects. The decision is then made for the whole object. Which is basically a segmentation.

In a task of segmentation, we want to assign labels to pixels according to what object they are representing. Widely used approach to segmentation of an image in spatial context are Gibbs-Markov random fields [9]. Pixels are considered nodes of an undirected graph connected only with pixels in its neighborhood, defined by the used neighborhood system, see Figure 2.4. The label assigned to a node (pixel) can only be influenced by its neighbors.

The segmentation prior to the change detection was used, for example, in the paper [10] evaluating different approaches to change detection over a satellite image of the New Burg El-Arab city from 1990 and 2000. In this approach, the author first segmented the images into classes "soil", "vegetation" and "urban" and then compared the bitmaps of the two images. Pixels that were labeled differently than the corresponding pixels in the image taken at the different time were considered changed. Moreover, the author gained information about the type of the change at every pixel. This approach is referred to as Post-Classification.

Another method used for change detection is the Principal Component Analysis [9]. It is an eigenvector-based analysis utilizing the information about the variance between images. Resulting principal components are uncorrelated (covariance matrix is diagonal) and most of the variance of the data is concentrated in the first principal component [11]. Thus the first principal component contains most information about differences between images.

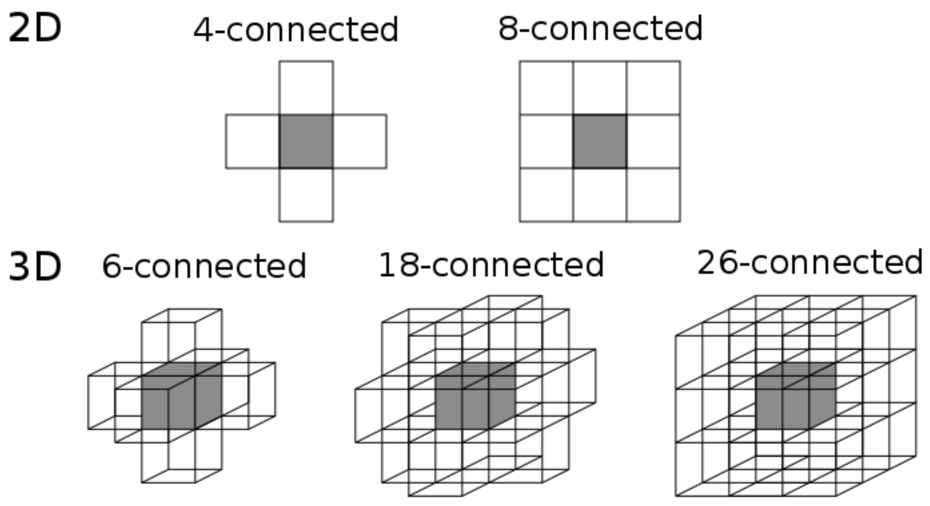


Figure 2.4: Common neighborhood systems for square grid in 2D and 3D.

## 2.4 Feature Detection

Widely used approach to an image processing is detecting individual points (features) in the image, which are easily recognizable in the scene even if the image was taken from another viewpoint, distance and under different conditions (illumination, for example). Localization is usually carried out by matching features found in both images. Images that have a large number of the same features are displaying the same scene. Another application is panorama stitching, where the software finds corresponding features and stitches the images at those points.

To match the features, each feature needs its descriptor. Descriptor should describe the appearance of the feature in such a way, that the same feature has the same descriptor even if the feature has a different scale, rotation, affine transformation, if the image has different lighting conditions or additional noise [12]. There is a number of methods that aspire to do this. For example SIFT, SURF, MSER or FAST, to name some of the most widely used.

Scale Invariant Feature Transform [13] (SIFT) searches at different scales for feature location candidates using the difference-of-Gaussian function. The local image gradient is computed, and its orientation is assigned to the feature. The descriptor is created based on the scale and the orientation of the feature. The descriptor consists of values of orientation histogram of the feature. The histograms with 8 orientation bins are computed in $4 \times 4$ subregions which creates descriptor of dimension 128.

Speeded Up Robust Features [14] (SURF) detector is based on Hessian matrix. The orientation is obtained from Haar-wavelet responses. The descriptor consists of Haar-wavelet responses. The descriptor has a dimension of 64.

Maximally Stable Extremal Regions [15] (MSER) uses connected components as a function of intensity to find maximally stable extremal regions. Thresholding an image at different levels produces regions (connected components) with intensity lower than the threshold. If an area in one image (for example a letter on a poster) creates a connected component under a certain threshold, it is very probable, that it will create a connected component under some threshold also in other images no matter what scale, rotation or illumination differences there are between the images.

Features from Accelerated Segment Test [16] (FAST) is simple and fast corner detector. FAST considers a circle of 16 pixels around a corner candidate $p$ and decides the point $p$ is a corner if there is a continuous set of 12 pixels in that circle that are all brighter or darker than $p$ by a certain offset. This offset is called `threshold`.

## 2.5 Related Work

Mobile robotics has been gaining popularity in the past decades and lot of work has been done in this field. Many papers have been published, that brought new approaches or tested the best utilization of the existing ones. There is a continuous discussion going on about drawbacks of the existing methods and how to overcome them. Lastly, many research teams focus on specific practical applications of the mobile robotics in the current world.

In the paper [17] the research team propose a method of dynamic maps in which the map consists of multiple maps that are being updated with different frequency. They called it the *short-term memory map* and the *long-term memory map*. The algorithm is randomly picking samples that are to be replaced with the new measurement data. It is a process of forgetting. The samples from *short-term memory map* gets replaced in a shorter period, therefore gets forget sooner. This way the map is able to reliably represent a dynamic environment.

In the paper [18] the research team proposed a method to detect changes in architecture and the appearance of streets and update the map accordingly. First, the 3D model of an area, for example, city, is reconstructed. Then with new imagery, the algorithm decides which changes are substantial and based on this decision, the 3D model gets updated. For the reconstruction it is important to ignore changes like people appearing in the street, a tree losing its leaves or a new poster being glued to a wall.

This research is motivated by services like GoogleEarth or StreetView. The idea is that the new imagery can be obtained from civilian devices, such as onboard cameras of service vehicles. Therefore the 3D model can always be up-to-date.

The method is based on assumption, that only sparse and low-resolution imagery might be available for the model update. The research team argues that the intuitive approach of comparing 3D model reconstructed from the new imagery to the original 3D model relies on the quality of the reconstruction and therefore is not ideal for this scenario.

When the correspondences of an image with the 3D model are found, the algorithm computes the transformation between the coordinate system of the image and coordinate system of the 3D model. There might be a problem when changes cover a significant part of an image, and therefore sufficient amount of correspondences cannot be found. The method addresses the problem by computing transformations between coordinate systems of images with respect to each other. Then when one image has enough correspondences with the 3D model, transformations for the rest of the images can be computed as well.

The 3D model is divided into 3D regions. Each 3D region consists of voxels. Those voxels are put into a graph as vertices (similar to Gibbs-Markov random fields approach see section 2.3). Two vertices are connected (are neighbors), when the two corresponding voxels touch in a corner (26-connected, see Figure 2.4). To decide whether a voxel has changed, a pair of images with a sufficient overlap in the scene are chosen. One is the source image, and the other is the target image. Pixels from the source image are projected onto 3D region. Then the colors are projected from the 3D region to the target image. If there is no change in geometry of the scene, the pixels from source image are correctly projected into the target image. If there is a change, however, the inconsistency between the projected image and the target image will occur. The decision, whether a change occurred, is dependent also on decisions made in neighboring voxels.

The algorithm uses a probabilistic framework with the semantic knowledge of the environment. The semantic knowledge is used in following way: There are $r$ mutually exclusive classes of objects $\{0, 1, 2, \cdots, r-1\}$, where only the class 0 are objects relevant to the task. For every source image, a probability of each pixel belonging to the class 0 is computed and projected into the target image the same way as described previously. If either the pixel of the target image or corresponding projected pixel has low probability of belonging to the class 0, the corresponding voxel is considered to be irrelevant to geometric changes in the environment. The acquisition of the semantic knowledge is not discussed in this paper.

In [19], the research team argued that two places are best to compare when viewed from similar viewpoints. Their approach was to transform an image into the viewpoint of the other image. To find which images capture the same scene, dense VLAD descriptors are compared. Sparse descriptors showed significantly worse performance during the experiments.

In the task of people detection, [20] explored the possibility of combining RGB image, depth image and optical flow to detect people in various conditions – such as in the dark or on long distance – by adaptively assigning weights to information from those images based on the said conditions. They used a mixture of convolutional neural network experts. Each expert is trained for specific type of images (i.e. RGB image, depth image or optical flow). The Gating network is trained to assign weights to outputs of the experts. In their experiments, the multi-modal approach consistently outperforms the single-modal approach.

In an effort to increase the performance of convolutional neural networks (CNNs), [21] developed a training procedure based on weakly supervised ranking loss to learn parameters of their CNN architecture. As a dataset, they used images from Google Street View Time

Machine. The work was focused on recognizing places despite changes in illumination during the day or seasonal changes. The new CNN structure was named NetVLAD and was shown to outperform the off-the-shelf CNN models.

In the study [22] the research team focused on long-term visual navigation. The team argues that from the long-term navigation perspective, the viewpoint and scale robustness is not as important as the robustness to variable lighting and seasonal changes. They evaluated some of the existing methods for their suitability in a changing environment. They also proposed the GRIEF descriptor, that scored high in their evaluation experiment.

One of the difficulties in place recognition pose repeated structures. When repeated structures are present in the scene, like windows on a building, tiles or decorative patterns, commonly used place recognition methods fail. [23] addressed this problem and proposed useful representation of repeated structures and geometric verification method taking advantage of this representation.

The study [24] focused on finding factors influencing the performance of classifiers in indoor place recognition. Since CNN is computationally demanding, the team uses SIFT descriptor and OISVM classifier and use spatial pyramid for obtaining the set of initial descriptors. All that with the emphasis on real-world scenarios. They found that for indoor scene classification, the horizontal pyramid division is more appropriate than the vertical one. For the real-time classification, the size of the descriptors is crucial. The team estimated the optimal size of the descriptor at 400. They found that large-size descriptors perform with similar accuracy to middle-size descriptors.

# Chapter 3

# Framework Preparation

## 3.1 ROS

The Robotic Operating System [25] is an open source framework that offers tools and libraries for robotic systems development. The processes running within ROS are called *nodes*. Nodes are defined inside so-called *packages*, which is a collection of source codes, make files and other configuration files needed to compile and run a project.

ROS, while running, maintains a list of *topics* that individual nodes can subscribe to and publish to. Subscribing to a topic means, that the node will get every *message* that is being published to the topic. Publishing means that the node can send a message to the topic. This way the processes (nodes) in ROS communicate. Each topic has its message type that the publishing nodes have to respect. Another type of file in a package is *launch*. Launch contains nodes that are to be started up after running the launch file. In the launch file, node parameters can be set.

The first part of this work was to learn how to use ROS with the turtlebot robot. Turtlebot software publishes odometry information in form of `nav_msgs/Odometry` message into the `/odom` topic. The Hokuyo laser rangefinder mounted on the robot publishes `sensor_msgs/LaserScan` messages into the `/scan` topic. The position from odometry can be read right away from the `nav_msgs/Odometry` message, however, the `sensor_msgs/LaserScan` message only publishes its measurements and does not make sense of them. The gmapping node [26] was used for generating the map and keeping track of the robot's position in that map, as a new laser data are being published.

For obtaining images from camera mounted on the turtlebot, the `pylon_camera` package [27] was used.

## 3.2 Improvised Arena

To further develop the framework and to get better accustomed to the turtlebot robot, the improvised arena has been built. Few chairs and boxes were laid down to create a circular

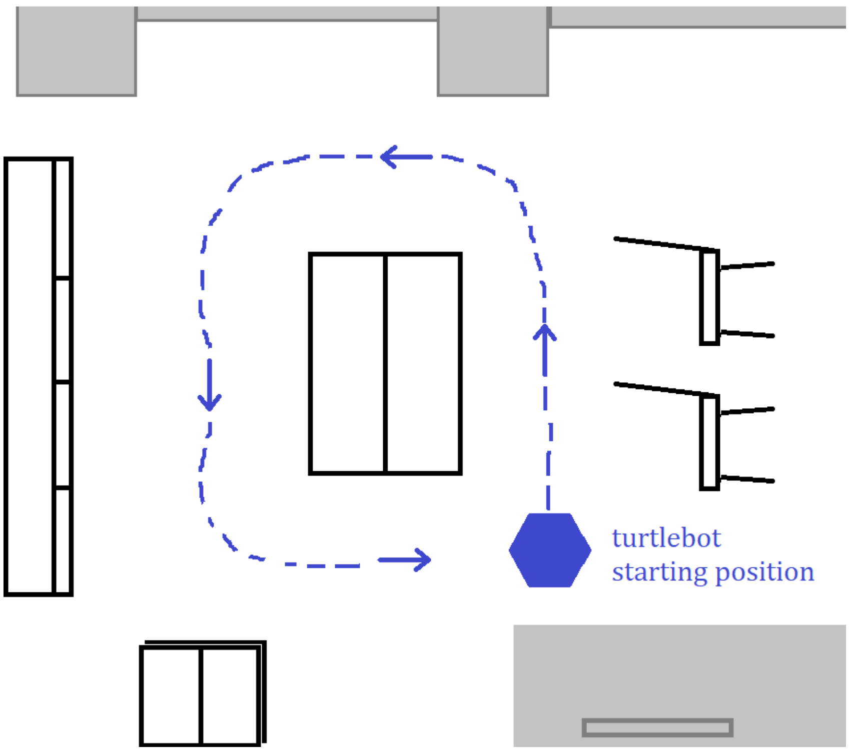


Figure 3.1: Improvized arena scheme.

corridor. The layout can be seen in the scheme in Figure 3.1. The measurements has been recorded using tools from the `rosbag` package [28].

### 3.2.1 Recording

Three runs have been recorded. During all of them, the robot followed the corridor in a counter-clockwise direction closing the loop. The computer mounted on turtlebot robot was connected to WiFi and running a screen sharing service TeamViewer [29]. Via the TeamViewer application, the robot was controlled wirelessly from a laptop.

The turtlebot computer was running a `minimal.launch` launch from the `turtlebot_bringup` package [30] that turns on and controls the turtlebot base and enables it to move and track odometry. For manual control of the robot's movement, the `keyboard_teleop.launch` launch file from the `turtlebot_teleop` package [31] was used. A communication with the Hokuyo laser rangefinder ensured the `hokuyo_node` node from the `hokuyo_node` package [32]. For the map generation, as stated above, `slam_gmapping` node from `gmapping` package [26] was used. The `tf_remap` node from the `tf` package [33] was used to get transformation between the `base_link` frame and the `base_laser` frame. Lastly, the `camera.launch` launch file from the `camera_node` package [34] was used to access the camera information.

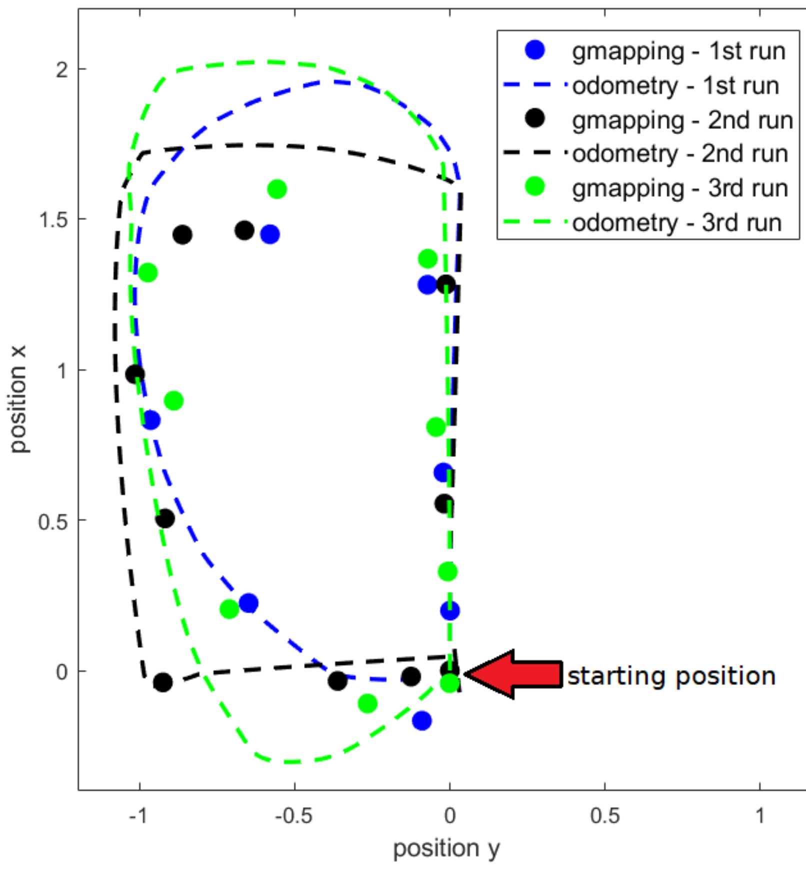


Figure 3.2: Positions extracted from the rosbag files recorded in the improvised arena.

### 3.2.2 Processing

To extract odometry and gmapping positions from the rosbag file, a node `data_recorder` has been created. The `data_recorder` node subscribes to the `/odom`, `/map` and `frontCamera/image` topics. The extracted position data are shown in Figure 3.2. As discovered during data processing, the gmapping positions were acquired with the period of 5 seconds (see the `rtq_graph` in Figure 3.4) which is not suitable for further work.

#### 3.2.2.1 Speed of Gmapping

There is a gmapping node parameter `map_update_interval` (see figure 3.3), which is set to 5 seconds by default. This parameter needs to be adjusted before recording. It has not been done before recording in the improvised arena. Therefore some kind of position interpolation was needed to fill data between the map update intervals.

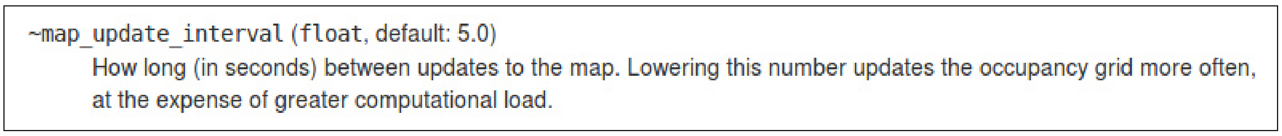

~map_update_interval (float, default: 5.0)
How long (in seconds) between updates to the map. Lowering this number updates the occupancy grid more often, at the expense of greater computational load.

Figure 3.3: Screenshot cutout from gmapping wikipage http://wiki.ros.org/gmapping. Parameter of the node.

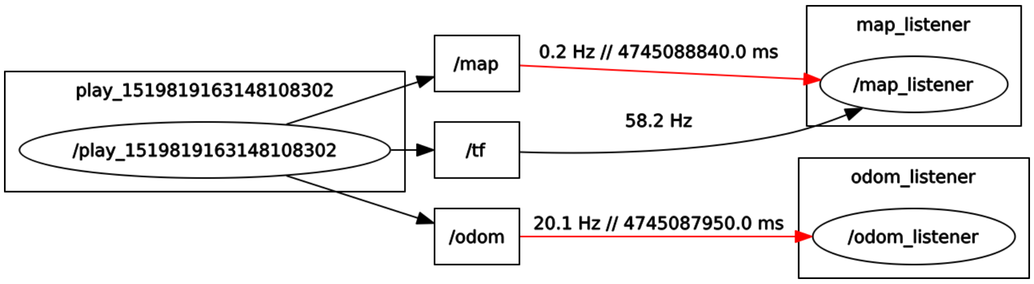


Figure 3.4: rtq_graph of the nodes and topics during rosbag replay.

#### 3.2.2.2 Interpolation

For the $x$, $y$ position interpolation `spline` library [35] has been used. The results for the positions are good. The angle interpolation, however, cannot be done without any further processing. The interpolation of position and angle are shown in Figure 3.5. The representation of an angle in ROS is a quaternion, which can be easily transformed into wrapped Euler angle, but neither of those representations is suitable for the interpolation, as shown below.

#### 3.2.2.3 Angle Interpolation: Interpolated Quaternions and Wrapped Euler Angles

The quaternion interpolation has been tested as well as the Euler angle interpolation. The results are the same and are shown in Figure 3.6 for interpolated quaternions specifically.

#### 3.2.2.4 Angle Interpolation: Computed Angle from Interpolated Positions

One of the solutions to this problem might be to compute the angle between a pair of following positions and assign it to the earlier position. That would mean completely ignoring the angle data from gmapping. The angle will not be precise, because the robot moves along curves, not along lines, from one point to the other. Moreover, if the robot rotates around at one position, the information will be lost. The interpolated angles with this method are shown in Figure 3.6.

#### 3.2.2.5 Angle Interpolation: Continuous Angle Representation

The idea is to convert the angle values into a continuous representation, unwrapping the angles, before interpolating. After interpolation, the values are converted back into the

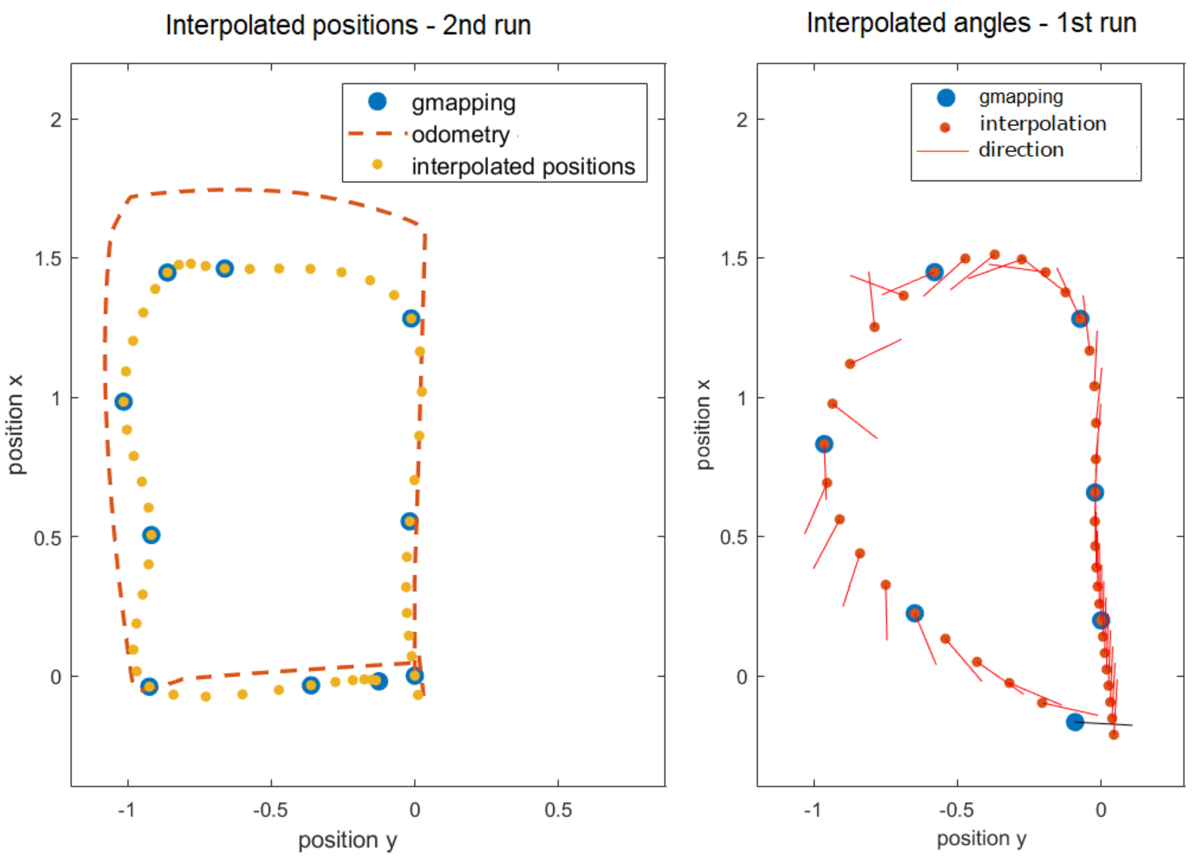


Figure 3.5: Interpolated positions (on left) and faultily interpolated angles (on right).

standard angle representation (Euler angles). Resulting interpolated directions can be seen in Figure 3.6. This approach produces the best results, and this method was chosen for the angle interpolation.

#### 3.2.2.6 Angle Interpolation: Decomposition to Sine and Cosine Component

Another considered approach was decomposing the angle to its sine and cosine component, interpolating the components and then compose the angle back. Results are seen in Figure 3.6.

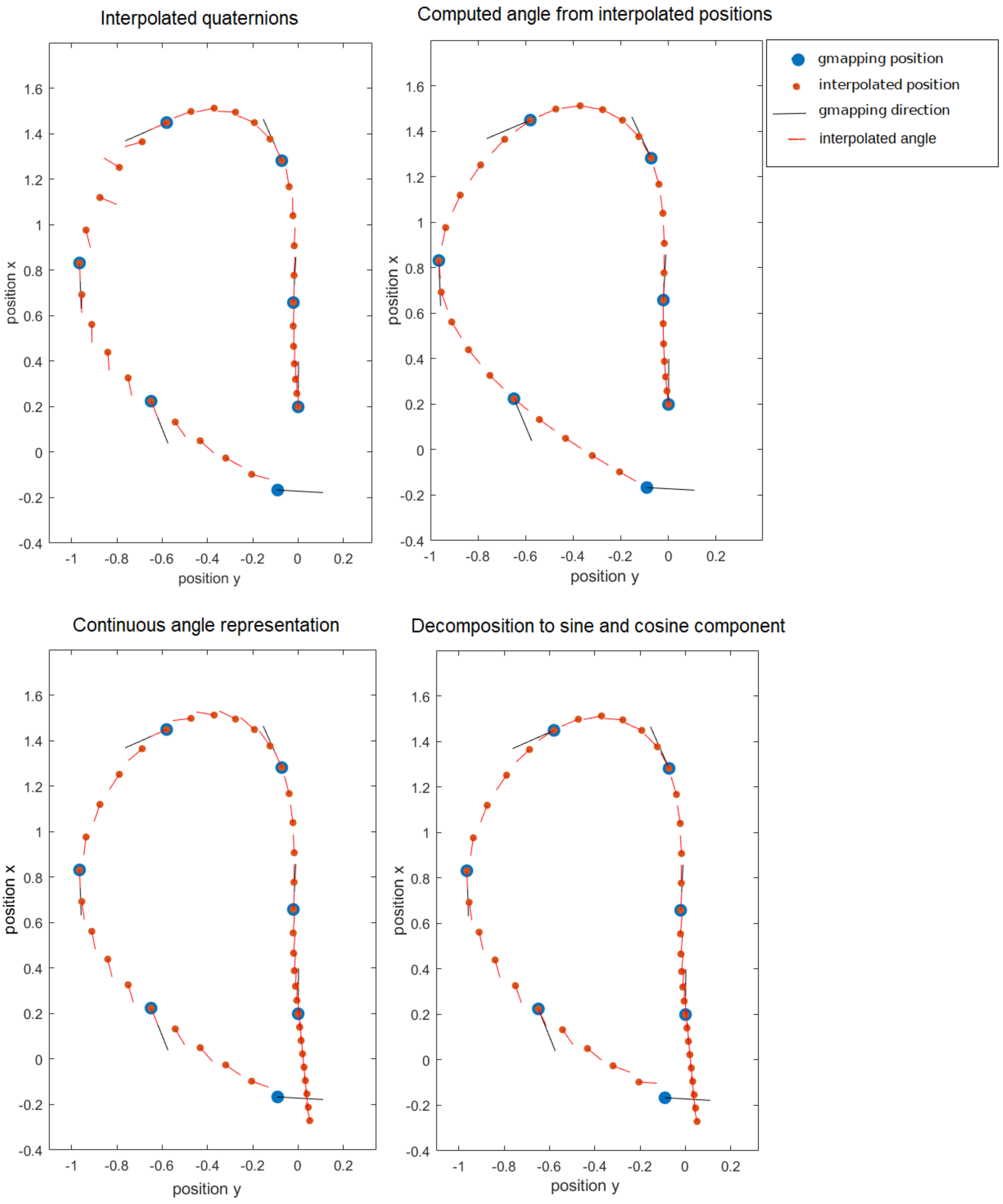


Figure 3.6: Interpolated angles for quaternion interpolation (top, left), computation of angle from interpolated positions (top, right), continuous angle representation (bottom, left) and sinus decomposition (bottom, right) methods.

## 3.3 Naive Localization

With the use of the data gathered from the improvised arena recordings and preprocessed as described above, the naive localization method has been implemented. The method works following way:

1. Load an image
2. Make a gray-scale version of the image
3. Scale the image down to 100x100 pixels
4. Put columns of the image pixels into a vector
5. Compute the distance between images

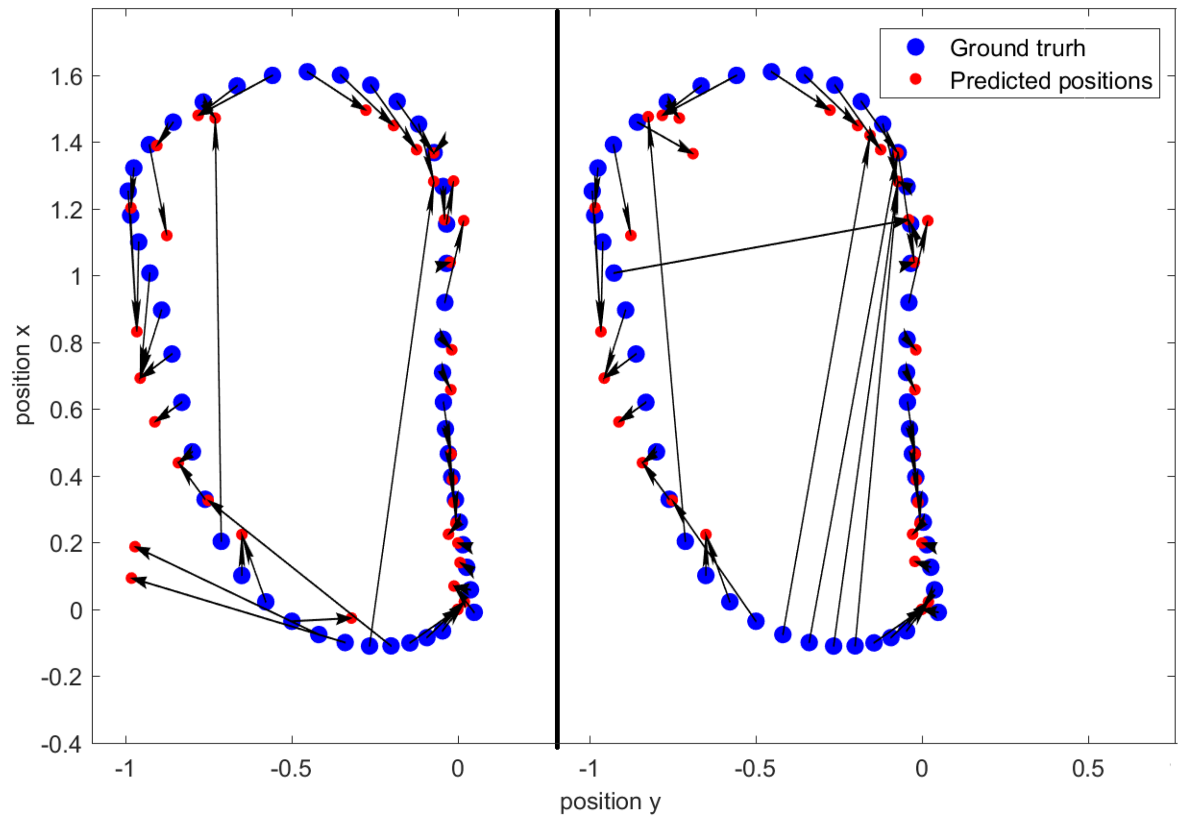
Two cost function have been used to compute the distance $d$ between images: $d = |I_1 - I_2|$ and $d = (I_1 - I_2)^2$, where $I_i$ represents the i-th image as a matrix of pixels. The data has been divided into two training and test datasets. For the first dataset, the positions from the first and the second run are put together as the training dataset. The positions from the third run are used as the test dataset. For the second dataset, every third position from all runs are put into the test dataset and all of the remaining positions (every first and second position) are put into the training dataset. Resulting estimations for both datasets can be seen in Figure 3.7.

The error of the estimation was computed as (3.1) describes, where $x_i$ is the true i-th position, and $\hat{x}_i$ is the estimate of the i-th position, and $N$ is the number of positions. The errors for both datasets and both cost functions are shown in Table 3.1.

$$e = \frac{\sum_i^N |x_i - \hat{x}_i|}{N} \tag{3.1}$$

| | | error_x | error_y | angle error | distance |
|---|---|---|---|---|---|
| 1st dataset | abs$(I_1 - I_2)$ | 0.0932 | 0.1806 | 0.9259 | 0.2032 |
| | $(I_1 - I_2)^2$ | 0.0955 | 0.2596 | 1.0761 | 0.2766 |
| 2nd dataset | abs$(I_1 - I_2)$ | 0.0993 | 0.1652 | 0.6601 | 0.1927 |
| | $(I_1 - I_2)^2$ | 0.1131 | 0.2253 | 0.9695 | 0.2521 |

Table 3.1: Errors computed as differences between the true position and the predicted position for every coordinate individually and average distance between the true and the estimated position for naive localization method.



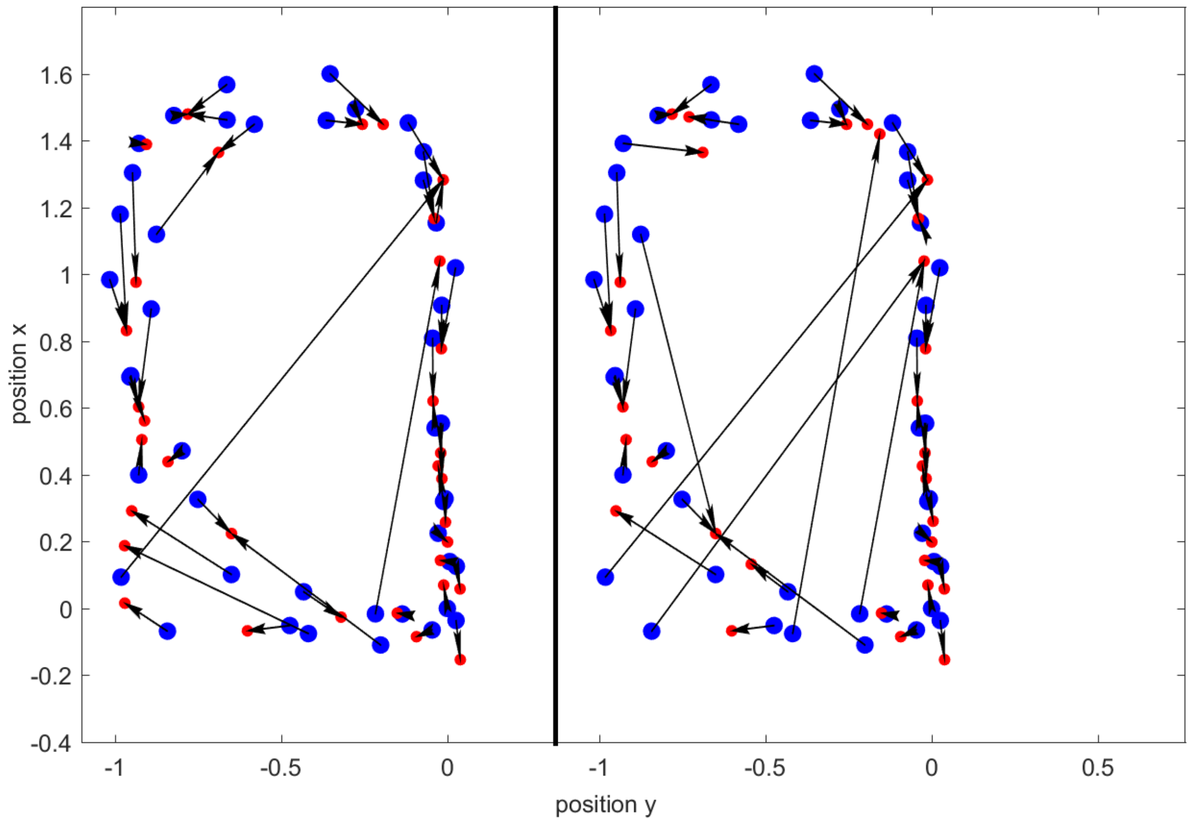


Figure 3.7: Resulting position estimates for the first dataset (in the upper row) and the second dataset(in the bottom row).

| | error_x | error_y | angle error | distance |
|---|---|---|---|---|
| first dataset | 0.1369 | 0.3661 | 1.1912 | 0.3909 |
| second dataset | 0.0825 | 0.1614 | 0.6346 | 0.1813 |

Table 3.2: Errors computed as differences between the true position and the predicted position for every coordinate individually and average distance between the true and the estimated position for BoW localization method trained with SIFT descriptors.

## 3.4 SIFT Feature Matching

The OpenCV 2 library [36] has been used to detect SIFT features and extract SIFT descriptors. The used SIFT detector setting is shown in (3.2). For matching the keypoints the FLANN matcher was used. The results of matching can be seen in Figures 3.8 and 3.9.

$$\begin{aligned} \texttt{nOctaveLayers} &= 4 \\ \texttt{contrastThreshold} &= 0.03 \\ \texttt{edgeThreshold} &= 5 \\ \texttt{sigma} &= 1.6 \end{aligned} \tag{3.2}$$

A Bag of Words trainer was created to simplify image registration. The SIFT descriptors of the training dataset were quantized into Bags of Words. Each bag is a vector of zeros and ones. A bag of an image contains a number one on the position, that is assigned to a SIFT descriptor of a feature that is present in the image. For both the training and the test dataset are then created Bags of Words. Those can be used to find the closest image among database images to the query image. The procedure is depicted in Figure 3.10.

The same two datasets as in Section 3.3 were used. Results for both datasets are shown in Figure 3.11.The distance between two images is computed as the Euclidean distance $d_{L_2}$ between the BoW descriptors $\delta_{BoW,x}$ as show in (3.3). The average distances between the true and the predicted positions are shown in Table 3.2.

$$d_{L_2} = \sqrt{\sum (\delta_{BoW,1} - \delta_{BoW,2})^2} \tag{3.3}$$

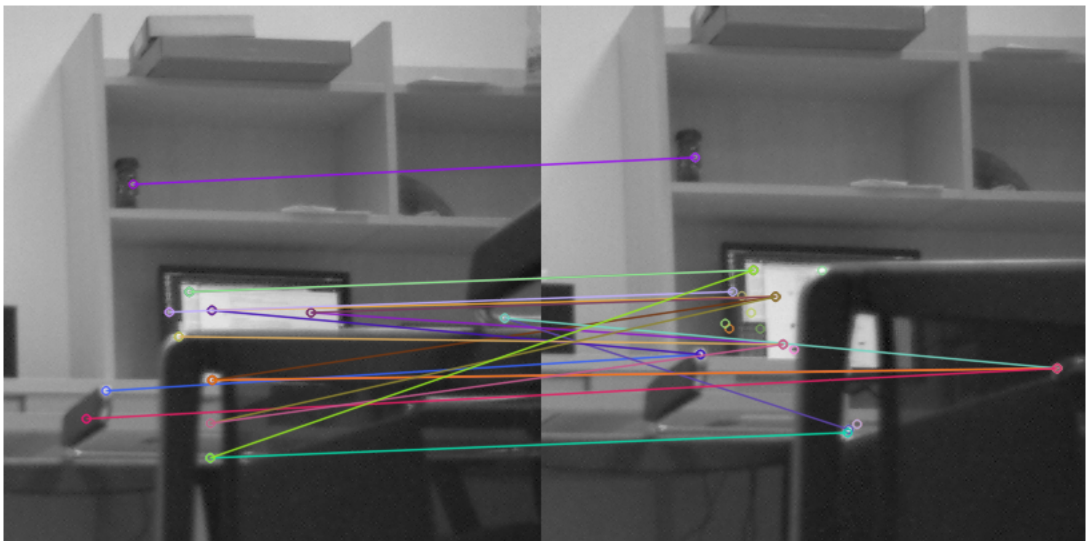


Figure 3.8: Example of SIFT feature matching on a shelf.

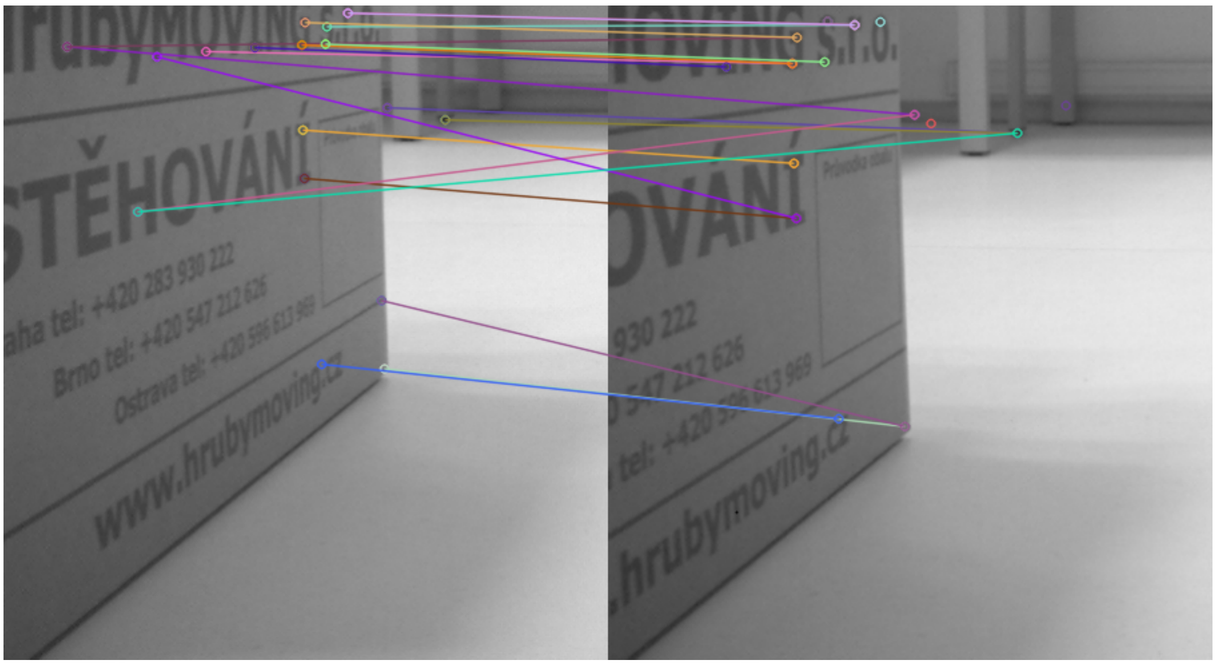


Figure 3.9: Example of SIFT feature matching on a box.

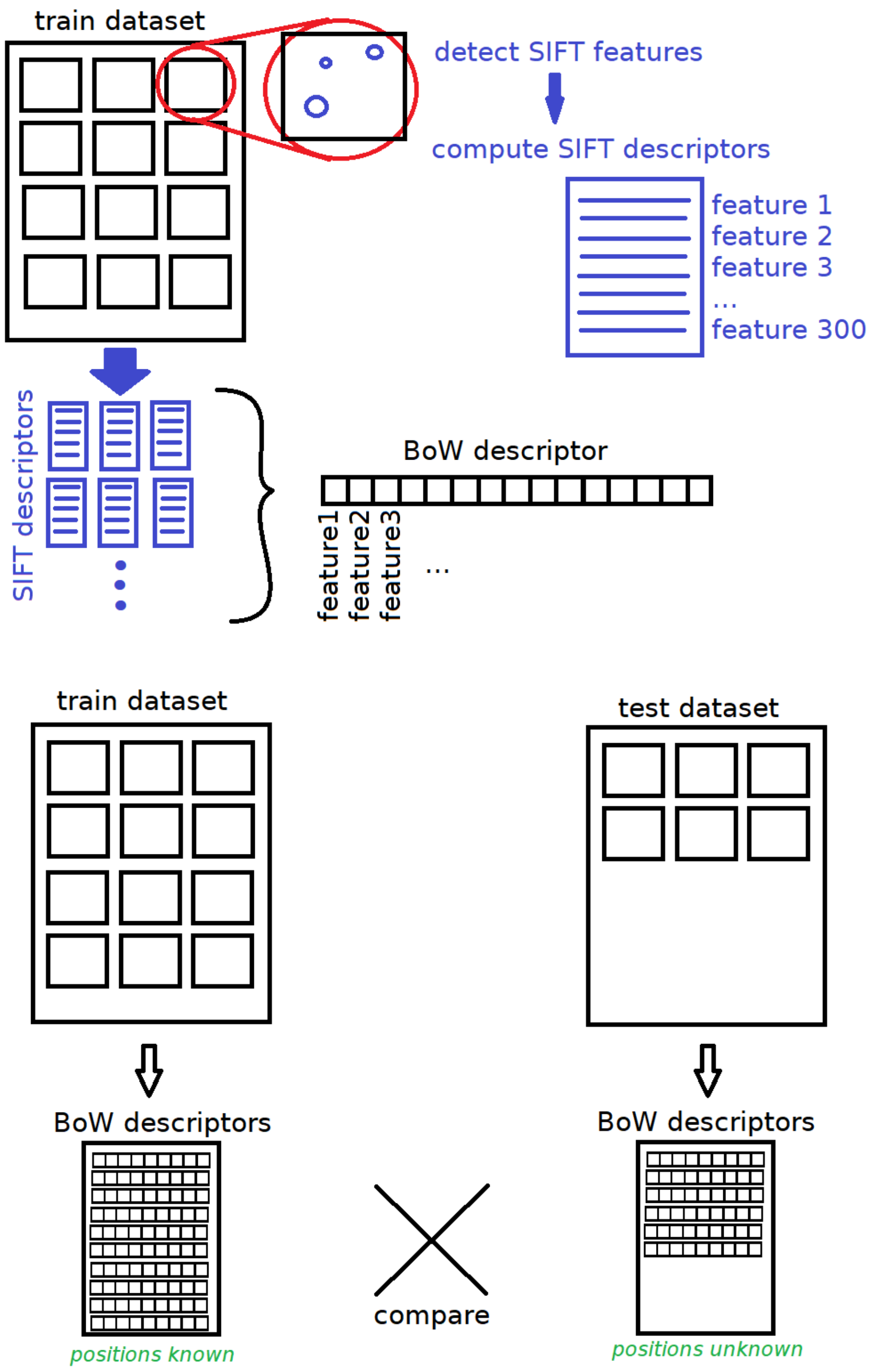


Figure 3.10: Process of creating BoW descriptor.

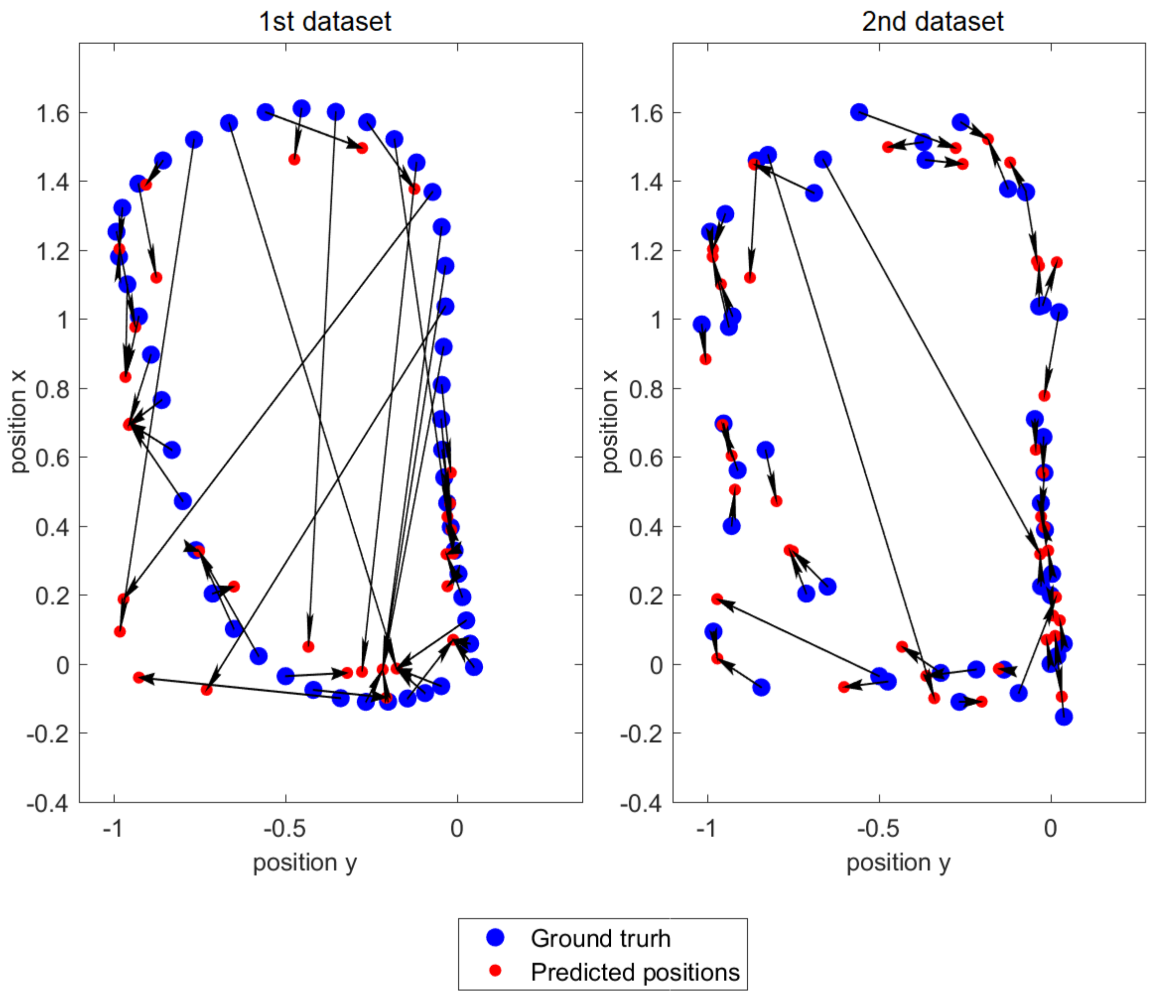


Figure 3.11: Results for the first dataset (on left) and the second dataset (on right) using the BoW descriptor trained with SIFT descriptors.

## 3.5 Feature Detector and Descriptor Extractor Evaluation

OpenCV offers various feature detectors [37] and descriptor extractors [38]. To find the best combination, tests have been conducted. The tested feature detectors were following:

- `SiftFeatureDetector`
- `SurfFeatureDetector`
- `StarFeatureDetector`
- `MserFeatureDetector`
- `FastFeatureDetector`

The tested descriptor extractors were following:

- `SiftDescriptorExtractor`
- `SurfDescriptorExtractor`
- `BriefDescriptorExtractor`

For the test dataset of 6 pairs of images were used. The dataset is shown in Figure 3.12. The used setting of feature detectors:

**SIFT**:

$$\begin{aligned}\texttt{nFeatures} &= 0;\\ \texttt{nOctaveLayers} &= 4;\\ \texttt{contrastThreshold} &= 0.03;\\ \texttt{edgeThreshold} &= 3;\\ \texttt{sigma} &= 1.6;\end{aligned} \tag{3.4}$$

**SURF**:

$$\begin{aligned}\texttt{hessianThreshold} &= 400;\\ \texttt{octaves} &= 1;\\ \texttt{octaveLayers} &= 8;\end{aligned} \tag{3.5}$$

**STAR**:

$$\begin{aligned}\texttt{maxSize} &= 30;\\ \texttt{responseThreshold} &= 12;\\ \texttt{lineThresholdProjected} &= 10;\\ \texttt{lineThresholdBinarized} &= 20;\\ \texttt{suppressNonmaxSize} &= 10;\end{aligned} \tag{3.6}$$

**MSER**:

$$\begin{aligned} \texttt{delta} &= 5; \\ \texttt{minArea} &= 60; \\ \texttt{maxArea} &= 14400; \\ \texttt{maxVariation} &= 0.8; \\ \texttt{minDiversity} &= 0.3; \\ \texttt{maxEvolution} &= 300; \\ \texttt{areaThreshold} &= 1.01; \\ \texttt{minMargin} &= 0.003; \\ \texttt{edgeBlurSize} &= 3; \end{aligned} \tag{3.7}$$

**FAST**:

$$\begin{aligned} \texttt{threshold} &= 13; \\ \texttt{nonmaxSuppression} &= \mathit{true}; \end{aligned} \tag{3.8}$$

The evaluation was conducted following way:

1. Load the source and the target image (`cv::imread`)
2. Detect keypoints in both images (`cv::FeatureDetector::detect`)
3. Extract descriptors from both images (`cv::DescriptorExtractor::compute`)
4. Find matches between keypoints (`cv::DescriptorMatcher::match`)
5. Convert keypoints from `std::vector<cv::KeyPoint>` to `std::vector<cv::Point2f>`
6. Compute homography using RANSAC and get the inlier mask (`cv::findHomography`)
7. Sum the element in the inlier mask to get number of inliers

Graphs in figures 3.13 and 3.14 shows the statistics. Generally can be said, that, in the sense of the number of inliers, the SIFT feature detector works best with the SIFT descriptor extractor, SURF feature detector works best with the SURF descriptor extractor and the rest of the feature detectors works best with the SIFT descriptor extractor.

The combination of the FAST feature detector and the SIFT descriptor extractor gains the biggest number of inliers due to detecting a large number of features. The MSER detects very few features, but those are the good features, that end up to be inliers often.

Various settings of each feature detector were tested to find the optimal setting. Each feature detector was used with the most suitable descriptor extractor as described above. Results were observed, and the following conclusion has been drawn.

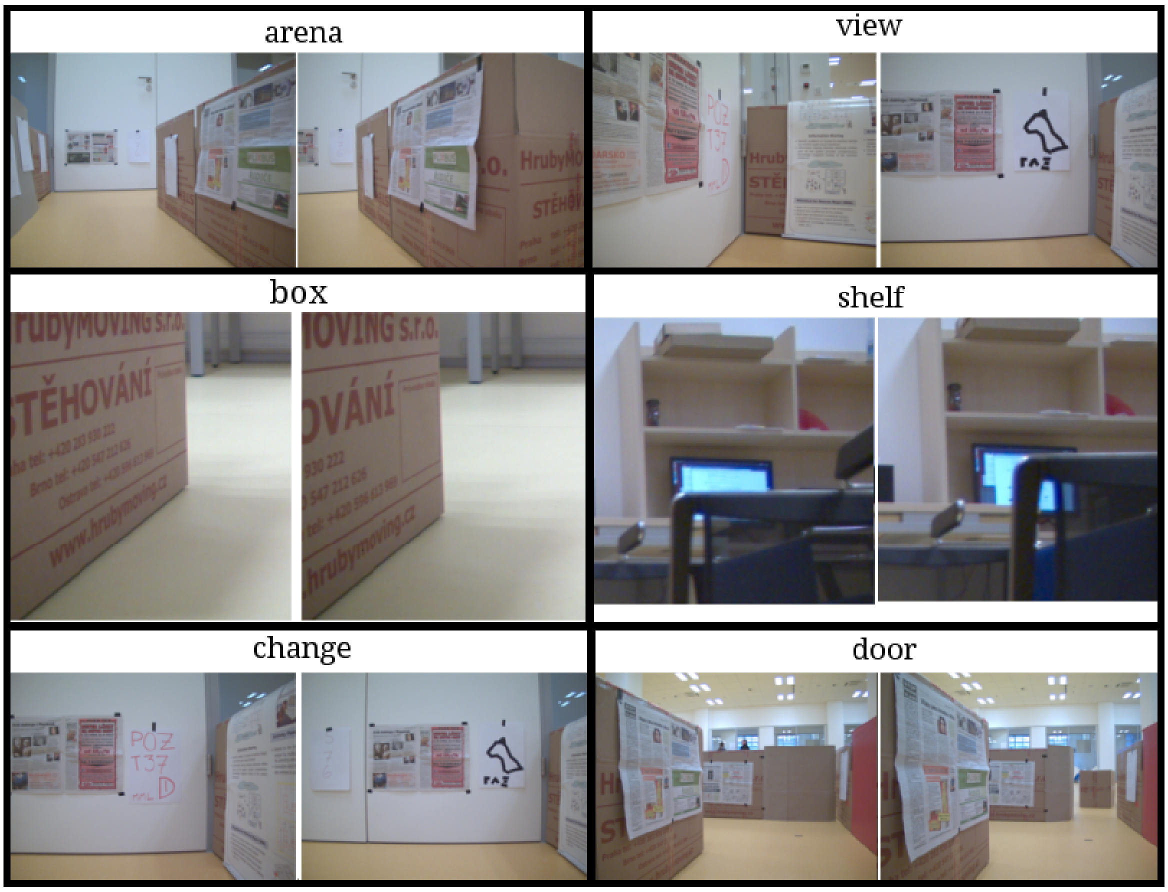


Figure 3.12: Pairs of images used to evaluate combinations of feature detectors and descriptor extractors.

The SIFT feature detector evaluation of was run on 864 settings. The used setting proved to perform optimally. Effect of `contrastThreshold` value on number of detected features can be seen in table 3.3. The bigger the value of `edgeTheshold` the more features are detected, but above the value of 3, the relative number of inliers stays about the same. Around the value of 3, the number of features (in the image of resolution 1280x960) is in thousands. `sigma` of 1.6 is also the best compromise between the number of detected features and inliers.

The bigger `hessianThreshold` in SURF setting, the fewer features is detected and the bigger the relative number of inliers is. At the value of 400, the number of detected features (in the image of resolution 1280x960) is around thousands. The `octaveLayers` values show a tendency to cause the maximum relative number of inliers between values 5 and 9. Otherwise the bigger `octaveLayers` the more detected features. At those values of `octaveLayers`, the maximum relative number of inliers is produced with `octaves` value of 4. The evaluation has been run on about 380 settings.

The STAR detector has a lot of parameters.The evaluation has been run on 2250 settings. The bigger the `maxSize` value is, the better performance. The `responseThreshold` at value of 8

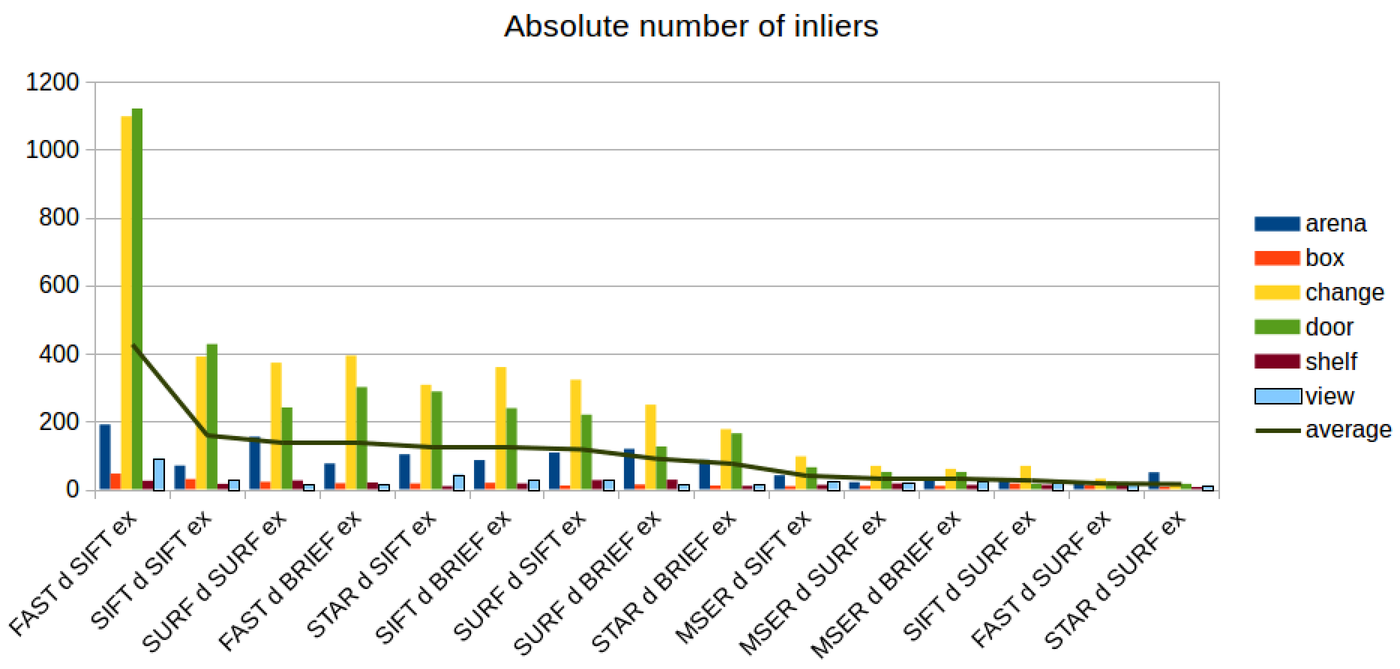


Figure 3.13: Absolute number of inliers in descending order.

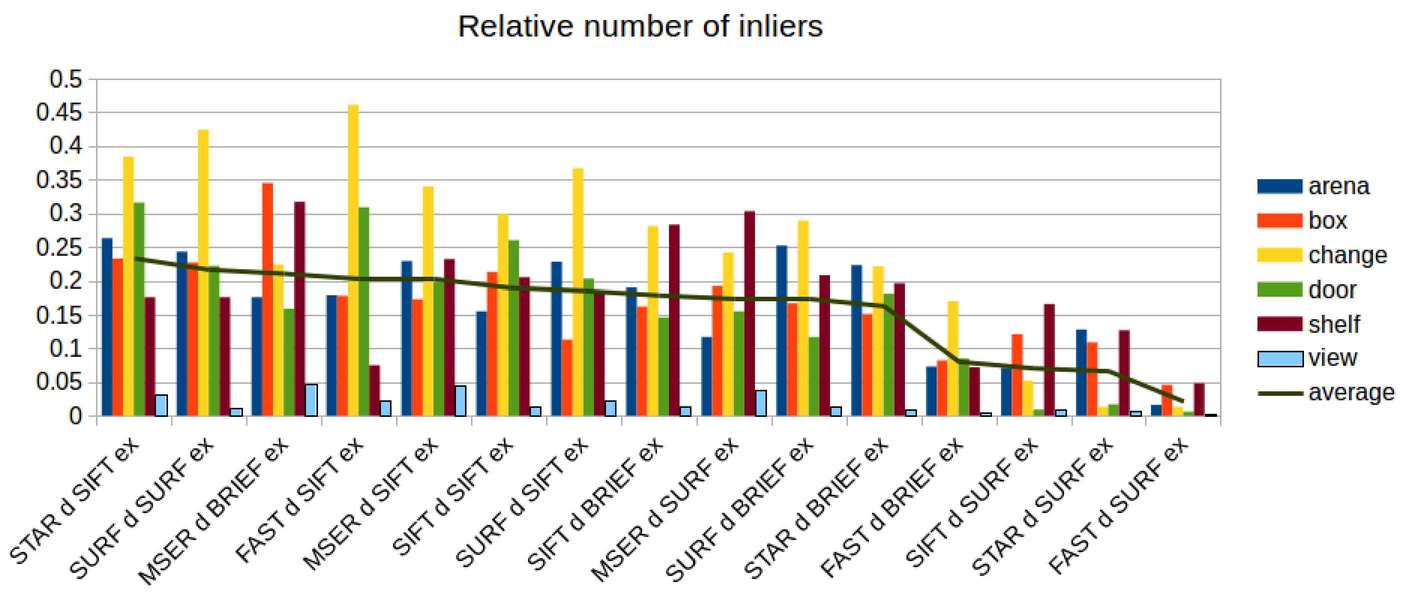


Figure 3.14: Relative number of inliers in descending order.

detects in a 1280x960 image around 1000. The higher this parameter is, the lower number of inliers it detects. The `lineThresholdProjected` and `lineThresholdBinarized` show direct proportional correlation with the number of detected features. The `supressNonmaxSize`

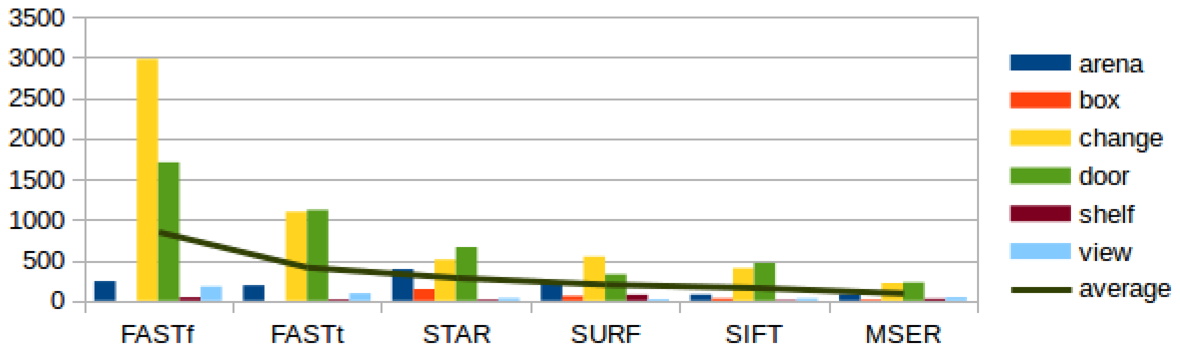


Figure 3.15: Absolute number of inliers in descending order.

| contrastThreshold | #features |
|---|---|
| $> 1$ | 0 |
| 0.5-1 | 100s |
| 0.1-0.5 | 1000s |

Table 3.3: Effect of contrastThreshold value on number of detected features.

shows reverse correlation. Absolute and relative number of inliers taken into consideration, the best performing setting seems to be `STAR(30,5,16,25,8)`.

For the MSER detector 120 settings have been tested. The `maxEvolution`, `areaThreshold`, `minMargin` and `edgeBlurSize` seem to have no effect on the result. The optimal setting seems to be `MSER(5,40,14400,1,.1,300,1.01,.003,3)`, with those uninfluential parameters left to default.

The FAST detector parameter `threshold` above 8 causes a quick decrease in found features. However, the value of 13 makes the detector to find ten thousands of features. With the value of 20, there are still thousands of features found. With the value of 8 and above the relative number of inliers significantly increases. The parameter `nonmaxSuppression` disables detecting features too close to other feature. This sounds useful, but even though the detector with enabled suppression detects 10 times fewer features, the relative number of inliers is also smaller than when the suppression is disabled. The evaluation has been run on 16 settings. For the following analysis `FAST(false,20)` and `FAST(true,13)` were used.

Using the settings described above, the absolute and the relative number of inliers is shown in Figures 3.15 and 3.16. The FAST detector with the suppression disabled (FASTf) has the highest relative and the absolute number of inlier of all the detectors. The FAST detector with the suppression enabled (FASTt) still finds hundreds of inliers and scores second. After this analysis, FAST and SURF feature detectors could be considered the best options, however, after adding time measurements to the evaluation, SURF detector proved to be the one of the slowest.

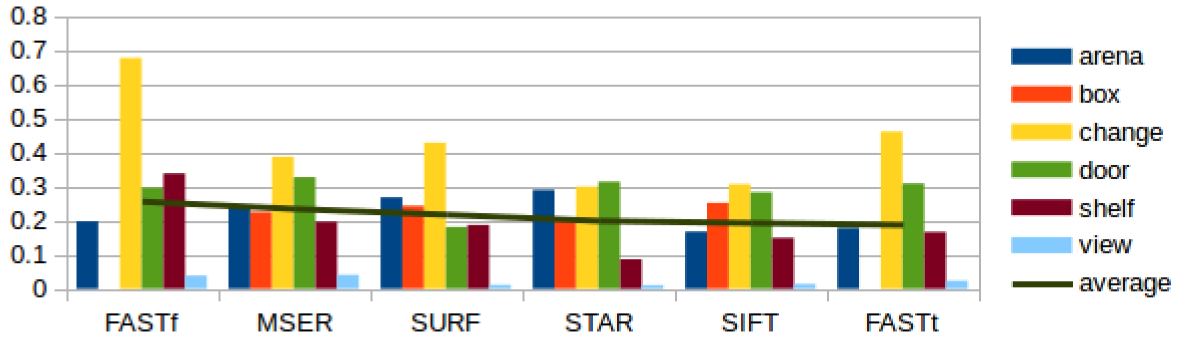


Figure 3.16: Relative number of inliers in descending order.

## 3.6 Image Viewpoint Transformation

Aside of number of inliers, the quality of the transformation might be a valuable information. One of the possible approaches might be transforming all pixels $p_x$ from the source image *img_src* into the viewpoint of the target image *img_dst* and sum the euclidean distance of each pair of pixel as $\frac{1}{n}\sum_x^n (img\_dst(p_{x,tf}) - img\_tf(p_{x,tf}))^2$, excluding those pixels of target image, that does not have a representation in the transformed image. The process is depicted in Figure 3.17. This approach has been tested but the method didn't discriminate worse transformations consistently.

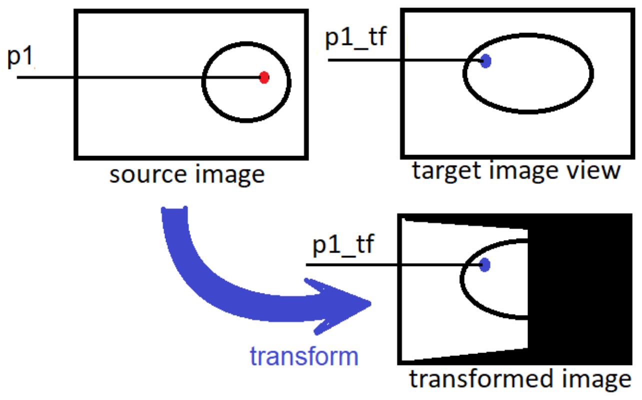


Figure 3.17: Pixels transformation process.

## 3.7 Manual Selection of Keypoints

An error calculation method that uses only manually selected points in the image instead of all pixels was implemented. While the pixel method did not show very reliable results, the manually selected point transformation method works well.

The manual keypoint selection has following steps:

- A window with the source and the target image is shown
- A person clicks on 10 selected keypoints in both images in the same order
- Windows with the source and the target image zoomed in on the selected keypoint are shown for every keypoint
- A person adjusts the position of the keypoint in both images
- The keypoints are stored into a file

The zoomed in keypoint selection is shown in Figure 3.19. The example of manually selected keypoints in a pair of images is shown in figures 3.18 on the arena pair of images.

The keypoints are loaded from the file, and the source image keypoints are transformed into the target image's viewpoint. The error is then computed as the distance between corresponding keypoints in the target and the transformed image as described in 3.9.

$$\sqrt{\sum (p_{dst}(i) - p_{tf}(i))^2} \tag{3.9}$$

where $p_{dst}(i)$ is the i-th keypoint position in the target image and $p_{tf}(i)$ is the i-th keypoint position in the transformed image.

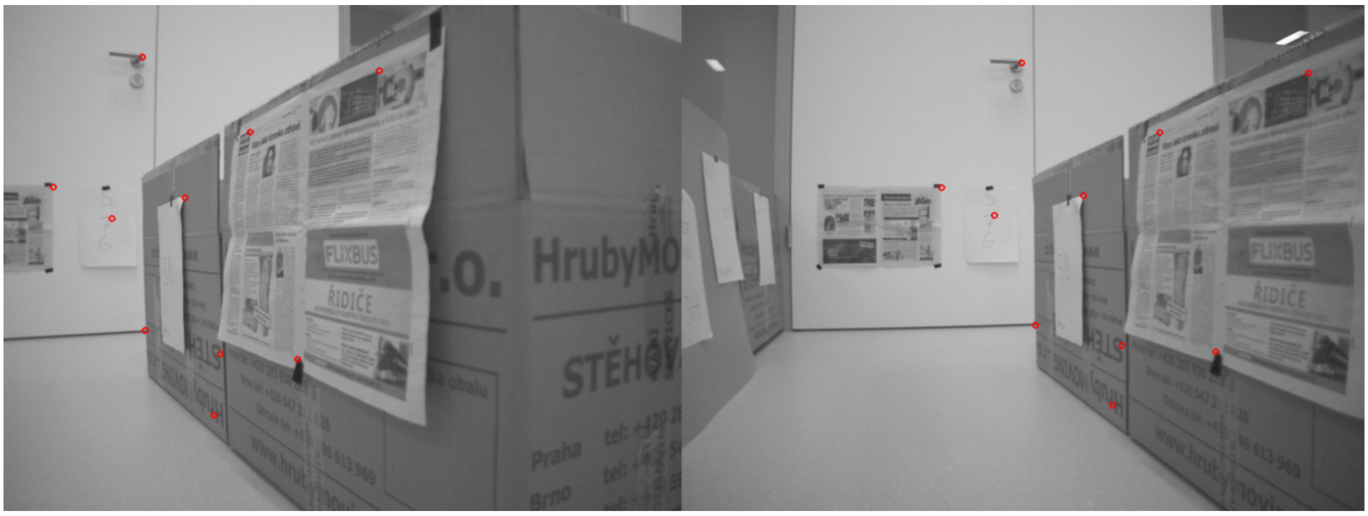


Figure 3.18: An example of manually selected keypoints on ARENA pair of images.

## 3.8 Size of Images

Another parameter that might affect results is the size of the image. Smaller images should be processed faster. The question is whether using smaller images will result in a bigger

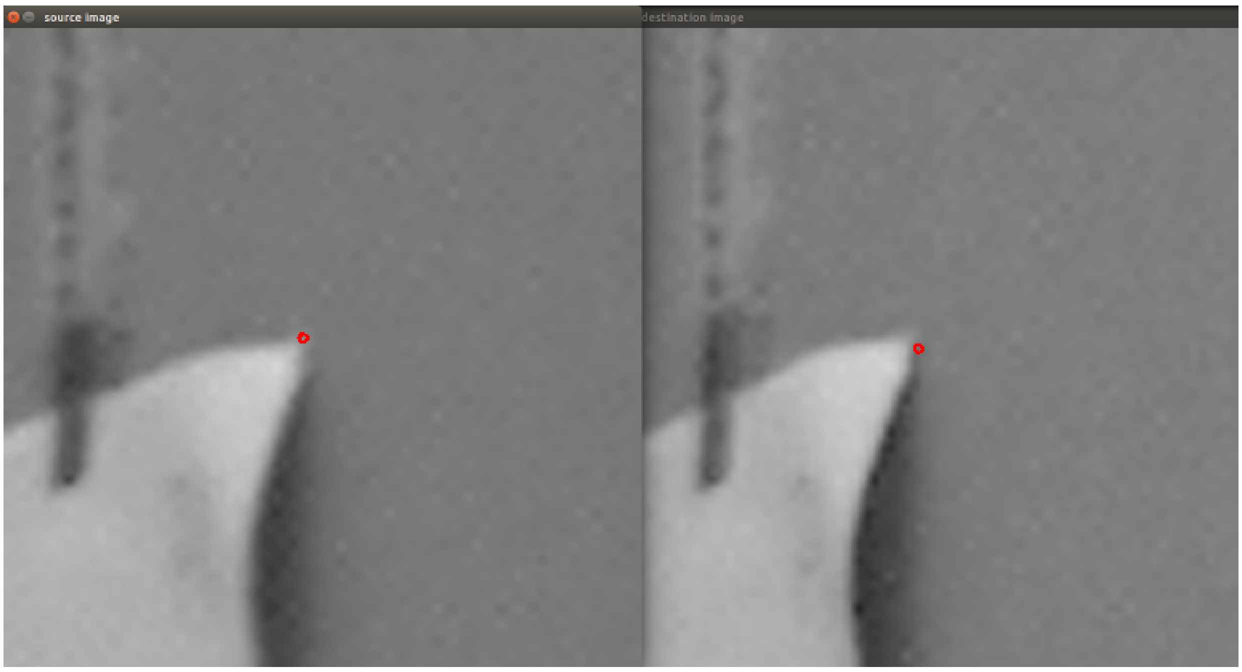


Figure 3.19: Zoomed in manual selection of a keypoint.

error and whether the different configuration of feature detector might be more successful. Comparison of the number of detected features, number of inliers, the relative number of inliers and computation time on 2 sets of image pairs were made. The first set contained the big images with the resolution of 1280x960 and the second set contained the small images with the resolution of 640x480.

The small images get processed 4 times faster and have a larger relative number of inliers. Moreover, the small images have a higher number of successful transformations. Statistics for the set of small images are shown in Figure 3.20

The `FAST(false,20)` seems to be the best performing setting for small images. The most similarly performing SIFT setting is `SIFT(0,4,.02,10,1.2)`, which is about 200 milliseconds slower, has a similar number of inliers, but a twice lower relative number of inliers.

The statistics of FAST detector show same tendencies for the small and big images in every aspect. The FAST detector without suppression detects too many keypoints, and their processing is too slow. The FAST detector with the `threshold` of 30 has a bigger error than a FAST detector with lower values of the `threshold`.

On the set with the small images the `FAST(20, true)` has duration 0.16 s on average, while the fastest SIFT has duration around 0.3 s. The errors are the same, but the duration is 2 times shorter for FAST. The transformations when using `FAST(20,true)` detector are shown in Figure 3.21

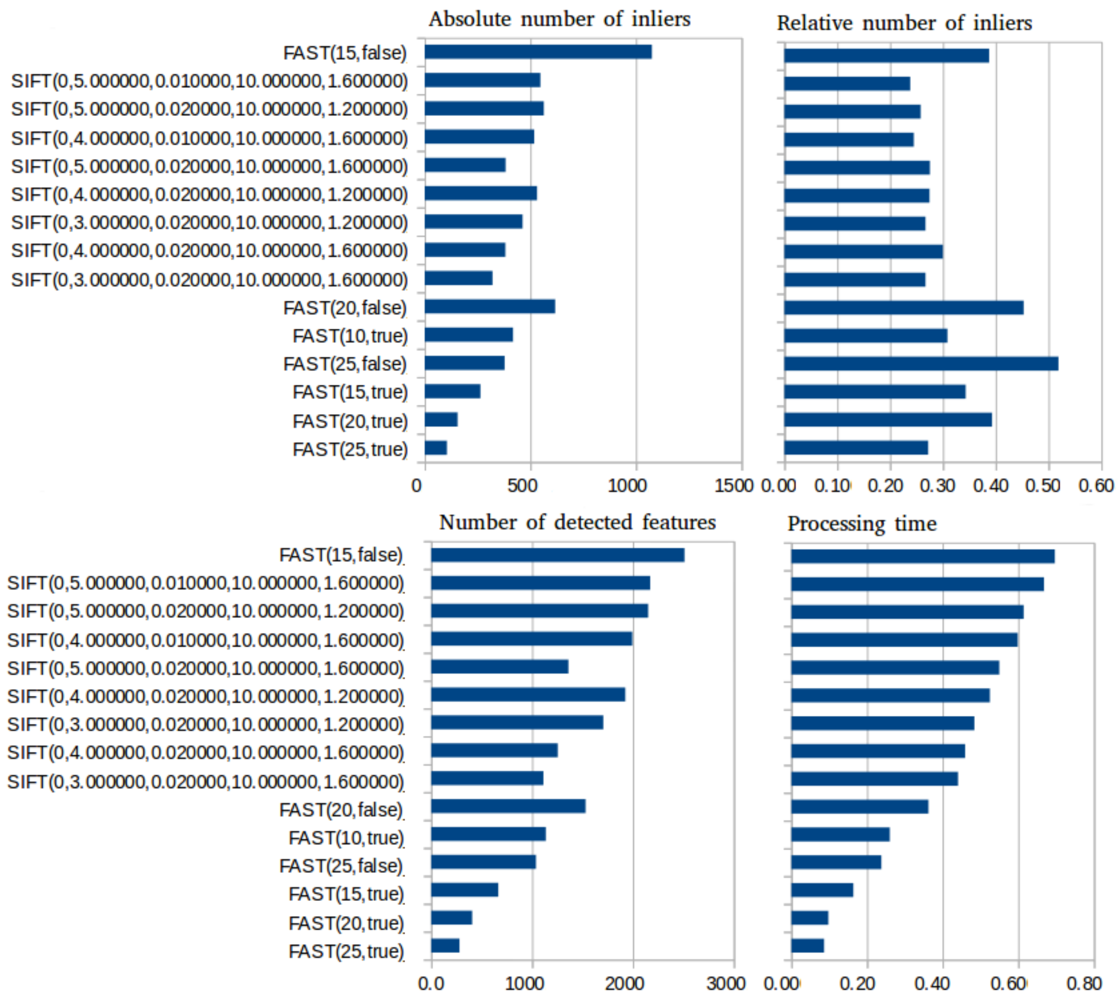


Figure 3.20: Statistics for best-performing settings of feature detectors on small images sorted from the longest processing time to the shortest. Average of all pairs of images.

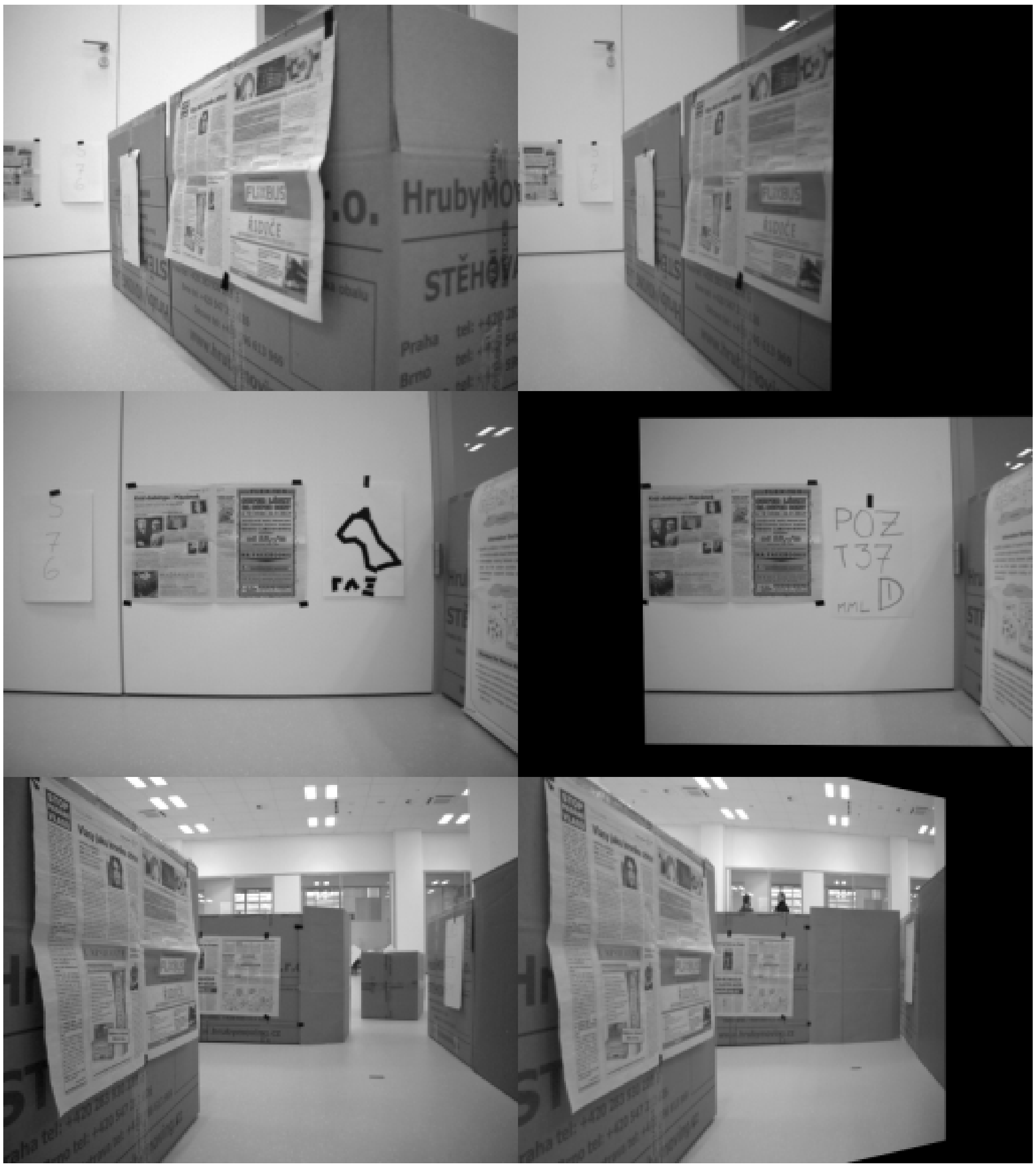


Figure 3.21: Transformation with FAST(20,true).

# Chapter 4

# Solution Description

First, the difference image change detection method (as discussed in Section 2.3) was implemented in C++ using the OpenCV library. The change detection was supposed to better the localization in cases where there is a change in the scene. The chosen approach is to assign weights to descriptors. The weight of the descriptors in the area that changes a lot will gradually decrease leaving the descriptor insignificant to the localization task.

## 4.1 Difference Image Change Detection

This method was supposed to localize the areas of changes. The process of change detection consists of following steps:

- Load source and target images in B&W
- Resize images to 640x480 (`cv::resize`)
- Blur the images
- Normalize images to 0-255 values (`cv::normalize`)
- Find homography matrix $H$ (`cv::findHomography`)
- Transform the source image to target image viewpoint (`cv::warpPerspective`)
- Normalize the transformed image to 0-255 values (`cv::normalize`)
- Create mask for pixels of the target image that are not represented in the transformed image
- Multiply the mask and the target image, so that only pixels represented in both target and transformed image are being processed (`cv::multiply`)
- Create difference image by subtracting the target and the transformed image (`cv::absdiff`)
- Normalize the difference image to 0-255 values (`cv::normalize`)
- Blur the difference image (to soften edges)

- Apply the threshold to get the change mask

Results for the DOOR and CHANGE pairs are shown in Figures 4.1 and 4.2 respectively. This algorithm is able to detect significant changes (as bold black tape on the white door), but all the fine details are lost. It is observable that in the DOOR pair the method detects a piece of column exposed after opening the door, but the box behind the door itself is too similar to the color of the door and stays undetected.

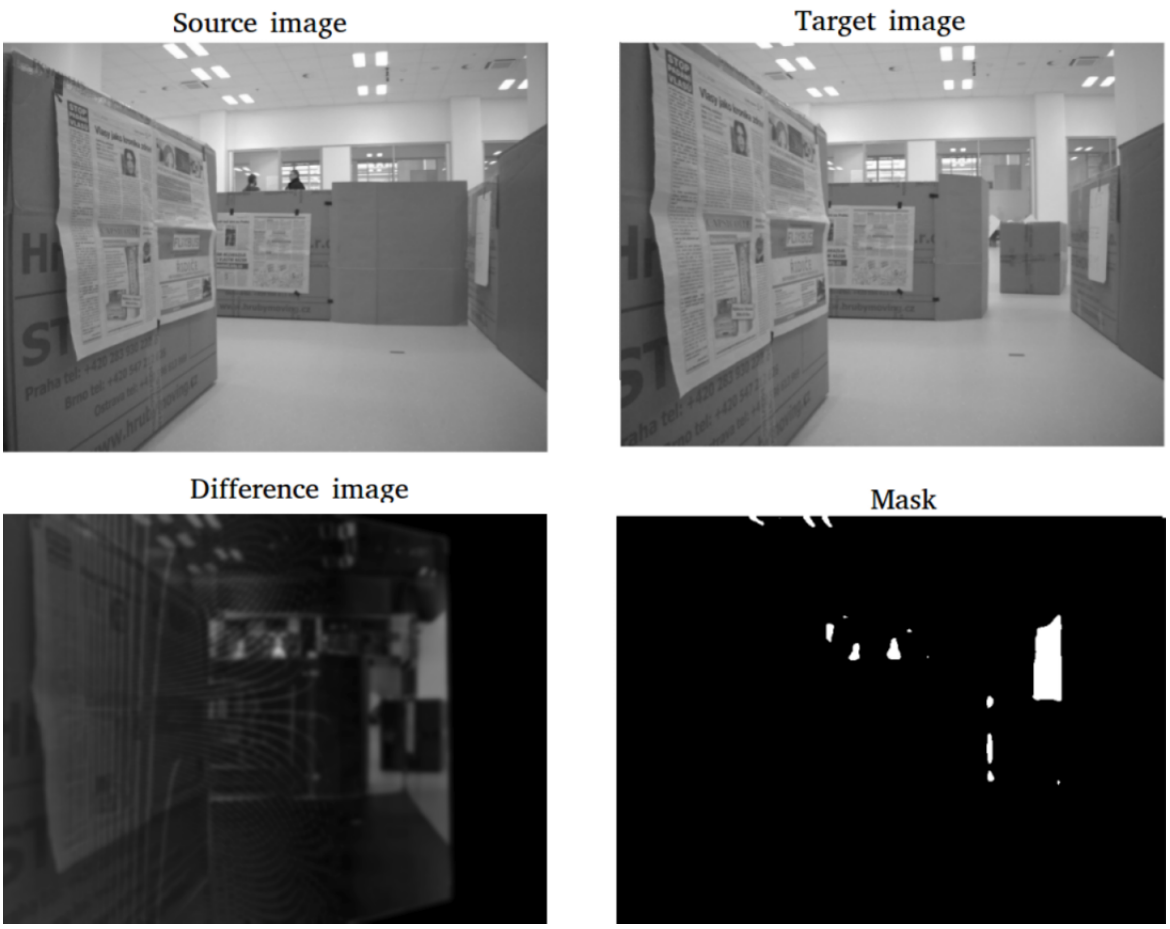


Figure 4.1: Difference image from the DOOR image pair.

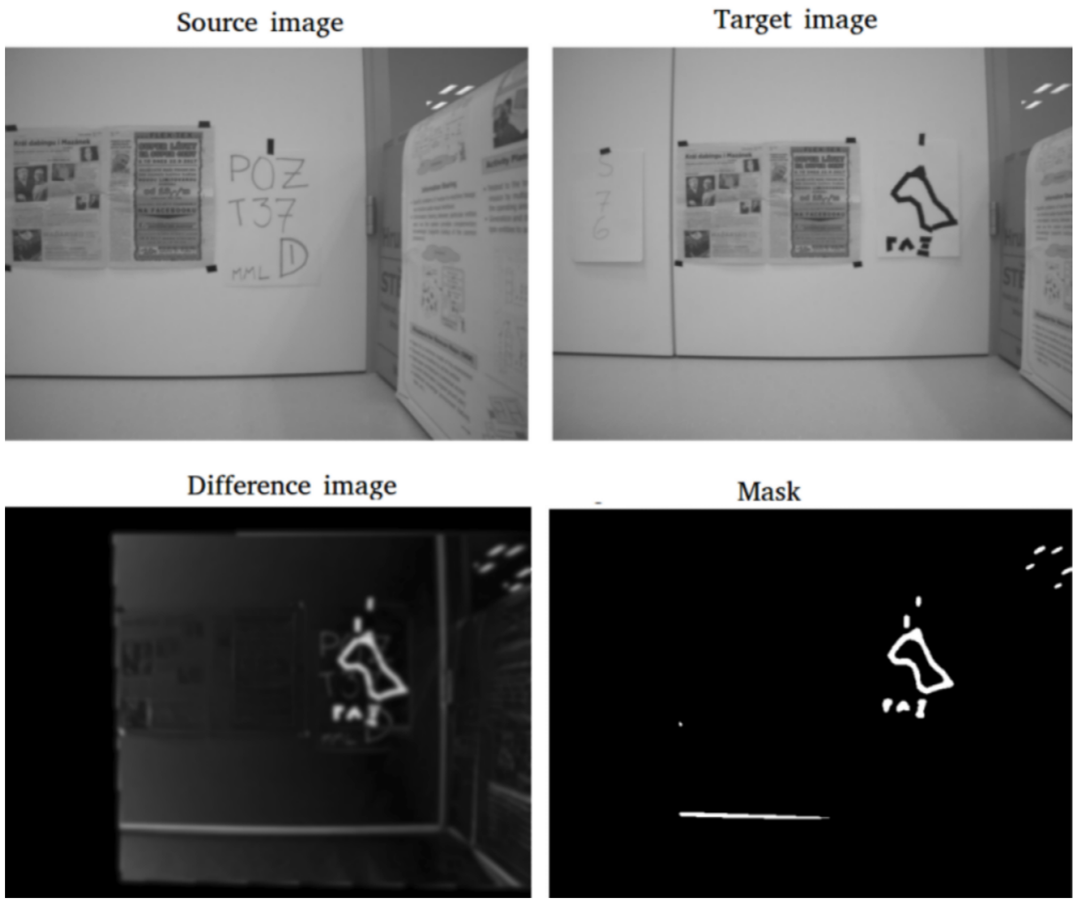


Figure 4.2: Difference image from the CHANGE image pair.

## 4.2 Keypoint Descriptor Distance Change Detection

In this section a concept of a change detection using descriptor distances is explained. The process consists of following steps:

- Load source (src) and target (dst) images in B&W
- Resize images to 640x480 (`cv::resize`)
- Blur the images
- Normalize images to 0-255 values (`cv::normalize`)
- Detect the keypoints $kp_{src}$ and $kp_{dst}$ with the `FAST(13,true)` feature detector (`cv::cv::FeatureDetector::detect`)
- Extract the descriptors $\delta_{src}$ and $\delta_{dst}$ using the SIFT descriptor extractor (`cv::DescriptorExtractor::compute`)
- Find homography matrix $H$ (`cv::findHomography`)

- Transform the source image into the viewpoint of the target image (`cv::warpPerspective`)
- Extract descriptors from the transformed image at the $kp_{dst}$
- For every keypoint $kp_{dst}(i)$ compute Euclidean distance between descriptors of the target and the transformed image

The distance of the descriptors is showed in Figure 4.3 as the radius of the circles. The process is described in one direction only, but the detection is needed to be done also from the target image to the source image to detect changes in areas, there the source image didn't have any keypoints.

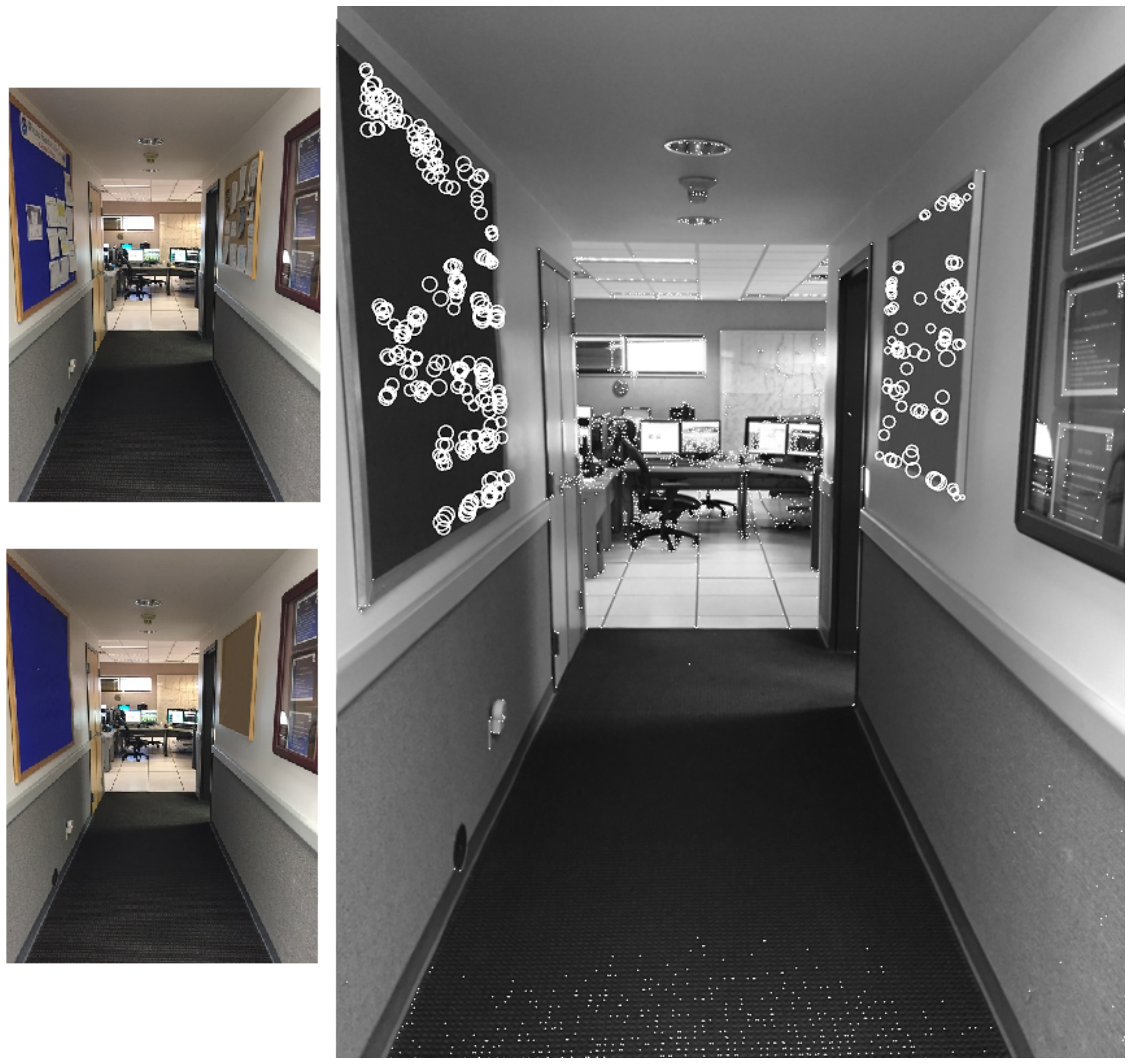


Figure 4.3: Keypoint distance change detection method showed on indoor enviroment picture. The source and target images are shown in the left column.

## 4.3 Matched Keypoints Localization

The localization method has been implemented using FAST feature detector and SIFT descriptor extractor. The first prototype was without weighting. Following methods adopted weighting. There were 3 methods with weighing implemented each of which presents a different way of weights update.

### 4.3.1 Without Weighting

This method works the following way. Images from the database are all processed into the descriptor and position files. The raw SIFT descriptors of each database image are stored into a file in the form of a matrix with an index in the first column to denote which image does the group of descriptors belong to. The descriptor matrix is $128+1$ columns wide and $\sum_{i=1}^{N} n_i$ rows deep, where $N$ is number of images and $n_i$ is number of features (descriptors) of the image $i$. The position file consists of $N \times 3$ matrix with the x coordinate in the first column, the y coordinate in the second column and the angle in the third column (see Figure 4.4).

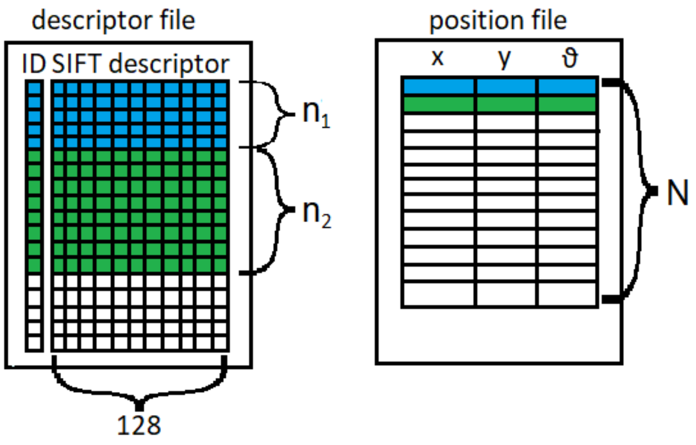


Figure 4.4: Structure of the descriptor and the position files for the localization task.

The query image gets processed into a matrix of SIFT descriptors just as every individual database image. Then the query image descriptors are matched to descriptors of every database image by Brute Force matcher. The Brute Force matcher finds the pairs of descriptors that are the closest to each other. The resulting distance of an image $d_i$ is an average distance of those pairs of closest descriptors between the query and the database image. (4.1) describes the distance computation. The $\delta_j^{db}$ is the j-th descriptor of the database image, and the $\delta_j^{q}$ is the respective descriptor of the query image. The query image is assigned the position of the closest database image.

$$d_i = \frac{\sum_{j=1}^{n} \sqrt{(\delta_j^{q} - \delta_j^{db})^2}}{n} \tag{4.1}$$

### 4.3.2 Weighting Using Distance Vector Normalization

The difference between the method without weighting and this method is that at the point when distances of the descriptors are being summed into the image distance $d_i$, every descriptor distance is multiplied by its weight as described by (4.2).

$$d_i = \frac{\sum_{j=1}^{n} w_j \sqrt{(\delta_j^q - \delta_j^{db})^2}}{n} \tag{4.2}$$

The weights get updated during every image distance computation. The update step is described by (4.3). The factor $\gamma$ determines how much can a weight change from the previous weight. The estimate of a new weight $\hat{w}$ is obtained by normalizing the distances of all descriptor pairs of the current pair of images (the query and the database image) to the interval $< 0, 1 >$ and inverting the value by subtracting the normalized vector from the vector of ones. This way the pair of descriptors with the smallest distance gets assigned the weight of value 1 and the pair of descriptors with the largest distance gets assigned the weight of value 0.

$$w(t+1) = \gamma w(t) + (1-\gamma)\hat{w}(t+1) \tag{4.3}$$

### 4.3.3 Weighing Using Past Values

This method keeps track of the previous value of distances. The current and the previous distance are subtracted and the difference is used as the feedback to adjust weights. (4.4) shows the update. The $l(i)$ is the descriptor distance at iteration $i$. The $\alpha$ is some scaling parameter. The resulting errors are in Table 6.7. If a weight is desired to be decreased by 0.1 and $\gamma = 0.9$, for example, the scaled difference of distances should be around 1. That would require $\alpha$ of a value of the difference. Maximum difference occurring during the test was around 300.

$$w(i+1) = \gamma w(i) + (1-\gamma)(l(i+1) - l(i))\alpha \tag{4.4}$$

### 4.3.4 Weighting Using Global Reference Value

A global reference value could be used to scale down the distance between descriptors that could be directly used as the new weight estimate $\hat{w}$. The weight estimate computation is described by (4.5), where $l_i$ is the distance of the i-th descriptor. The update is computed the same way as in (4.3). The maximum distances are typically around 430. This number can be used as an anchor to decide for the global `ref` value.

$$\hat{w} = \begin{cases} 1 - \frac{l_i}{\text{ref}} & \text{for } \frac{l_i}{\text{ref}} \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

## 4.4 Transformed Keypoints Localization

### 4.4.1 Keypoints Transformed into the Target Image Viewpoint

This method uses keypoints detected in the database image, transforms them into the viewpoint of the query image and computes their descriptor distance.

The method works following way:

1. Load database image descriptors $\delta_{db}$
2. Compute query image descriptors $\delta_q$
3. Find matches (`cv::DescriptorMatcher::match`)
4. Find homography matrix $H$ with the matched keypoints
5. Transform database keypoints into viewpoint of the query image $k_{tf} = Hk_{db}$
6. Compute descriptors $\delta_{tf}$ in the query image at the $k_{tf}$
7. For $k_{tf}$ within the image bounds compute distance between $\delta_{db}$ and $\delta_{tf}$
8. Repeat for all database images. Assign to the query image the position of the closest database image.

The SIFT feature descriptor is scale invariant, rotation invariant and is robust to change in viewpoint [13], therefore the descriptor of the feature should be the same in the query image and the database image even though the viewpoint and scale differs.

Some modifications have been made to better the results. Firstly, the relative number of inliers is computed, and the image pairs whose relative number of inliers is lower than some threshold (0.2 in this case) is assigned infinity distance. Second, the determinant of the homography matrix $H$ is computed, and image pairs that produce homography with determinant close to zero (lower than 0.2 in this case) are assigned infinity distance. The reason is to discard transformations that were unsuccessful. The sets of images discarded by those two modifications overlap, since most of the transformations with close to singular homography matrix also produce little inliers, but some cases are only caught by one of those modifications.

Figure 4.5 shows the correct keypoint transformation with non-singular homography matrix and the incorrect keypoint transformation with the homography matrix close to singular.

### 4.4.2 Target Image Transformed into the Source Image Viewpoint

This methods consists of following steps:

1. Load database image descriptors $\delta_{db}$ and keypoint $k_{db}$

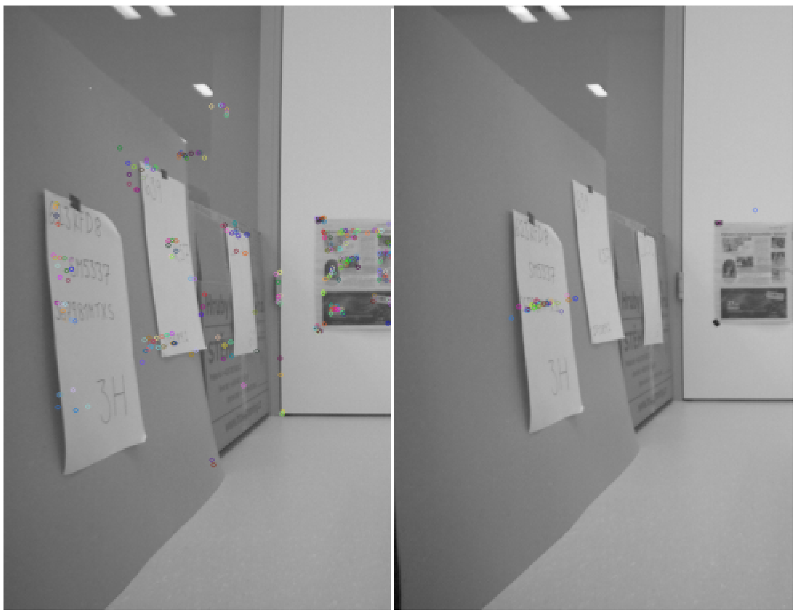


Figure 4.5: Example of correctly (on the left) and incorrectly (on the right) transformed keypoints. Homography matrix is non-singular in the first case and singular in the latter.

2. Compute query image descriptors $\delta_q$
3. Find matches (`cv::DescriptorMatcher::match`)
4. Find homography matrix $H$ with the matched keypoints
5. Transform query image into viewpoint of the database image
6. Compute descriptors $\delta_{tf}$ within the transformed image at the database keypoints.
7. For all $k_{db}$ compute distance between $\delta_{db}$ and $\delta_{tf}$

# Chapter 5

# Scene Setting and Recording

## 5.1 The Arena Setting

The arena built for the robot had a circular shape. It was built out of cupboard boxes. Posters were glued to the walls to add distinctive features. There was one poster that has been changed during the measurements and small cardboard doors that have been closed at the beginning of the recording and were opened during the recording. Those represent the change in the dataset.

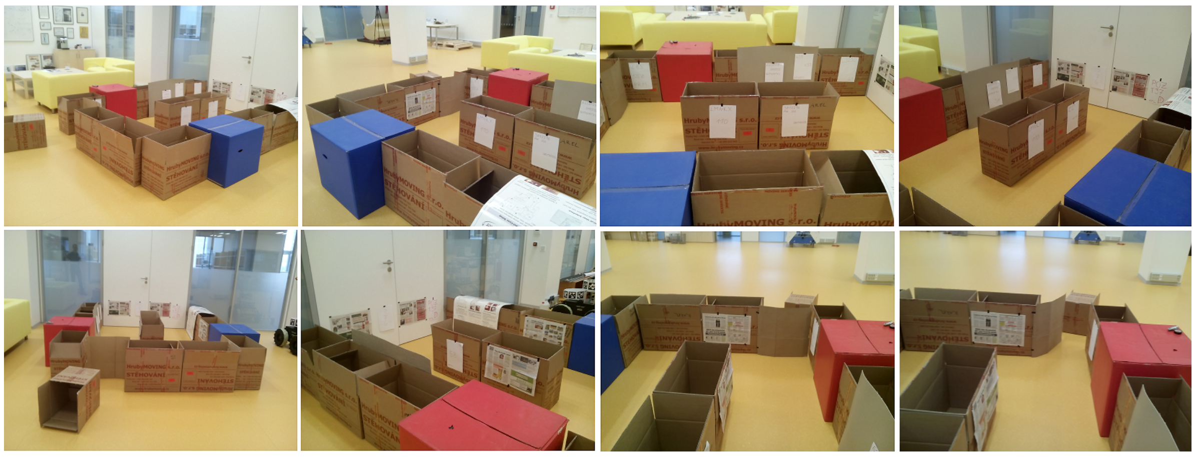


Figure 5.1: Cardboard arena used for localization data recording.

Pictures of the arena are shown in Figure 5.1. The scheme of the arena is depicted in Figure 5.2

## 5.2 Arena Recording

The recording setting was the same as described in Section 3.2.1. In the default, arena layout is a poster #1 glued to the wall and door is closed. In the layout with the change, the poster

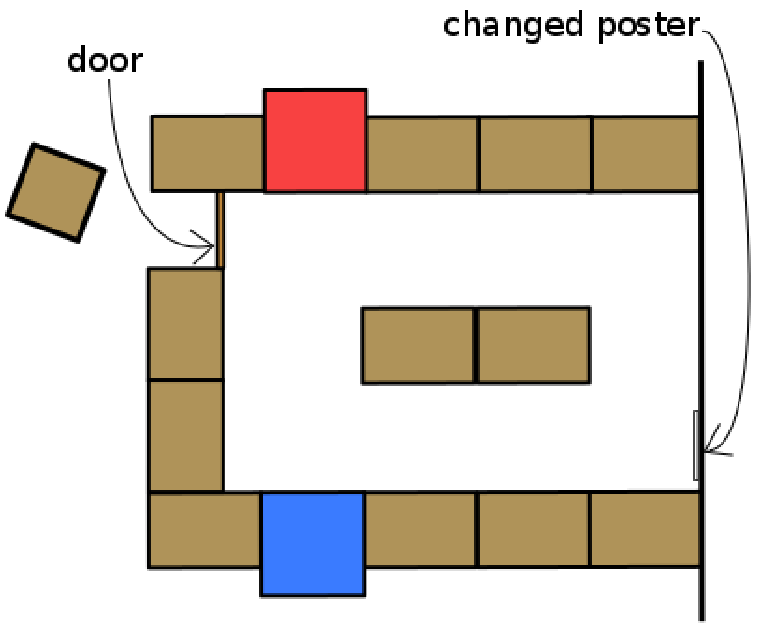


Figure 5.2: Arena scheme.

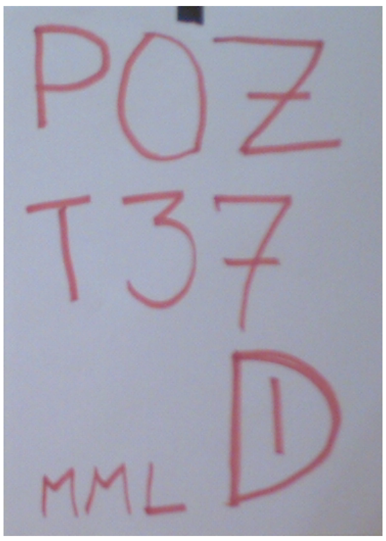
#1 is replaced with the poster #2 and door are open. Picture of the posters is in Figure 5.3

Three sets of measurements were conducted, each consisting of three measurements. The first set was recorded with the default arena layout. After this measurement, the robot was restarted. The gmapping node lost its previous information and started building a new map. Therefore gmapping map of the first and the second set differ. The second set was recorded with the default arena layout after few minutes of driving around the arena for the robot to build up a new map. The third set was recorded within the same map as the second set but with the changed layout.

The first record of a set was recorded while the robot drove around the arena in the clockwise direction. The second record is the counter-clockwise loop. The third measurement is the record of a robot driving to the other side of the arena, turning around and driving back the same way. Table 5.1 gives an overview of the records.

| set | map | change | measurement | direction |
|---|---|---|---|---|
| 1st | #1 | NO | 1st | clockwise |
| | | | 2nd | counter-clockwise |
| | | | 3rd | half way |
| 2nd | #2 | NO | 1st | clockwise |
| | | | 2nd | counter-clockwise |
| | | | 3rd | half way |
| 3rd | #2 | YES | 1st | clockwise |
| | | | 2nd | counter-clockwise |
| | | | 3rd | half way |

Table 5.1: Overview of arena measurements.



(a) poster #1

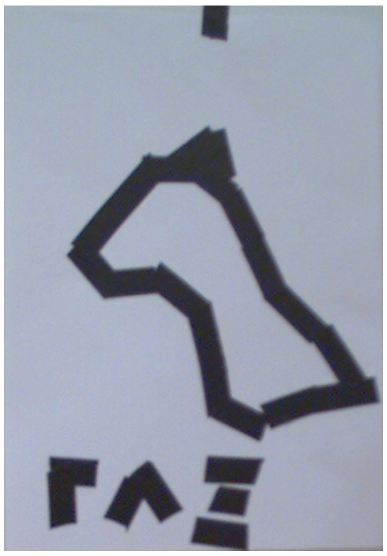

(b) poster #2

Figure 5.3: Posters that are being switched to introduce change into dataset.

# Chapter 6

# Results

In this chapter, the results of the implemented methods described in Chapter 4 are presented.

## 6.1 Matched Keypoints Localization

### 6.1.1 Without Weighting

The matched keypoints localization method without weighing was tested on three training datasets with the same test dataset. The test dataset was the 3rd record from the 2nd set. The first training dataset was only the 1st record from the 2nd set, the second training dataset was only the 2nd record from the 2nd set and the third training dataset was both 1st and 2nd records from the second set. Since the 3rd record captures half of the clockwise loop and half of the counter-clockwise loop, the best results should be gained by the third training dataset. The errors are in Table 6.1 and estimates in Figure 6.1.

Next test was made on a training dataset of all 3 records from the 2nd set, and test dataset were all the remaining records. The estimated and the true positions for the 2nd record from the 3rd set are in Figure 6.2. The errors and distances of this dataset are shown in Table 6.2.

| training dataset | error$_x$ | error$_y$ | error$_\theta$ | distance |
|---|---|---|---|---|
| clockwise | 0.426 | 0.471 | 0.897 | 0.6351 |
| counter-clockwise | 0.349 | 0.228 | 0.880 | 0.4169 |
| both loops | 0.117 | 0.130 | 0.655 | 0.1749 |

Table 6.1: Error in x and y coordinate, error in angle and the average distance. Testing dataset is the 3rd record from the 2nd set. Without weighting

### 6.1.2 Weighting Using Distance Vector Normalization

Resulting errors for different values of $\gamma$ are printed in Tables 6.3, 6.4, 6.5 and 6.6.

The weighted method for the $\gamma = 0.95$ had average distance between the true and the estimated positions in the 2nd and 3rd record shorter than the method without weighting,

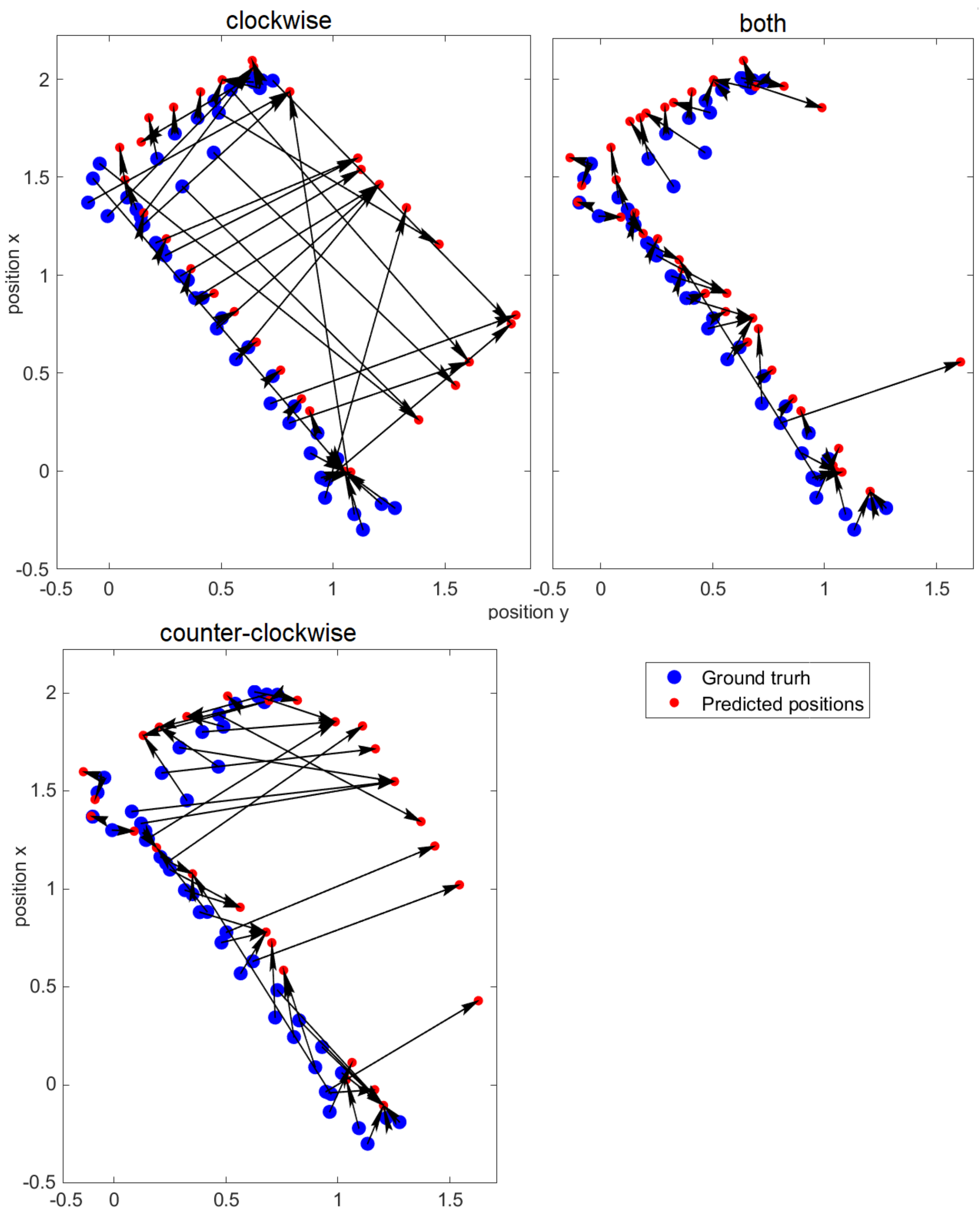


Figure 6.1: The true and predicted positions with only the clockwise loop as the training dataset, only the counter-clockwise loop as the training dataset and both clockwise and counter-clockwise loops as the training dataset. The test dataset is the half-way run. All used records are from the second set.

| Test dataset | | error$_x$ | error$_y$ | error$_\theta$ | distance |
|---|---|---|---|---|---|
| 1st set | clockwise | 0.2679 | 0.1580 | 0.4132 | 0.3110 |
| | counter-clockwise | 0.3085 | 0.2411 | 0.6791 | 0.3915 |
| | third measurement | 0.2792 | 0.1660 | 0.4518 | 0.3248 |
| 3rd set | clockwise | 0.0917 | 0.0980 | 0.1623 | 0.1342 |
| | counter-clockwise | 0.1609 | 0.3087 | 0.3923 | 0.3481 |
| | third measurement | 0.1462 | 0.3360 | 0.7448 | 0.3664 |

Table 6.2: Error in x and y coordinate, error in angle and the average distance. Training dataset is the 2nd set. Without weighting.

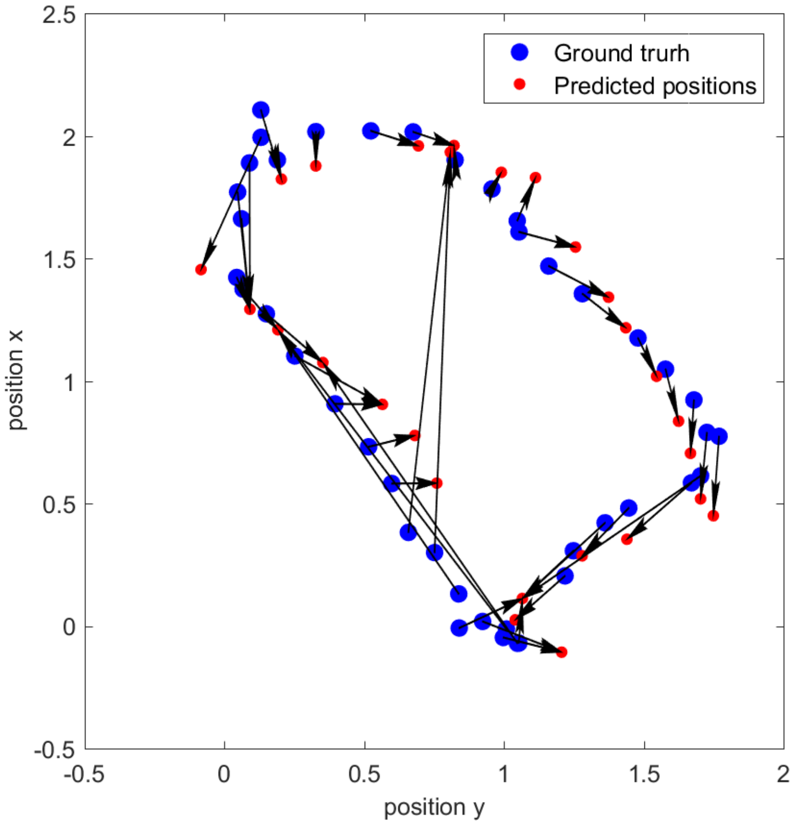


Figure 6.2: The true and the predicted positions for the second record (counter-clockwise) from the third set. Without weighting.

but larger in the 1st record, where the robot does not face directly any of the changes. Ideally, the results in a nonchanging environment should stay the same as without weighting and only in case of an often changing area the results should improve. The normalization across the current descriptor distances puts some weights estimates on zero even if all the distances are small (the images are the same, and no change has occurred). That might be the reason of the worse results.

| Test dataset | | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distance |
|---|---|---|---|---|---|
| | clockwise | 0.1081 | 0.1074 | 0.1357 | 0.1524 |
| 3rd set | counter-clockwise | 0.2019 | 0.2466 | 0.5013 | 0.3187 |
| | third measurement | 0.1347 | 0.3185 | 0.4734 | 0.3458 |

Table 6.3: Error in x and y coordinate, error in angle and the average distance. Training dataset is the 2nd set. Weighted with $\gamma = 0.95$

| Test dataset | | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distance |
|---|---|---|---|---|---|
| | clockwise | 0.1272 | 0.1196 | 0.2670 | 0.1746 |
| 3rd set | counter-clockwise | 0.2032 | 0.2570 | 0.7501 | 0.3276 |
| | third measurement | 0.2090 | 0.3612 | 0.6268 | 0.4173 |

Table 6.4: Error in x and y coordinate, error in angle and the average distance. Training dataset is the 2nd set. Weighted with $\gamma = 0.90$

| Test dataset | | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distance |
|---|---|---|---|---|---|
| | clockwise | 0.1965 | 0.1818 | 0.5589 | 0.2677 |
| 3rd set | counter-clockwise | 0.2792 | 0.3115 | 0.6235 | 0.4183 |
| | third measurement | 0.1925 | 0.3523 | 0.6801 | 0.4015 |

Table 6.5: Error in x and y coordinate, error in angle and the average distance. Training dataset is the 2nd set. Weighted with $\gamma = 0.80$

| Test dataset | | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distances |
|---|---|---|---|---|---|
| | clockwise | 0.1908 | 0.1994 | 0.5571 | 0.2760 |
| 3rd set | counter-clockwise | 0.3061 | 0.3621 | 0.8620 | 0.4741 |
| | third measurement | 0.2573 | 0.4171 | 0.8591 | 0.4901 |

Table 6.6: Error in x and y coordinate, error in angle and the average distance. Training dataset is the 2nd set. Weighted with $\gamma = 0.70$

### 6.1.3 Weighting Using Past Values

For the values of $\alpha$ of 700 and higher the average distance between the true and the estimated positions is shorter than with the method without weighting and with weighting by distance vector normalization for the same $\gamma$ value.

### 6.1.4 Weighting Using Global Reference Value

The errors for the different values of `ref` can be seen in Table 6.8. For the `ref` of 450 and higher the results are better than the original without weighting and all the other weighting methods.

Estimated positions for the parameter `ref` of 500 can be seen in Figure 6.3. The two positions where the robot is looking toward the open door got estimated correctly with this method,

| $\alpha$ | error$_x$ | error$_y$ | error$_\theta$ | distance |
|---|---|---|---|---|
| 30 | 0.3554 | 0.8336 | 2.1256 | 0.9062 |
| 50 | 0.1913 | 0.4303 | 0.8834 | 0.4709 |
| 100 | 0.1843 | 0.3667 | 0.8463 | 0.4104 |
| 200 | 0.1592 | 0.2787 | 0.7590 | 0.3210 |
| 300 | 0.1592 | 0.2787 | 0.7590 | 0.3210 |
| 500 | 0.1580 | 0.2746 | 0.7586 | 0.3883 |
| 700 | 0.1580 | 0.2746 | 0.7586 | 0.3168 |
| 800 | 0.1577 | 0.2737 | 0.6092 | 0.3159 |
| 1000 | 0.1577 | 0.2737 | 0.6092 | 0.3159 |

Table 6.7: Error in x and y coordinate, error in angle and average error from all coordinates combined and the average distance. Training dataset is the 2nd set. The test dataset is the 3rd record from the 3rd set. Weighted with $\gamma = 0.95$. Past values method

but were estimated incorrectly with the method without weighting.

The errors for the second record from the 3rd set for different values of $\gamma$ are shown in Table 6.9. Decreasing the $\gamma$ caused worse performance.

This weighting method is among the all implemented methods the one that showed the most significant decrease in estimate error.

| ref | error$_x$ | error$_y$ | error$_\theta$ | distance |
|---|---|---|---|---|
| 370 | 0.2490 | 0.2946 | 0.5436 | 0.3857 |
| 400 | 0.1961 | 0.2416 | 0.4023 | 0.3112 |
| 430 | 0.1647 | 0.2233 | 0.4310 | 0.2775 |
| 450 | 0.1457 | 0.1939 | 0.4278 | 0.2425 |
| 500 | 0.1346 | 0.1653 | 0.4190 | 0.2132 |

Table 6.8: Error in x and y coordinate, error in angle and average error from all coordinates combined and the average distance. Training dataset is the 2nd set. The test dataset is the 2nd record from the 3rd set. Weighted with $\gamma = 0.95$. Global reference method.

| $\gamma$ | error$_x$ | error$_y$ | error$_\theta$ | distance |
|---|---|---|---|---|
| 0.9 | 0.1527 | 0.1962 | 0.4239 | 0.2486 |
| 0.8 | 0.1732 | 0.2350 | 0.4493 | 0.2919 |
| 0.7 | 0.1802 | 0.2852 | 0.3820 | 0.3374 |
| 0.6 | 0.1891 | 0.3852 | 0.3789 | 0.4291 |

Table 6.9: Error in x and y coordinate, error in angle and the average distance. The training dataset is the second set. The test dataset is the 2nd record from the 3rd set. Global reference method. ref = 500.

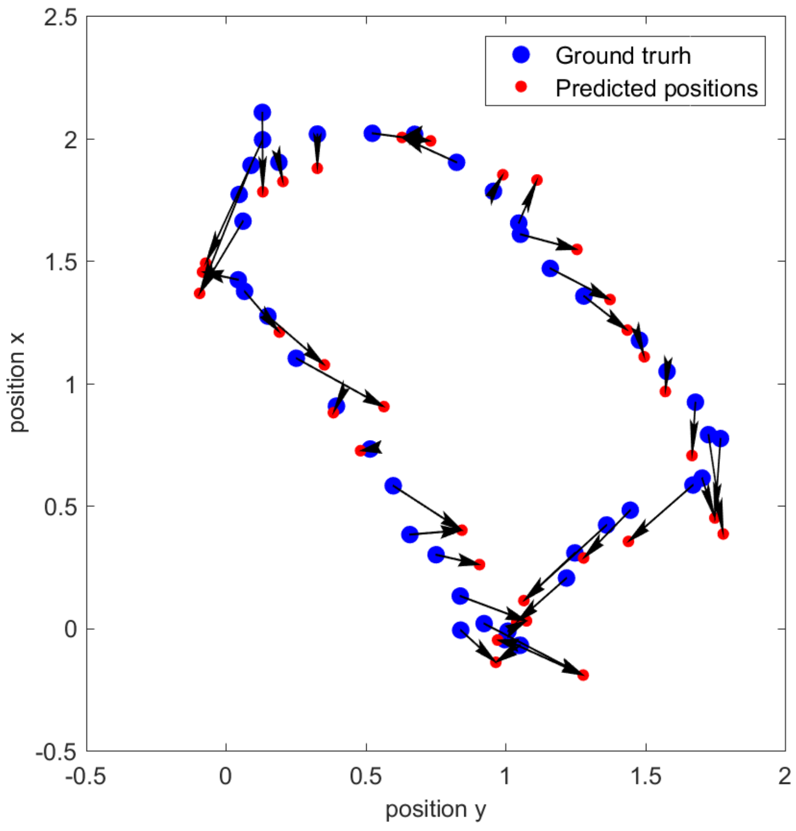


Figure 6.3: The true and the predicted positions for the second record (counter-clockwise) from the third set with the weighting parameters $\gamma = 0.95$ and $\text{ref} = 500$. Global reference method.

## 6.2 Transformed Keypoints Localization

### 6.2.1 Keypoints Transformed into the Target Image Viewpoint

Results of this method without weighting are shown in Table 6.10. Results for weighting are shown in Tables 6.11 ($\gamma = 0.9$) and 6.12 ($\gamma = 0.98$). This method doesn't result in shorter distances between the estimated and the true positions.

| record | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distance |
|---|---|---|---|---|
| 1 | 0.2624 | 0.3024 | 0.5688 | 0.4004 |
| 2 | 0.2005 | 0.3106 | 0.5758 | 0.3697 |
| 3 | 0.2063 | 0.4952 | 0.5280 | 0.5365 |

Table 6.10: The errors and distance. The test dataset is the third set. Keypoint transformation method without weighting.

| ref | rocord 1 | rocord 2 | rocord 3 |
|---|---|---|---|
| 200 | 0.4124 | 0.3599 | 0.5600 |
| 300 | 0.4256 | 0.3479 | 0.6230 |
| 400 | 0.4385 | 0.4060 | 0.6258 |
| 500 | 0.4783 | 0.4631 | 0.6410 |
| 600 | 0.3718 | 0.3754 | 0.6069 |
| 700 | 0.5122 | 0.4619 | 0.7397 |

Table 6.11: The distance between the estimated and the true positions.. The test dataset is the third set. Keypoint transformation method with weighting. $\gamma = 0.90$.

| ref | rocord 1 | rocord 2 | rocord 3 |
|---|---|---|---|
| 200 | 0.3718 | 0.3819 | 0.6081 |
| 300 | 0.3899 | 0.4072 | 0.5952 |
| 400 | 0.3507 | 0.4254 | 0.6773 |
| 500 | 0.3709 | 0.4358 | 0.6787 |
| 700 | 0.3499 | 0.4813 | 0.7454 |

Table 6.12: The distance between the estimated and the true positions. The test dataset is the third set. Keypoint transformation method with weighting. $\gamma = 0.98$.

### 6.2.2 Target Image Transformed into the Source Image Viewpoint

The results of this approach without weighting are shown in Table 6.13. The results are slightly better than for the keypoint transformation approach. Results for the weighting are in Table 6.14.

The image transformation method shows better results than the keypoint transformation method. However, even the image transformation method is worse than keypoint matching methods discussed above. Addition of weighting does not achieve better results than the keypoint matching method.

| measurement | $\text{error}_x$ | $\text{error}_y$ | $\text{error}_\theta$ | distance |
|---|---|---|---|---|
| 1 | 0.2050 | 0.2141 | 0.5823 | 0.2964 |
| 2 | 0.1888 | 0.2620 | 0.5840 | 0.3229 |
| 3 | 0.1832 | 0.4879 | 0.5427 | 0.5212 |

Table 6.13: The errors and distance. The test dataset is the third set. The image transformation method without weighting.

| ref | record 1 | record 2 | record 3 |
|---|---|---|---|
| 100 | 0.2994 | 0.3072 | 0.5406 |
| 200 | 0.3073 | 0.3109 | 0.5488 |
| 300 | 0.3058 | 0.3114 | 0.5546 |
| 400 | 0.2728 | 0.3069 | 0.5397 |
| 600 | 0.3073 | 0.3311 | 0.5354 |
| 800 | 0.2956 | 0.3119 | 0.5463 |
| 1000 | 0.3019 | 0.3119 | 0.5445 |

Table 6.14: The distance. The test dataset is the third set. Image transformation weighted method. $\gamma = 0.98$.

# Chapter 7

# Conclusion

Chapter 3 explains the usage of ROS, how the data for experiments were obtained. It describes an error computation that is used later on during the whole work on this thesis. In that chapter various feature detector configuration are tested. For the evaluation, several metrics were considered. Among the absolute number of inliers, the relative number of inliers, processing time or transformation error. From those tests the FAST feature detector with `threshold` of 20 and `nonmaxSuppression` set on true was chosen and used later on.

The effect of image size was examined for resolutions $1280 \times 960$ and $640 \times 480$. The lower resolution images were processed 4 times faster and resulted in successful transformation more often. Therefore this resolution was chosen. At this point, more resolutions could be tested. Maybe yet lower resolution would give even better results.

In Chapter 4 a principle of a change detection using keypoint descriptors was introduced and explained. Several methods of weighting were discussed: the weighting using distance vector normalization, the weighting using past distance value, and weighting using a global reference value; In Chapter 6 are provided results of the evaluation. The method of weighting using a global reference value was most successful. To get the best out of all of the methods, tuning of parameters should help. Aside of localization using matched keypoints, transformed keypoints localization method was introduced in Chapter 4. The results of evaluations in Chapter 6 shows that this method is suboptimal.

According to the definition of the life-long machine learning (LML) [39], the characteristics of LML are continuous learning, knowledge accumulation and its maintenance, and the ability to use the past knowledge to help future learning. All of the implemented methods (with weighting) are accumulating and maintaining knowledge by updating weights and show continuous learning by applying those weights. Using past knowledge to help future learning might be found in the weighting using past values method, where the past distance value actively determines how to update the weight.

# Bibliography

[1] Dimitrios Karageorgos. “Human-Aware Autonomous Navigation of a Care Robot in Domestic Environments”. MA thesis. Delft University of Technology, 2017.

[2] Pierre-Luc St-Charles, Guillaume-Alexandre Bilodeau, and Robert Bergevin. “Subsense: A universal change detection method with local adaptive sensitivity”. In: *IEEE Transactions on Image Processing* 24.1 (2015), pp. 359–373.

[3] Sicong Liu et al. “Hierarchical unsupervised change detection in multitemporal hyperspectral images”. In: *IEEE Transactions on Geoscience and Remote Sensing* 53.1 (2015), pp. 244–260.

[4] Ali Can et al. “A feature-based, robust, hierarchical algorithm for registering pairs of images of the curved human retina”. In: *IEEE transactions on pattern analysis and machine intelligence* 24.3 (2002), pp. 347–364.

[5] Nikolaus Correll et al. “Analysis and observations from the first amazon picking challenge”. In: *IEEE Transactions on Automation Science and Engineering* (2016).

[6] *RoboCup@Work - Rules*. 2018. URL: http://www.robocupatwork.org/rules.html.

[7] Richard J Radke et al. “Image change detection algorithms: a systematic survey”. In: *IEEE transactions on image processing* 14.3 (2005), pp. 294–307.

[8] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Thomson Learning, 2008.

[9] Morton J Canty. *Image analysis, classification and change detection in remote sensing: with algorithms for ENVI/IDL and Python*. Crc Press, 2010.

[10] Hafez A Afify. “Evaluation of change detection techniques for monitoring land-cover changes: a case study in new Burg El-Arab area”. In: *Alexandria engineering journal* 50.2 (2011), pp. 187–195.

[11] S Baronti et al. “Principal component analysis for change detection on polarimetric multitemporal SAR data”. In: *Geoscience and Remote Sensing Symposium, 1994. IGARSS’94. Surface and Atmospheric Remote Sensing: Technologies, Data Analysis and Interpretation., International*. Vol. 4. IEEE. 1994, pp. 2152–2154.

[12] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[13] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[14] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[15] Jiri Matas et al. "Robust wide-baseline stereo from maximally stable extremal regions". In: *Image and vision computing* 22.10 (2004), pp. 761–767.

[16] Edward Rosten and Tom Drummond. "Machine learning for high-speed corner detection". In: *European conference on computer vision*. Springer. 2006, pp. 430–443.

[17] Peter Biber, Tom Duckett, et al. "Dynamic maps for long-term operation of mobile service robots". In: *Robotics: science and systems*. 2005, pp. 17–24.

[18] Aparna Taneja, Luca Ballan, and Marc Pollefeys. "Image based detection of geometric changes in urban environments". In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2336–2343.

[19] Akihiko Torii et al. "24/7 place recognition by view synthesis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1808–1817.

[20] Oier Mees, Andreas Eitel, and Wolfram Burgard. "Choosing smartly: Adaptive multimodal fusion for object detection in changing environments". In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE. 2016, pp. 151–156.

[21] Relja Arandjelovic et al. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5297–5307.

[22] Tomas Krajnik et al. "Image features for visual teach-and-repeat navigation in changing environments". In: *Robotics and Autonomous Systems* 88 (2017), pp. 127–141.

[23] Akihiko Torii et al. "Visual Place Recognition with Repetitive Structures". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.11 (2015), pp. 2346–2359.

[24] Cristina Romero-Gonzalez et al. "On robot indoor scene classification based on descriptor quality and efficiency". In: *Expert Systems with Applications* 79 (2017), pp. 181–193.

[25] *ROS*. May 2018. URL: `http://www.ros.org`.

[26] *ROS - gmapping package*. May 2018. URL: `http://wiki.ros.org/gmapping`.

[27] *ROS - pylon camera package*. May 2018. URL: `http://wiki.ros.org/pylon_camera`.

[28] *ROS - rosbag package*. May 2018. URL: `http://wiki.ros.org/rosbag`.

[29] *TeamViewer*. May 2018. URL: `https://www.teamviewer.com/en/`.

[30] *ROS - Turtlebot bringup package*. May 2018. URL: `http://wiki.ros.org/turtlebot_bringup`.

[31] *ROS - Turtlebot teleop package*. May 2018. URL: `http://wiki.ros.org/turtlebot_teleop`.

[32] *ROS - Hokuyo node package*. May 2018. URL: `http://wiki.ros.org/hokuyo_node`.

[33] *ROS - tf package*. May 2018. URL: `http://wiki.ros.org/tf`.

[34] Matej Beranek, Martin Novotny, and Martin Zakovec. "ROS - camera node". Unpublished ROS package.

[35] Tino Kluge. *Cubic Spline interpolation in C++*. C++ library. 2011-2014. URL: http://kluge.in-chemnitz.de/opensource/spline/.

[36] *OpenCV 2*. May 2018. URL: https://opencv.org/opencv-2-4-8.html.

[37] *OpenCV FeatureDetector Class Reference*. May 2018. URL: http://physics.nyu.edu/grierlab/manuals/opencv/classcv_1_1FeatureDetector.html.

[38] *OpenCV DescriptorExtractor Class Reference*. May 2018. URL: http://physics.nyu.edu/grierlab/manuals/opencv/classcv_1_1DescriptorExtractor.html.

[39] Zhiyuan Chen and Bing Liu. *Lifelong machine learning*. Morgan & Claypool, 2016.

Appendix A

# The contents of the enclosed CD

The CD includes:

- Text of this document
- Developed ROS package
- `MATLAB` scripts used for visualization and evaluation