Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science and Engineering

# DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Erik Derner**

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: **Visual localization and place recognition**

Guidelines:

An outdoor robot is equipped with various sensors including an omnidirectional camera. The robot should be able to self-localize itself in a highly unstructured and GPS denied environment. The goal of the work is to develop, implement and integrate a method for robot localization based on visual recognition of previously seen places. The previous observations may not be taken in the same weather and lighting conditions or exactly from the same positions and also may be captured by different cameras. The developed system should take these requirements into account and should enable reliable vision-based self-localization in variable conditions.

Bibliography/Sources:

[Murillo-ToR2013] Murillo, A. C.; Singh, G.; Kosecká, J. & Guerrero, J.
Localization in Urban Environments Using a Panoramic Gist Descriptor
IEEE Transactions on Robotics, 2013, 29, 146-160

[Churchill-IJRR2013] Churchill, W. & Newman, P.
Experience-based Navigation for Long-term Localisation
The International Journal of Robotics Research (IJRR), 2013, 32, 1645-1661

[Caffe] http://caffe.berkeleyvision.org

Diploma Thesis Supervisor: PhD Daniel, Skočaj

Valid until the end of the summer semester of academic year 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, March 26, 2015

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

# Visual Localization and Place Recognition

Erik Derner

May 2015

**Thesis supervisor:** doc. dr. Danijel Skočaj

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 11. 5. 2015

## Acknowledgment

First of all, I would like to thank my supervisor doc. Danijel Skočaj for his great guidance, help and comments. I would also like to thank doc. Tomáš Svoboda, who helped me to arrange the stay at the University of Ljubljana, where I worked on this thesis. I am grateful to my colleagues from the ViCoS laboratory for valuable advice and help with the preparation of the robot, in particular to Anže Rezelj and Jaka Cikač. Finally, I would like to thank my parents for their endless support during my studies.

# Abstract

This thesis presents a method for visual localization of a mobile robot equipped with an omnidirectional camera. The proposed algorithm uses a pre-trained convolutional neural network to compute the descriptors of panoramic images. The position of a query image is estimated using these descriptors, following the assumption that images with similar descriptors were taken at physically close positions. The basic nearest neighbor approach for finding images with the most similar descriptors was tested together with some improvements, such as the weighted $k$-nearest neighbors algorithm and the $k$-means clustering. These extensions showed to improve significantly the precision of the position estimation and decrease the time necessary to process a query image. The algorithm was implemented in Matlab and tested on data from a mobile robot in various experiments. The results of the experiments indicate that the approach based on convolutional neural networks is suitable for the visual localization task and that the proposed algorithm can handle various changes of the environment and capturing conditions, such as different lighting, images taken from slightly different places and with an arbitrary orientation of the robot.



# Abstrakt

Tato práce se zabývá metodou vizuální lokalizace mobilního robotu vybaveného všesměrovou kamerou. Navržený algoritmus používá předučenou konvoluční neuronovou síť pro výpočet deskriptorů panoramatických snímků. Pozice snímku se odhaduje pomocí těchto deskriptorů na základě předpokladu, že snímky s podobnými deskriptory byly pořízeny na blízkých místech. Základní metoda nejbližšího souseda pro vyhledání snímků s nejvíce podobným deskriptorem byla testována společně s vylepšeními jako metoda vážených $k$ nejbližších sousedů a shlukování $k$-středů. Tato rozšíření významně zlepšují přesnost odhadu pozice a snižují čas potřebný pro zpracování dotazovaného snímku. Algoritmus byl implementován v jazyce Matlab a testován na datech z mobilního robotu v různých experimentech. Výsledky experimentů ukazují, že metoda založená na konvolučních neuronových sítích je vhodná pro úlohu vizuální lokalizace a navržený algoritmus se dokáže vypořádat s různými změnami prostředí a podmínek pro snímání, jako například různé osvětlení, snímky pořízené z mírně odlišných míst a s libovolnou orientací robotu.

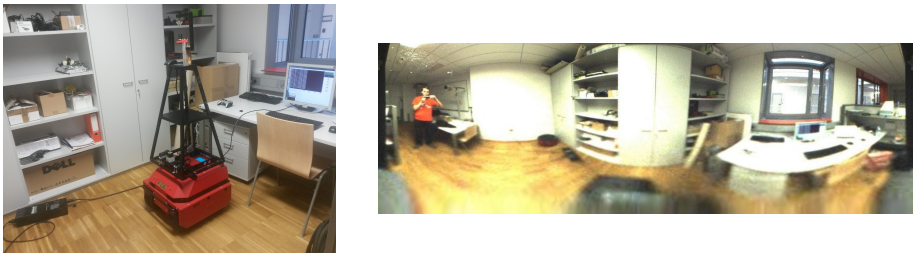
# Contents

# List of Figures

# List of Tables

# 1 Introduction

Visual localization and place recognition are important tasks to be carried out by mobile robots. It is advantageous in numerous scenarios that the robot knows its position. In this work, we will present a localization approach for mobile robots equipped with an omnidirectional camera, see Figure 1. The method is based on the increasingly popular deep learning techinques using convolutional neural networks.



a b

Figure 1: A mobile robot equipped with an omnidirectional camera (a) and a panoramic image of the scene around the robot from the view of its camera (b), captured at the same moment.

## 1.1 Motivation

Localization of the robot might be performed using various sensors, such as a laser rangefinder, stereo vision or by a GPS device. However, these methods have some considerable drawbacks. GPS signal is not available everywhere – there is almost no chance to use GPS indoors. Localization based on a laser rangefinder suffers from difficulties with scenes containing transparent materials such as glass. Vision-based approaches do not have any of these disadvantages, so it seems advantageous to use images as a source of data for localization. Images using standard field of view might not encode enough information about the environment around the robot though. We will focus therefore on localization based on omnidirectional images in this work.

## 1.2 Task formulation

The main objective of the work is to develop a method for robot localization based on visual recognition of previously seen places. The foreseen scenario is that a mobile robot records omnidirectional images together with the position in order to build a database of images with known positions. Using this database, the robot should be able to localize itself given a query omnidirectional image. The motivation for this approach is that similar images are likely to be captured at physically close places.

The proposed algorithm should take into account that the query images might not be captured under the same conditions as the database images. The robustness of the method should be sufficient to handle changes of the environment such as different lighting conditions or occlusion.

## 1.3 Localization methods

Localization of mobile robots can be carried out in various ways. The suitable methods vary depending on the indoor or outdoor usage.

A typical example of an indoor localization method is using a laser rangefinder. The laser rangefinder determines the distance of objects in its field of view based on the reflection of the emitted laser rays. The information about the distance of objects in scans from different positions is used to build a map of the environment.

Another common method is stereo vision, which consists in localization based on a 3D model of the world constructed using a calibrated pair of cameras. It can be used both indoors and outdoors. One of the widely known methods for outdoor localization is using global navigation systems such as GPS or Glonass, which calculate the position based on the data broadcast by satellites in the Earth's orbit.

We will further focus on vision-based approaches for localization. The aforementioned localization methods serve as a source of the position data when building the database of images.

## 1.4 Related work

The problem of visual localization in variable environment is addressed in several works. In the paper by Churchill and Newman on long-term navigation in changing environments [1], the concept of *experiences* is used to capture variable appearance of the same place and thereby make the localization robust. An experience captures a visual mode, i.e., a particular appearance of the same scene. As the robot traverses the workspace, it gathers distinct visual experiences. When the robot operates in a previously visited area, it simultaneously attempts to localize itself in the previous experiences and determine its position using a vision-based pose estimation system. If the localization fails in large enough number of previous experiences, the current live stream is added as a new experience. That way, the coverage of the possible distinct appearances of the scene is maximized.

The paper by Milford and Wyeth [2] presents a similar approach using an experience map. The term experience is however used in a different sense than in [1]. The experience map is a topological graph and its nodes contain an image, a position estimate and a transition to other experiences. The distances between global node positions and the relative transitions are continuously attempted to be minimized.

Krajník et al. present in the paper [3] a novel method for localization of mobile robots in dynamic environments. The algorithm builds a spatio-temporal world model that is able to predict changes of the environment in time by exploiting the repetitiveness

of daily activities in populated environments. Local states of the environment are modeled as a probability function of time. It is shown that the model learned during the period of one week is still applicable even after three months and the precision of localization is significantly better than using static representations.

The paper on localization based on a global GIST descriptor by Murillo et al. [4] also uses omnidirectional images for visual localization, as in this work. The paper describes the way of representation of omnidirectional images using a GIST descriptor and relating them by different GIST similarity measures. To briefly introduce the GIST descriptor, it is a global descriptor combining responses of several filters applied to the input image. In the default configuration, it is a 960-dimensional vector, with 320 values per color channel. It encodes the distribution of orientations and scales in the image. The advantages of the descriptor are speed (it is fast to compute) and compactness. An omnidirectional image is transformed into 4 views with a step of 90° and the 4-tuple of GIST descriptors of those views is used in the algorithm. It is therefore mainly suitable for urban environments, which feature perpendicular and parallel streets. The results of the method using the GIST descriptors are shown to be comparable with approaches using local features.

Chen et al. present in the paper [5] for the first time a method for place recognition based on pre-trained convolutional neural networks (CNNs), which are also used in this work. Place recognition can be viewed as an image retrieval task, i.e., determining a match between the current scene and a previously visited location. Responses of the CNN layers are used as descriptors. The match for a test image is the training image with the smallest Euclidean distance of its descriptor to the descriptor of the test image. This match is then verified using spatio-temporal filtering. They show that the CNN features significantly outperform SIFT (scale-invariant feature transform) descriptors.

In the project preceding this thesis [6], we compared the performance of an implementation [7] of the approach using GIST descriptor [4] with a method using the CNN descriptor that we proposed. The proposed method consists in simply minimizing the Euclidean distance of the CNN-based descriptor of the query image and the database images using a linear search in the database to find the nearest neighbor. Both algorithms were run on the same dataset and the CNN-based approach proved to outperform the GIST-based algorithm. This thesis builds on the results of the project.

## 1.5 Proposed method

To conclude this chapter, we will present an overview of the approach used in this work.

As the robot moves through the environment, it records omnidirectional images together with their positions and stores them in a *database*. The position is determined using a laser rangefinder.

An image with an unknown position will be denoted in this work as a *query* image. For a given query image, the robot should be able to localize itself using the database of previously seen images.

Based on the promising results of our previous work using convolutional neural networks [6], we decided to continue in that direction also in the thesis. A descriptor calculated using a convolutional neural network is saved for each database image and likewise it is calculated for each query image. The position of a query image is estimated by making use of the nearest neighbor method, weighted $k$-nearest neighbors algorithm and $k$-means clustering. In Chapter 2, it will be described, how the descriptor is computed and used for the localization of the robot.

## 1.6 Thesis structure

The text of the thesis is structured as follows. Chapter 2 contains a brief introduction to convolutional neural networks, description of the algorithm and its implementation. Chapter 3 describes the robot used to test the algorithm and the process of recording the images and building the datasets for experiments. Description of the performed experiments can be found in Chapter 4. The overall review of the achieved results is summarized in Chapter 5.

# 2 Our approach

In this chapter, it will be explained, how the position of a query image is estimated given a database of images with known positions. First of all, the convolutional neural networks will be introduced and the used framework for deep learning will be described. After that, the algorithm used in this work will be described, starting by introducing the concept of observations and then explaining the computation of the descriptor. Finally, the implementation of the algorithm will be shortly described.

## 2.1 Convolutional neural networks

In this work, convolutional neural networks are used to localize the robot and therefore they will be presented in this section. Let us begin with briefly introducing simple artificial neural networks (ANNs). ANNs are statistical learning algorithms inspired by biological neural networks. They are used to estimate or approximate functions dependent on a generally large number of inputs. ANN can be viewed as a system of interconnected neurons. Simply said, each neuron computes a value from its inputs and sends it to its output. The adaptivity of the neural networks make them an useful tool in machine learning.

Convolutional neural networks (CNNs) are variations of multi-layer perceptrons. They are designed to use minimal amount of preprocessing. In the context of image processing, it means that the images are sent to the network almost directly, with none or minimal preprocessing such as histogram equalization or applying various filters [8].

CNNs consist of several layers. These layers can be of three types:

- *Convolutional layer* – consists of a rectangular grid of neurons. The previous layer has to be also a rectangular grid of neurons, since the neurons in this layer take weighted outputs of a rectangular area in the previous layer. Weights for the neurons in a particular rectangular area are the same. This layer represents image convolution of the previous layer, the weights for the rectangular areas specify the convolution filter.
- *Max-pooling layer* – may be present after a convolutional layer. Neurons in this layer take small rectangular blocks of the preceding convolutional layer and find the maximum among the outputs of the neurons in the block. The maximum is sent to the output of the neuron.
- *Fully connected layer* – presents a higher-level reasoning and is usually present after several convolutional and max-pooling layers. It can follow after any kind of a layer – convolutional, max-pooling or another fully connected. Its name is derived from the fact that it connects all outputs of the previous layer to all neurons present in the current layer. As the fully connected layers do not form anymore a rectangular grid, only other full connected layers may follow them, but not the other types of layers [8].

A convolutional neural network consists of a combination of these layers connected together. An example of a CNN can be seen in Figure 2.

The *Caffe* deep learning framework was used in this work. Caffe, which stands for Convolutional Architecture for Fast Feature Embedding, is a C++/CUDA architecture for deep learning with Matlab and Python interfaces. Although the toolbox is designed to work on any domain, such as speech, haptics, neuroscience etc., the main focus of the authors is on the field of visual recognition.

From the variety of software packages within this project that are freely available under the BSD license, we use a framework containing a Matlab bridge that is designed for retrieval of similar images from a large image database [9].

## 2.2 Observation

The concept of *observations* is central in this work. Let us start with a definition of an observation. An observation is a container for the captured image and all important parameters of it, in the first place its real-world position and descriptor. The position is a pair $(x, y)$ of values in meters, specifying the location of the observation relative to a given origin of a common coordinate system. An observation can also hold some additional parameters such as the rotation angle. All database images are handled in the algorithm as observations.

Also the query images are handled as observations. However, the position naturally does not need to be set in that case. If it is set, it serves as the ground truth for evaluation of the precision of the position estimation algorithm in the experiments.

## 2.3 Descriptor

A trained CNN model *ImageNet* is used to calculate the descriptor in this work. Let us start by a brief introduction of this model. ImageNet is a convolutional neural network learned on 1.2 million images, assigned to 1000 classes. The network, featuring 60 million parameters and 650000 neurons, consists of 5 convolutional layers and 3 fully connected layers with a final 1000-way softmax, which produces a distribution over the 1000 classes. Some of the convolutional layers are followed by max-pooling layers [10]. The architecture of the ImageNet CNN is depicted in Figure 2.

The response of the last hidden (fully connected) layer of the convolutional neural network is used in this work as a descriptor of an image and it will be further referred to shortly as the *CNN descriptor*. The descriptor is therefore a 4096-dimensional vector.

The image descriptor is calculated with the use of the Caffe toolbox [9]. The pre-processing consists in resizing the image to $224 \times 224$ px and subtracting the mean calculated over the whole training set for each pixel and color channel. It is worth mentioning that this is the only preprocessing of the image, i.e., no filters are applied in advance to passing the image to the CNN. The CNN itself takes care of the robustness to changes of the scene captured in the image, such as variable lighting conditions and occlusion.

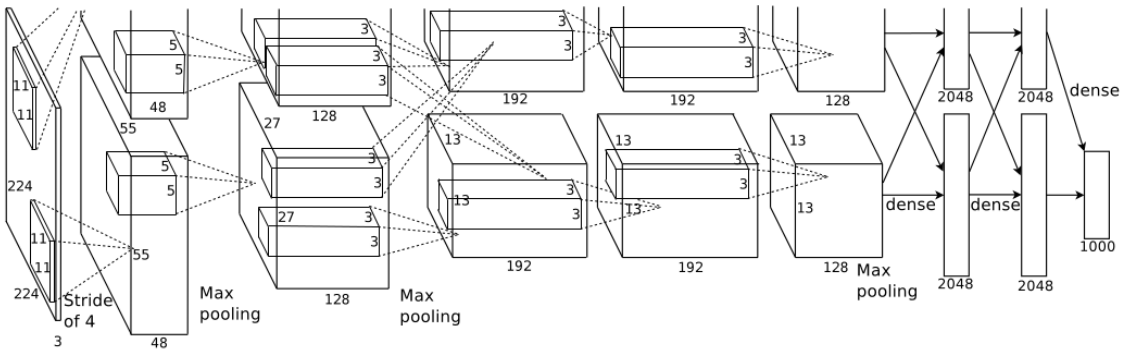



Figure 2: Illustration of the ImageNet convolutional neural network architecture. The two 'rows' represent the division of work between two GPUs [10].

## 2.4 Relations between observations

In order to quantify, how much two observations are related to each other, we need to specify the *dissimilarity measure*. We define the dissimilarity measure of two observations $\mathtt{o}_1$, $\mathtt{o}_2$ as the square of the Euclidean distance between their 4096-dimensional descriptors:

$$d_d^2(\mathtt{o}_1, \mathtt{o}_2) = \sum_{i=1}^{4096} (\mathtt{o}_1.\mathtt{desc}_i - \mathtt{o}_2.\mathtt{desc}_i)^2 \ , \tag{1}$$

where the lower index $d$ in $d_d^2$ stands for the distance of *descriptors* and $\mathtt{o}_\mathtt{j}.\mathtt{desc}$ represents the descriptor of the observation $\mathtt{o}_\mathtt{j}$. The measure is called dissimilarity, because the larger is its value, the less similar are the observations.

On the contrary, the *physical* (real-world) distance between two observations $\mathtt{o}_1$, $\mathtt{o}_2$ is denoted $d_p$ and it is calculated as the Euclidean distance of their positions:

$$d_p(\mathtt{o}_1, \mathtt{o}_2) = \sqrt{(\mathtt{o}_1.\mathtt{position.x} - \mathtt{o}_2.\mathtt{position.x})^2 + (\mathtt{o}_1.\mathtt{position.y} - \mathtt{o}_2.\mathtt{position.y})^2} \ . \tag{2}$$

## 2.5 Position estimation

The position of a query image given a database of images can be estimated in various ways. In all cases, the dissimilarity measure defined in the previous section is utilized. In the following text, the position estimation methods used in this work will be described.

### 2.5.1 Nearest neighbor

The simplest approach of position estimation is the nearest neighbor method. The database of $N$ images is linearly searched and for each database observation $\mathtt{o}_{\mathtt{d}i}$, the dissimilarity measure $d_d^2$ between $\mathtt{o}_{\mathtt{d}i}$ and the query observation $\mathtt{o}_{\mathtt{q}}$ is calculated. The position of the database observation with the minimal value of the dissimilarity measure $\mathtt{o}_{\mathtt{d}i_{\min}}$ is picked as the position estimate $\mathtt{position}_{\mathtt{est}}$ of the query observation:

$$i_{\min} = \arg\min_{i=1,\ldots,N} d_d^2\left(\mathtt{o}_{\mathtt{q}}, \mathtt{o}_{\mathtt{d}i}\right) \ , \tag{3}$$

$$\mathtt{position}_{\mathtt{est}}\left(\mathtt{o}_{\mathtt{q}}\right) = \mathtt{o}_{\mathtt{d}i_{\min}}.\mathtt{position} \ . \tag{4}$$

### 2.5.2 Weighted $k$-NN

In order to improve the accuracy of the position estimation, it might be advantageous to take into account more than only one nearest neighbor. It is necessary to first define the term *closest observation*, which will be used in the following text. A closest observation is such a database observation that has the lowest value of the dissimilarity measure between the query observation and that database observation, among all database observations.

The closest observations may be considered with equal weights, i.e., the estimated position can be calculated simply as a centroid of the positions of the $k$ closest observations. However, it proved to be beneficial to consider the observations with decreasing weights, i.e., the first closest observation has the largest influence and the influence of the following closest observations gradually declines.

At first, we sort the database observations by the dissimilarity measure between the query observation and the database observation in ascending order, so that the set of the first $k$ closest observations will be denoted

$$S_k\left(\mathtt{o}_{\mathtt{q}}\right) = \left\{\mathtt{o}_{\mathtt{d}i_{\min 1}}, \mathtt{o}_{\mathtt{d}i_{\min 2}}, \ldots, \mathtt{o}_{\mathtt{d}i_{\min k}}\right\} \ , \tag{5}$$

where

$$d_d^2\left(\mathtt{o}_{\mathtt{q}}, \mathtt{o}_{\mathtt{d}i_{\min 1}}\right) < d_d^2\left(\mathtt{o}_{\mathtt{q}}, \mathtt{o}_{\mathtt{d}i_{\min 2}}\right) < \cdots < d_d^2\left(\mathtt{o}_{\mathtt{q}}, \mathtt{o}_{\mathtt{d}i_{\min k}}\right) \ . \tag{6}$$

The estimated position of the query observation $\mathtt{o}_{\mathtt{q}}$ is then calculated using this method as follows:

$$\mathtt{position}_{\mathtt{est}}\left(\mathtt{o}_{\mathtt{q}}\right) = \sum_{j=1}^{k} d_d^2\left(\mathtt{o}_{\mathtt{q}}, \mathtt{o}_{\mathtt{d}i_{\min(k-j+1)}}\right) \cdot \mathtt{o}_{\mathtt{d}i_{\min j}}.\mathtt{position} \ , \tag{7}$$

where $d_d^2(\ldots)$ plays the role of a weight. The weight of the position of the first closest observation is the dissimilarity measure between the query observation and the $k$-th closest observation, the weight of the position of the second closest observation is the

dissimilarity measure between the query observation and the $(k-1)$-th closest observation and so on, hence the weights are decreasing with increasing $j$.

The time complexity of the described algorithm is $\mathcal{O}(n \log(n))$. However, as we need only first $k$ closest observations, we can find the closest observation using Equation 3, cross it out and search for the remaining $(k-1)$ closest observations in the same way. Using that approach, the time complexity stays linear as in the simple nearest neighbor method.

### 2.5.3 Clustering

The linear time complexity of the exhaustive search makes the position estimation algorithm too slow, if the database of observations becomes large. Therefore, it is desirable to use a more sophisticated search method to achieve a speed-up.

One of the methods that might come up for consideration is a $k$-d tree. This space partitioning data structure used for organizing points in a $k$-dimensional space is however not suitable for high-dimensional spaces. The general rule for effective usage of $k$-d trees is $N \gg 2^k$, where $N$ is the number of data points and $k$ is the dimensionality [11]. In our case, where $k = 4096$ and $N \approx 10^6$ for the largest foreseen observation databases, the aforementioned rule does not hold substantially, by many orders of magnitude, which makes the use of a $k$-d tree meaningless.

Data clustering seems as an approach suitable for our case. One of the options is approximate nearest neighbor search, in particular, locality-sensitive hashing. This method creates bins of similar data points, where the bin, into which a data point belongs, is determined by calculating a hash function. This hash function has, unlike conventional hash functions, a high probability of collision for similar items [12].

We will stick with data clustering, however using another method, $k$-means. In this algorithm, the database of observations is partitioned into $k$ clusters of similar observations, each of the partitions being represented by its mean [13]. The algorithm proceeds in our case in the following steps:

1. Initialize cluster means of all clusters $\mathtt{c}_i$ by the descriptors of observations randomly chosen among all database observations:

$$\forall i = 1, \ldots, k : \mathtt{c}_i.\mathtt{mean} = \mathtt{o}_{\mathtt{rand}(1,N)}.\mathtt{desc} \ , \tag{8}$$

   where $\mathtt{rand}(1, N)$ returns a random number between 1 and $N$. The number $N$ is the database size, i.e., number of database observations.

2. Assign all observations to the closest clusters. The dissimilarity measure $d_d^2$ is suitable as a value to be minimized when searching for the closest cluster for each observation $\mathtt{o_d}$:

$$i_{\min} = \underset{i=1,\ldots,k}{\arg\min}\, d_d^2\left(\mathtt{o_d}, \mathtt{c}_i.\mathtt{mean}\right) \ , \tag{9}$$

$$\mathtt{cluster}\left(\mathtt{o_d}\right) = \mathtt{c}_{i_{\min}} \ . \tag{10}$$

   To be formally correct, please note that we are overloading the function $d_d^2$, as the second argument is in this case not an observation, but directly the descriptor.

3. Recalculate the cluster means for each cluster $\mathtt{c}_i$ as a mean of descriptors of observations currently assigned to the cluster. Formally, for each cluster:

$$\mathtt{c}_i.\mathtt{mean} = \frac{1}{\mathtt{size}\,(\mathtt{c}_i)} \sum_{\mathtt{o}_j \in C_i} \mathtt{o}_j.\mathtt{desc}\ , \tag{11}$$

where

$$C_i = \{\mathtt{o}_j : \mathtt{cluster}\,(\mathtt{o}_j) = \mathtt{c}_i\}\ . \tag{12}$$

If any of the cluster means has changed, go to step 2, otherwise go to step 4.

4. Done.

The query observations $\mathtt{o_q}$ are processed within the frame of the partitioned database of observations in the following way. At first, the closest mean is found among all cluster means:

$$i_{\min} = \arg\min_{i=1,\ldots,k} d_d^2\,(\mathtt{o_q}, \mathtt{c}_i.\mathtt{mean})\ . \tag{13}$$

After that, the position of the query observation is estimated by the means of Equation 3 and 4 in the case of the simple nearest neighbor, or using Equation 5, 6 and 7 for the weighted $k$-NN algorithm. In both cases, the database of observations reduces to $C_{i_{\min}}$, as defined in Equation 12.

As an extension, it is possible to use not only the closest cluster, but a union of $c$ closest clusters as the database of observations to be considered when searching for the $k$ nearest neighbors. The database of observations to be considered is then $\bigcup_m C_m$, where the first value of $m = i_{\min}$ is found using Equation 13, then the found nearest cluster is crossed out and the search for the nearest cluster mean using Equation 13 is repeated until $c$ nearest cluster means are found. The motivation for this extension is that the $k$ closest observations found by an exhaustive search might have been partitioned into different clusters. The choice of the parameter $c$ therefore represents a trade-off between speed and accuracy.

## 2.6 Implementation

The described algorithm was implemented and tested in Matlab 7.13 (R2011b). We will briefly introduce the main concepts and functions of the implementation of the algorithm in this section. We will focus only on the codes relevant to the algorithm described in this chapter, as more programs were created to capture the data and prepare the datasets. These codes are described in Appendix A and B.

We will start by introducing the implementation of the concept of observations. It is implemented as a custom Matlab class `Observation` with the following fields:

- `im_pathname` – full path to the image,
- `dataset` – name of the dataset to which the image belongs,
- `position` – location where the image was captured; structure with fields `x` and `y`; values are in meters,

- `desc` – response of the last hidden CNN layer, used as an image descriptor,
- `rotation` – artificial rotation angle in degrees, relative to the original captured image (see Section 3.3.3),
- `timestamp` – identifier of the moment when the image was captured.

The sets of database and query observations have a form of arrays of `Observations`.

The core function is `best_matches()`, which sorts the database observations in ascending order by the dissimilarity measure, calculated with respect to a given query observation. There is also a variant of this function, `best_matches_clustering()`, which is used for the same purpose when using clustering. These functions are used in the function `find_most_similar_obs()`, which runs the computation of the sorted list of closest observations for all query observations present in the query set.

To use clustering, it is necessary to first partition the set of database observations using the function `partition_observations()`.

The functions `nearest_neighbor()` and `weighted_knn()` compute the estimation of the position of all query observations using the respective method.

Numerous helper functions are used for handling datasets, e.g. `filter_rotations()` and `filter_subset()`. All functions are described in the comments in their source codes.

There are also scripts starting with `experiment*`, which are used to perform various experiments described in this work. These scripts are used to plot figures to evaluate the performance of the algorithm. Moreover, they include an interactive tool for visualization of the position estimation, see Figure 3. A live example can be run from the enclosed DVD.

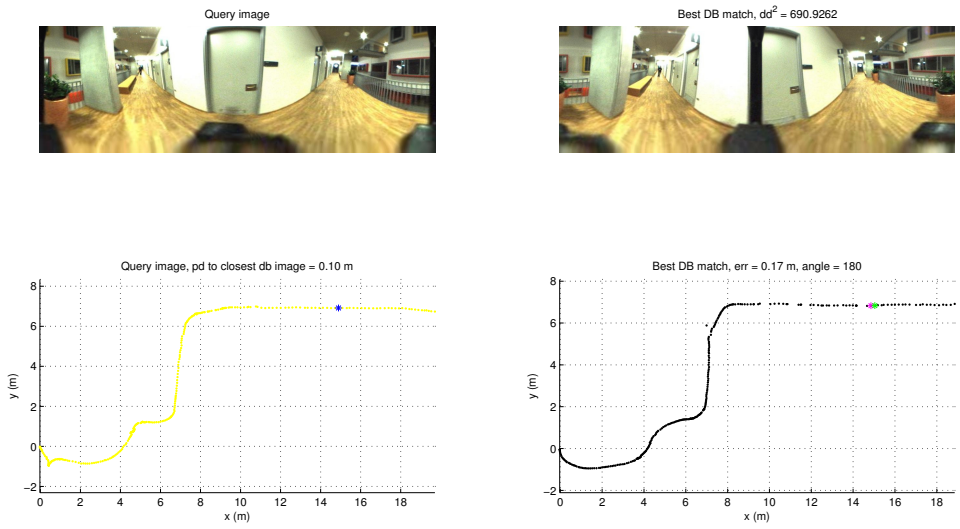



Figure 3: Example of the visualization of an experiment in Matlab. The left side represenets the query, the right side the database. The blue star is the position of the query observation. The magenta star represents the position of the database observation physically closest to the query observation, the green star shows the position estimated by the algorithm.

# 3 Data acquisition

This chapter describes, how the data are recorded on the robot and how they are further processed to form datasets for experiments. At first, we will describe the robot, its components and their parameters. In the second part, we will explain how the robot collects data from its sensors when moving around the environment. The final part of this chapter is focused on the process of building datasets for experiments.

## 3.1 Mobile robot

The mobile robot used for the experiments (see Figure 4) is built on the iRobot ATRV mini base. The operation of the robot is handled on the low level by a microcontroller running rFLEX. The user interface of the microcontroller consists of a monochromatic display and a single rotary/push button. The microcontroller serves as a bridge between the low level peripherals, such as actuators, and the higher level of operation, which is controlled by a PC running Linux Ubuntu with ROS [14]. The parameters of the PC are described in Table 1.

The original robot was adjsuted by custom modifications, which will be now shortly described. First of all, there is a large metal construction on the robot base, which holds various sensors. On the top of the construction, there is a camera pointing upwards to a paraboloidal reflector, see Figure 5. The camera captures 360° circular images, which can be easily transformed to standard wide panoramic images. The specifications of the camera are summarized in Table 2.

Another sensor, which is used in this work, is a laser rangefinder, see Table 3 for specifications. It is captured in Figure 6. It emits laser rays in a single plane and measures the distance, from which they are reflected. The laser rangefinder is used to build a map and localize the robot within it. That way, the position data are acquired for database images and as a ground truth also for query images.

The movement of the robot is controlled manually by the Logitech Cordless Rumblepad 2. The control device allows to operate independently the left-side and right-side wheels, which enables also movements such as on-spot turning.

## 3.2 Capturing the data

The PC in the robot runs Ubuntu 12.04 with ROS hydro [14]. Most of the data capturing tasks are performed within the ROS framework. The step-by-step procedure of capturing the data is described in detail in Appendix A.

To briefly summarize the capturing process, several commands have to be run within the ROS system to start the Rumblepad controller, laser rangefinder and the process of building the map. After that, the position can be saved to a file using another ROS component. One more program, this time outside of the ROS framework, is responsible


Figure 4: ATRV mini robot used for the experiments.

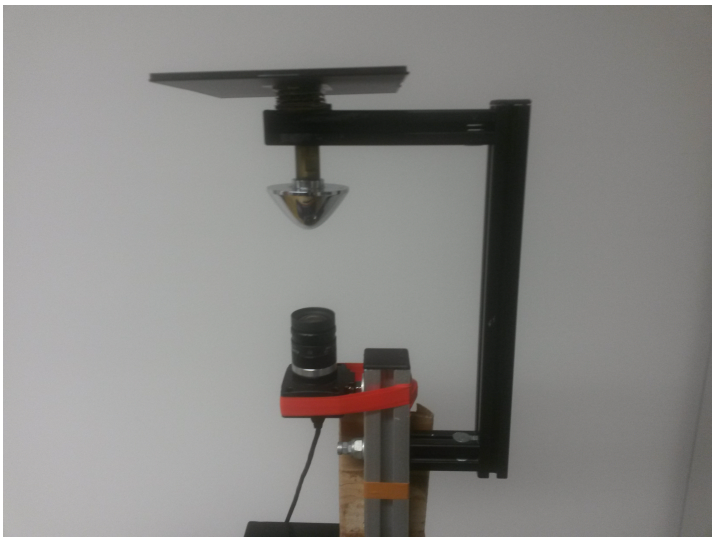


Figure 5: Camera installed on the ATRV mini robot, used for capturing omnidirectional images.


Figure 6: Laser scanner installed on the ATRV mini robot.

for saving the images from the camera. The frequency of saving both the images and the position can be set individually. The images and the corresponding position data are recorded independently and they are then matched together using timestamps.

## 3.3 Building datasets

The recordings of images and positions from a single continuous robot run form a dataset. In the following text, it will be described, how the images and positions are converted into observations, as defined in Section 2.2. A step-by-step procedure for building datasets including all the commands can be found in Appendix B.

| | |
|---|---|
| **Motherboard** | ASRock Z77E-ITX |
| **CPU** | Intel Core i7 3770T, quad-core, 2.5 GHz, 8 MB L3 cache |
| **GPU** | GeForce GTX 650, 1 GB DDR5 |
| **RAM** | 8 GB DDR3 |
| **Disk drive** | Samsung 840 series 120 GB SSD |
| **OS** | Linux Ubuntu 12.04 |

Table 1: Specifications of the PC in the robot [15].

| | |
|---|---|
| **Manufacturer** | Point Grey Research, Inc. |
| **Model** | Chameleon CMLN-13S2C-CS |
| **Interface** | USB 2.0 |
| **Resolution** | 1296 × 964 px |
| **Chroma** | Color |
| **Sensor** | Sony ICX445, CCD, 1/3” |

Table 2: Specifications of the camera used for capturing omnidirectional images [16].

| | |
|---|---|
| **Manufacturer** | Hokuyo Automatic Co., Ltd. |
| **Model** | URG-04LX |
| **Measuring distance** | 60 to 4095 mm |
| **Accuracy** | 60 to 1000 mm: ±10 mm,<br>1000 to 4095 mm: 1 % |
| **Angular range** | 240° |
| **Angular resolution** | 0.36° |
| **Scanning time** | 100 ms |

Table 3: Specifications of the laser rangefinder used for building a map [17].

### 3.3.1 Converting 360° images to wide panoramic images

As briefly described in Section 3.1, the camera takes 360° circular images, which have to be converted to wide panoramic images. An example of a 360° image and its dewarped version is shown in Figure 7. The advantage of wide panoramic images is that the rotation of the robot can be simulated by simply shifting the pixels in the horizontal direction. The process of dewarping the 360° images using bilinear interpolation consists of several steps:

1. The following parameters serve as an input for the dewarping function:

   - $x_c$, $y_c$ – coordinates of the center,
   - $r_i$ – radius of the inner circle, which is between the panoramic image area and the reflection of the paraboloidal reflector in the camera lens,
   - $r_o$ – radius of the outer circle, which is between the panoramic image area and the black cover.

   All values are in pixels, the origin of the coordinate system is in the top left corner of the image, the $x$-axis points to the right and the $y$-axis points down. Figure 8 shows the visualization of the aforementioned parameters.

2. Define the difference of radii:

$$r_d = r_o - r_i \ . \tag{14}$$

3. Calculate the dimensions of the resulting wide panoramic image ($h$ is height, $w$ is width, both values are in pixels):

$$h = r_d \ , \ w = \texttt{round}\left(\pi r_d\right) \ . \tag{15}$$

4. For each pixel $(x_o, y_o)$ of the output wide panoramic image, calculate the color $\texttt{c}$ in the following way:

$$r = \frac{(h - y + 1)^2}{h} + r_i \ , \tag{16}$$

$$\theta \ = 2\pi \frac{x}{w} \ , \tag{17}$$

$$x_p = x_c + r\sin(\theta) \ , \ y_p = y_c + r\cos(\theta) \ , \tag{18}$$

$$x_l = \lfloor x_p \rfloor \ , \ y_l = \lfloor y_p \rfloor \ , \ x_h = x_l + 1 \ , \ y_h = y_l + 1 \ , \tag{19}$$

$$\texttt{c}_1 = (y_h - y_p) \cdot \texttt{color}(x_l, y_l) + (y_p - y_l) \cdot \texttt{color}(x_l, y_h) \ , \tag{20}$$

$$\texttt{c}_2 = (y_h - y_p) \cdot \texttt{color}(x_h, y_l) + (y_p - y_l) \cdot \texttt{color}(x_h, y_h) \ , \tag{21}$$

$$\texttt{c} = (x_h - x_p)\texttt{c}_1 + (x_p - x_l)\texttt{c}_2 \ , \tag{22}$$

The function $\texttt{color}(x, y)$ returns the color of the pixel at the position $(x, y)$ in the original 360° image. Equations 16–18 perform the conversion between the polar and Carthesian coordinates, with an adaptation for equalization of the parabolic curvature of the reflector. Equations 19–22 represent the bilinear interpolation.

### 3.3.2 Computing descriptors

The CNN descriptor, as defined in Section 2.3, is calculated for all dewarped images using the Caffe toolbox [9]. Moreover, each panoramic image is rotated in 36 steps, i.e., with a step size 10°, and for all of these artificial rotations, the descriptors are calculated. No preprocessing of the images such as histogram equalization or applying any filters is done.

### 3.3.3 Generating observations

At this point, all resources are prepared, so that it is possible to generate observations. For each image, the matching (time-nearest) position is found. As stated in the previous section, each original image yields 36 observations, one for each of the rotation steps (10°). The artificial rotation angle is also stored in the observation, together with the image (link to the image), descriptor and position. The set of generated observations is saved and represents the dataset.

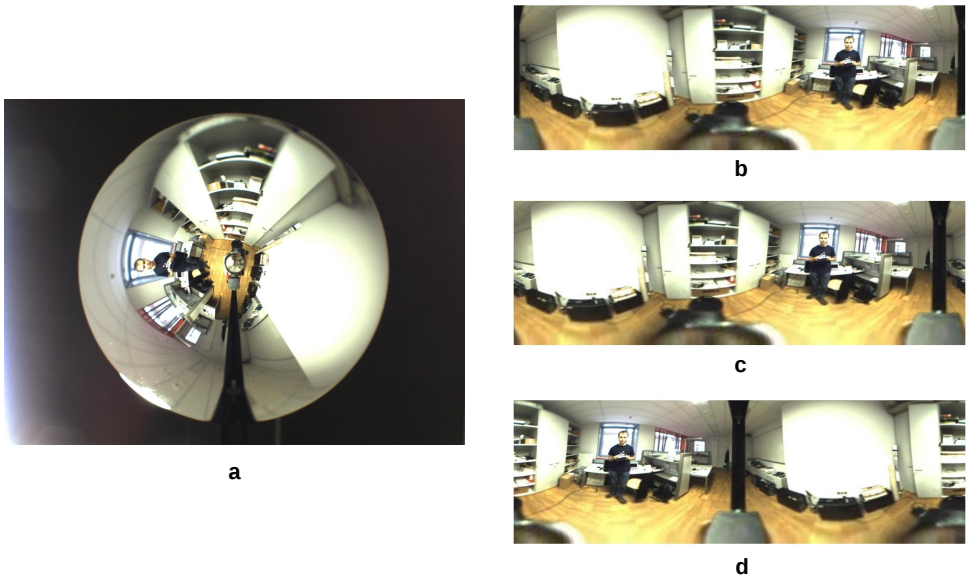



Figure 7: Demonstration of dewarping of 360° images. An example of a circular 360° image, as captured by the camera on the robot, is shown in (a). This image is dewarped to a wide panoramic image in the default position with the arm holding the paraboloidal reflector at the sides of the image (b). Using the panoramic images, rotation can be easily simulated by shifting, as can be seen in (c) – rotation by 20° and (d) – rotation by 180°.

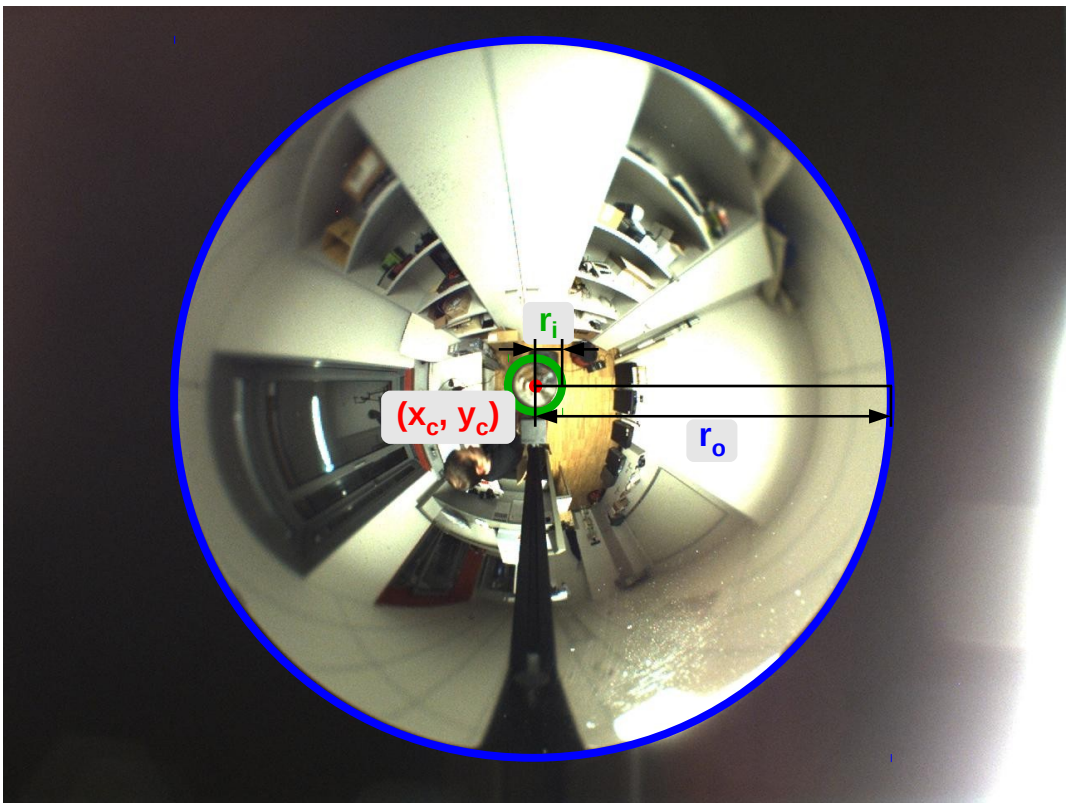



Figure 8: Schematic visualization of parameters for dewarping of 360° images. The outer circle is depicted in blue color, the inner circle is depicted in green color, the center is shown in red color.

# 4 Experiments

After the data were captured by means described in Chapter 3, the visual localization algorithm explained in Chapter 2 was tested in various experiments. First of all, the datasets used in the experiments will be described. Then, we will define the error measuring the accuracy of the position estimation. The description of individual experiments will follow.

## 4.1 Datasets

In order to test the performance of the algorithm under various conditions, 17 datasets were captured in 3 different areas at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. The datasets were captured under different lighting conditions, with and without the presence of people and with different paths of movement of the robot around the area to test the robustness of the proposed algorithm.

The data acquisition process can be controlled by several parameters. All the datasets were captured with the following setting of the parameters:

- the position of the robot is saved every 200 ms,
- an image is captured every 1000 ms.

The way how to set these parameters in the code is described in Appendix A.

In the experiments, some parts or combinations of the datasets are used as the database observations, other are used as the query observations. Therefore, the position data are available also for the query observations and they serve as the ground truth to evaluate the accuracy of the estimated position, as calculated by the algorithm.

### 4.1.1 Overview of areas

In advance to performing experiments, positions of all observations in all datasets were transformed to a common coordinate frame with the origin at the robot's 'depot' in the ViCoS laboratory[1]. All datasets were captured on the same floor of the same building and this transformation therefore allows to have an overall view of the locality of the captured data. Figure 9 shows all positions of images captured in 3 different areas: *Corridor*, *ViCoS lab* and *Coffee place*. The areas will be described in more detail in the following text.

[1] Visual Cognitive Systems laboratory, Faculty of Computer and Information Science, University of Ljubljana, Slovenia. URL: `http://www.vicos.si`

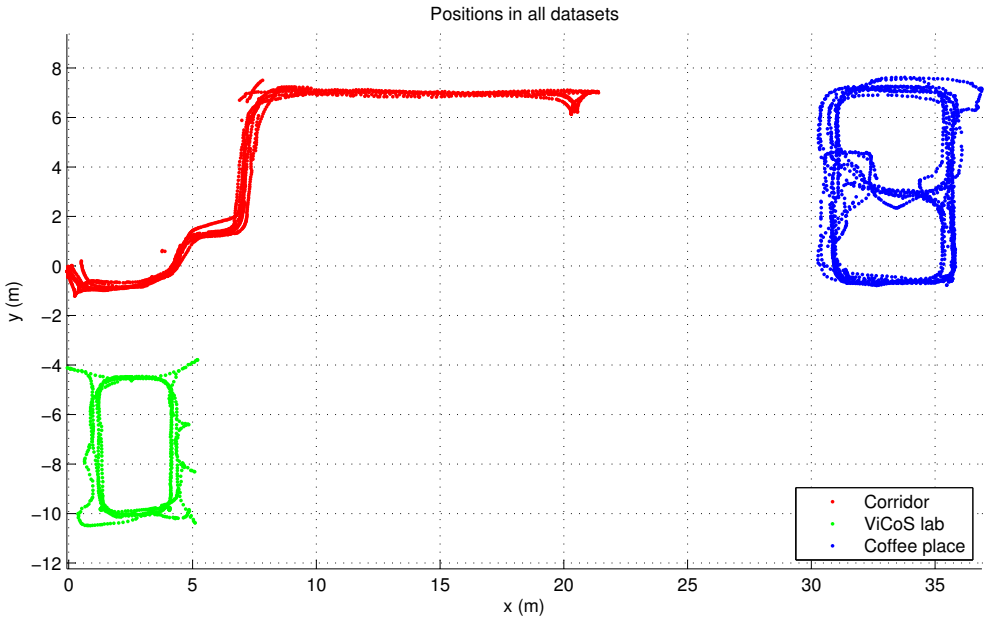



Figure 9: Visualization of areas with the positions of all captured images from all datasets.

| Dataset | Number of images | Lighting |
|---|---|---|
| *corridor1* | 683 | artificial |
| *corridor2* | 653 | artificial |
| *corridor3* | 749 | daylight |

Table 4: Overview of the datasets in the *Corridor* area.

### 4.1.2 Corridor area

All datasets in the *Corridor* area have almost the same trajectory, see Figure 10. The trajectory starts at the 'depot', continues through the small lab and across the bridge to the main corridor of the second floor and spans to the right towards the next bridge, where the robot turns and returns back to the 'depot'. The length of the trajectory is approx. 65 m. Two of the three datasets were captured in artificial light, one in daylight – see Table 4 for details. Figure 11 shows examples of images from the *Corridor* area.

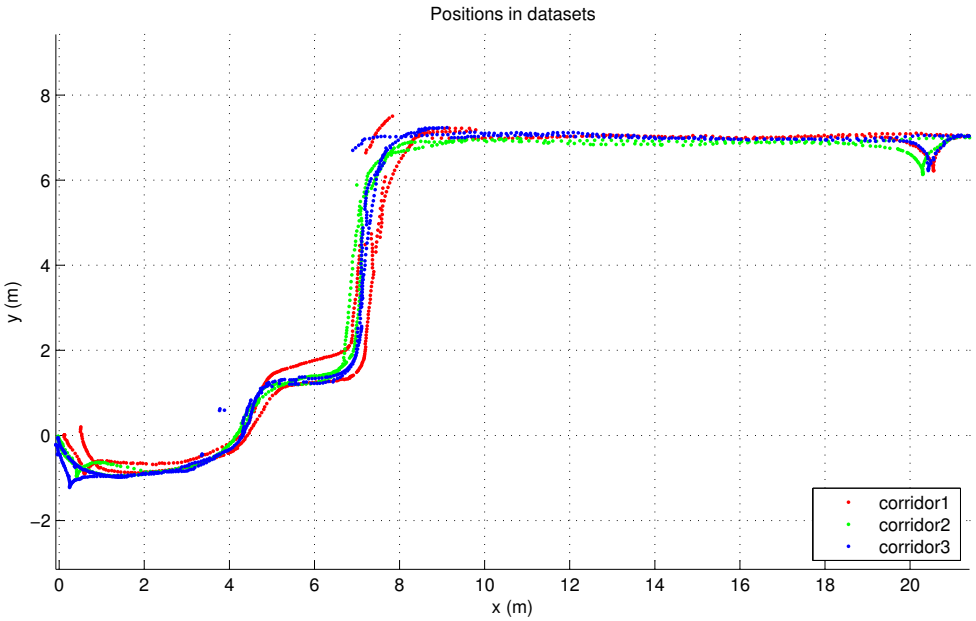



Figure 10: Visualization of positions in the *Corridor* area. The positions around (7,7) were generated as a result of measurement errors.

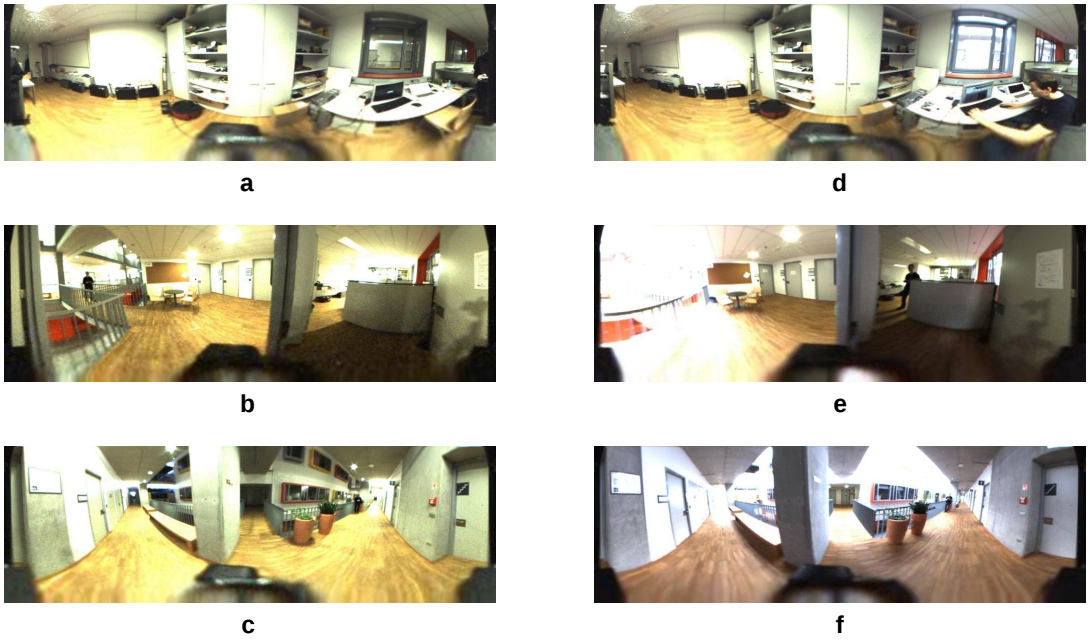



Figure 11: Examples of images in the *Corridor* area. Images (a), (b) and (c) are from datasets captured in artificial light, while images (d), (e) and (f) were taken at the same positions during daylight.

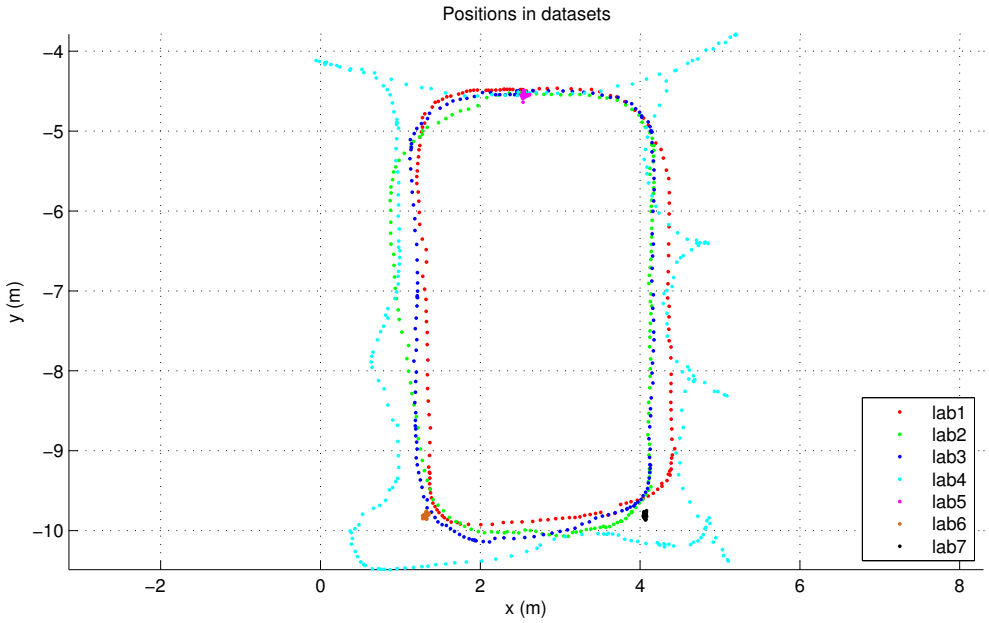



Figure 12: Visualization of positions in the *ViCoS lab* area.

### 4.1.3 ViCoS lab area

Seven datasets were captured in the main ViCoS laboratory, which is an area with many repetitive visual patterns and therefore it is considered rather difficult for vision-based algorithms. The datasets cover an area of approx. $5 \times 7$ meters. List of the datasets in this area can be found in Table 6. As can be seen in Figure 12, datasets *lab5*, *lab6* and *lab7* contain an on-spot rotation. These datasets are intended to be used in an experiment testing invariance of the algorithm to rotations. Examples of images from an on-spot rotation dataset are shown in Figure 13 (d), (e) and (f).

| Dataset | Number of images | Lighting |
|---|---|---|
| *lab1* | 195 | daylight |
| *lab2* | 174 | artificial |
| *lab3* | 170 | artificial |
| *lab4* | 238 | artificial |
| *lab5* | 86 | artificial |
| *lab6* | 88 | artificial |
| *lab7* | 83 | artificial |

Table 5: Overview of the datasets in the *ViCoS lab* area.


Figure 13: Examples of images in the *ViCoS lab* area. Image (a) was taken during daylight, images (b) and (c) are from datasets captured in artificial light. Images (d), (e) and (f) are from the dataset *lab6*, which features on-spot rotation.

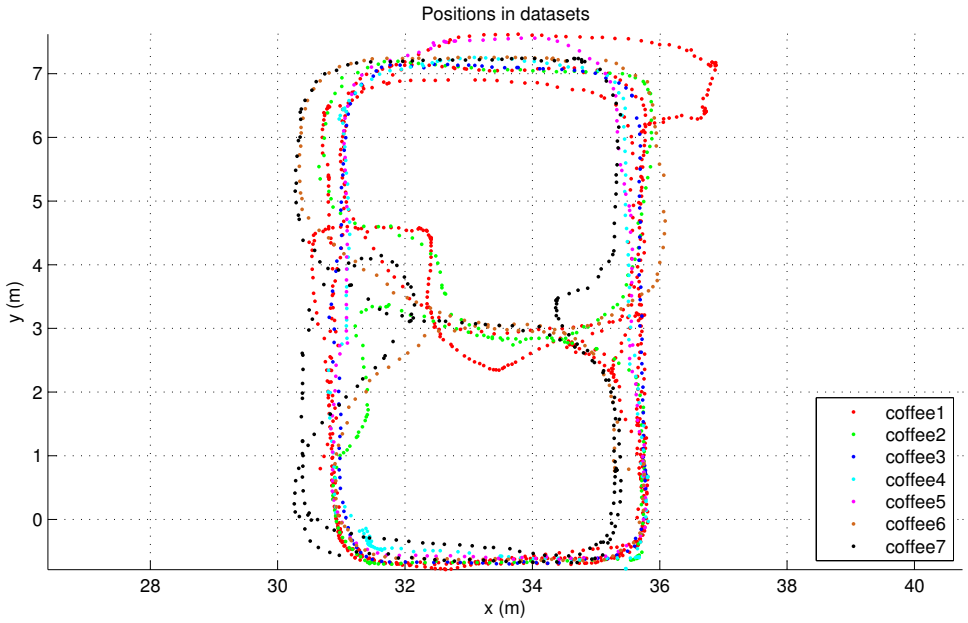



Figure 14: Visualization of positions in the *Coffee place* area.

### 4.1.4 Coffee place area

The last area, *Coffee place*, is a common resting place and workplace with a coffee machine, situated at the end of the corridor in the second floor of the faculty building, next to a balcony. Seven datasets were captured in this area, see Table 6 for the list of them. Figure 14 shows the visualization of positions in this area. The datasets cover an area of approx. $7 \times 9$ meters.

The area features large uniform surfaces and repetitive patterns. The balcony is separated from the coffee place by a glass wall, which has a very different appearance during the day and night, see Figure 15.

## 4.2 Measuring the error

Let us define at this point the error that will be used throughout the description of the experiments. The error is defined as the physical (Euclidean) distance between the ground truth position of a query observation and its estimated position, as calculated by the algorithm. It corresponds to Equation 2. A sample visualization of the error can be seen e.g. in Figure 17 as the black connectors of the green and blue points.

The collection of errors for all query observations in an experiment will be often visualized in the form of a box plot. Let us briefly describe the box plot, see Figure 16 for an example. The red line shows the median of the errors, i.e., the value that is in

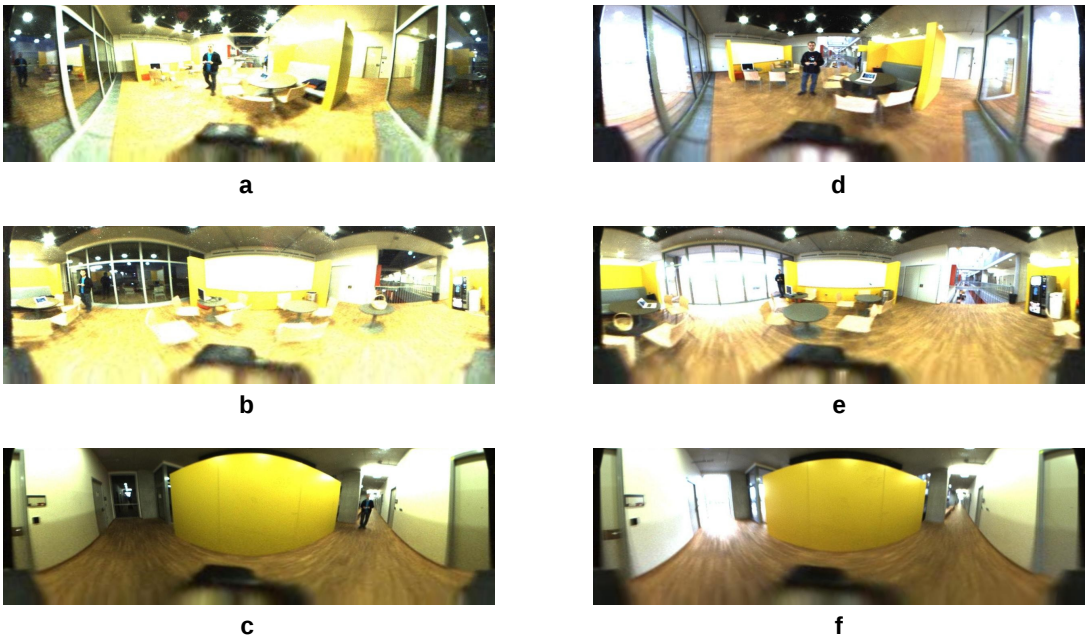



Figure 15: Examples of images in the *Coffee place* area. Images (a), (b) and (c) are from datasets captured in artificial light, while images (d), (e) and (f) were taken at the same positions during daylight.

| Dataset | Number of images | Lighting |
|---|---|---|
| *coffee1* | 759 | artificial |
| *coffee2* | 245 | artificial |
| *coffee3* | 143 | artificial |
| *coffee4* | 191 | daylight |
| *coffee5* | 137 | daylight |
| *coffee6* | 185 | daylight |
| *coffee7* | 260 | daylight |

Table 6: Overview of the datasets in the *Coffee place* area.

the middle of the sorted list of errors. The blue box extends from the 25th to the 75th percentile. The whiskers show the range of the errors that are not considered outliers. Finally, the red crosses represent outliers [18].

## 4.3 Overall performance

In the first experiment, the general performance of the visual localization algorithm is tested on a large database of images. The set of database and query images was created in the following way. All 7 datasets from the *Coffee place* area were joined together. In each 10 positions, the positions indexed 1–9 form the database and the 10th position becomes a query. In other words, 90 % of the datasets serve as the database observations and the remaining 10 % serve as the query observations.

Let us remind that each position in the recorded dataset expands to 36 observations, as the artificial rotations of an image are generated by shifting it with a step of 10°. The step of 20° is used in this experiment, i.e., every second rotation is left out. This reduces the number of observations per position to 18.

The database contains 1728 positions and 31104 observations, the query set consists of 192 observations. As in all experiments, only the observations in the original position, i.e., without any artificial rotation, are used as the query observations. The number of query positions and query observations is therefore equal.

### 4.3.1 Efficiency of the weighted $k$-NN algorithm

This experiment was utilized to test various aspects of the algorithm. Figure 16 shows, how the performance of the algorithm is dependent on the choice of $k$ in the weighted $k$-NN algorithm. It can be seen that the weighted $k$-NN algorithm outperforms the simple nearest neighbor algorithm ($k = 1$) in the vast majority of cases. The best results are achieved in this experiment for $k = 10$. Figures 17 and 18 show the position estimation accuracy when using the nearest neighbor algorithm and the weighted $k$-NN algorithm.

The positions in the dataset span over an area of approximately $6 \times 9$ meters. Provided that, the mean of the errors 0.10 m, standard deviation 0.10 m and median 0.07 m achieved with the weighted $k$-NN algorithm with $k = 10$ can be considered a very good result. Histogram of errors is shown in Figure 20. When using the simple nearest neighbor method, the error mean is 0.15 m, standard deviation 0.13 m and median 0.14 m. The histogram of errors for the simple nearest neighbor method is shown in Figure 19. It has to be taken into account that even though the database is very dense, there are basically no database observations at the exactly same positions as the query observations, so the localization is very precise even when using the simple nearest neighbor method.

### 4.3.2 Speed-up using clustering

Another purpose of the overall experiment is to test clustering, which is predicted to achieve algorithm speed-up while preserving the position estimation accuracy. The reasonable number of clusters for a dataset with approx. 30000 observations is 100, with an anticipated average of approx. 300 images per cluster. The clustering took 3 hours and 12 minutes, but the duration of the clustering process is strongly dependent on the random initialization of clusters. The observations, being partitioned based on the CNN descriptors, tend to preserve spatial locality within the clusters, i.e., clusters contain observations physically close to each other, see Figure 21. It is an evidence of that the descriptors of images taken at similar positions are similar.

Although the clustering takes a long time, it has to be done only once for a given database of observations and the achieved improvement in time for estimating the position of query observations is significant. To briefly summarize the clustering algorithm, the nearest neighbor (or $k$ nearest neighbors) of the query observation is searched first among the cluster means and then within the closest cluster. Therefore, in this experiment, instead of searching linearly through 30000 observations, it is necessary to search through only approximately $100 + 300$ observations when using clustering – 100 observations to find the closest cluster mean and 300 observations within the cluster.

As mentioned in Section 2.5.3, the clustering may take into account not only the subset of database observations belonging to the cluster with the closest cluster mean, but also more closest clusters. Figure 22 shows the accuracy of the algorithm for several values of closest clusters taken into account. Table 7 presents the time complexity of the position estimation algorithm for different number of closest clusters. The stated times are for the nearest neighbor method, but using the weighted $k$-NN algorithm instead does not change the times significantly for reasonable values of $k$. Please note that we use prepared datasets with dewarped images and precomputed descriptors, so that the time needed for dewarping the image (approx. 3.5 s using bilinear interpolation and full resolution) and computing the CNN descriptor (approx. 100 ms) is not included in these times.

The results show that the achieved speed-up is remarkable, the time needed for position estimation dropped by two orders of magnitude, from 1.455 s per image without clustering to 0.025 s per image with clustering using one closest cluster.

It is interesting to mention that the clustering may even bring a slight improvement in the position estimation accuracy. For example, when using the nearest neighbor method and clustering with $c = 1$ or $c = 2$ in this experiment, the average error decreases by several millimeters in comparison to position estimation without clustering.

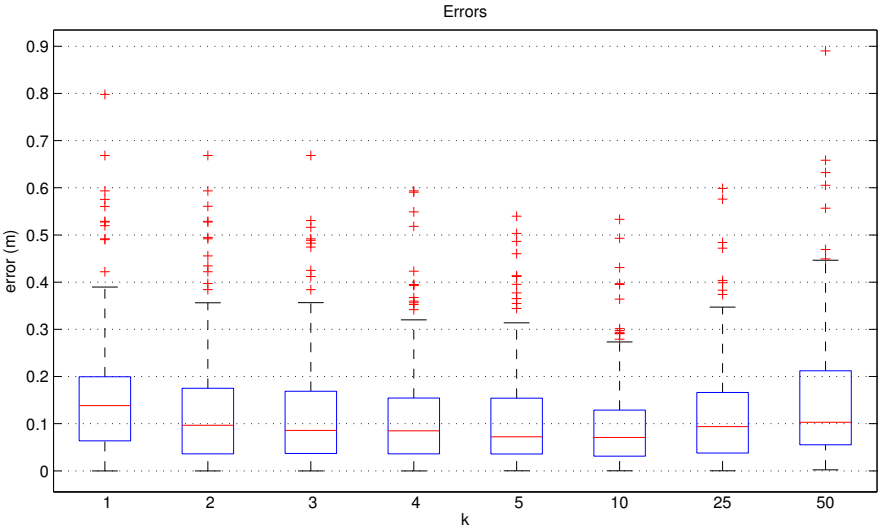



Figure 16: Box plot of errors in the overall experiment for selected values of $k$ in the weighted $k$-NN algorithm. Clustering was not applied in this case.

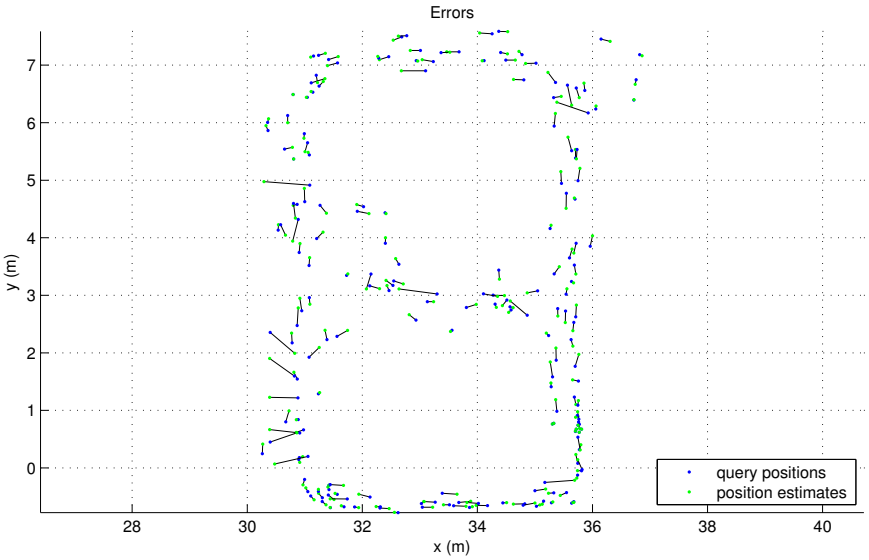



Figure 17: Visualization of distances of the estimated positions from the ground truth positions of the query observations in the overall experiment. The used algorithm is nearest neighbor, clustering was not applied.

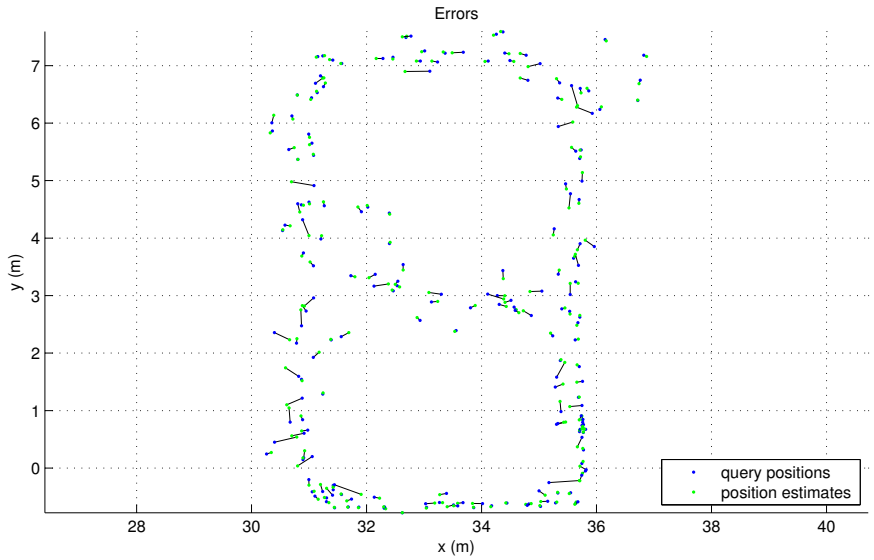



Figure 18: Visualization of distances of the estimated positions from the ground truth positions of the query observations in the overall experiment. The used algorithm is weighted $k$-NN with $k = 10$, clustering was not applied.

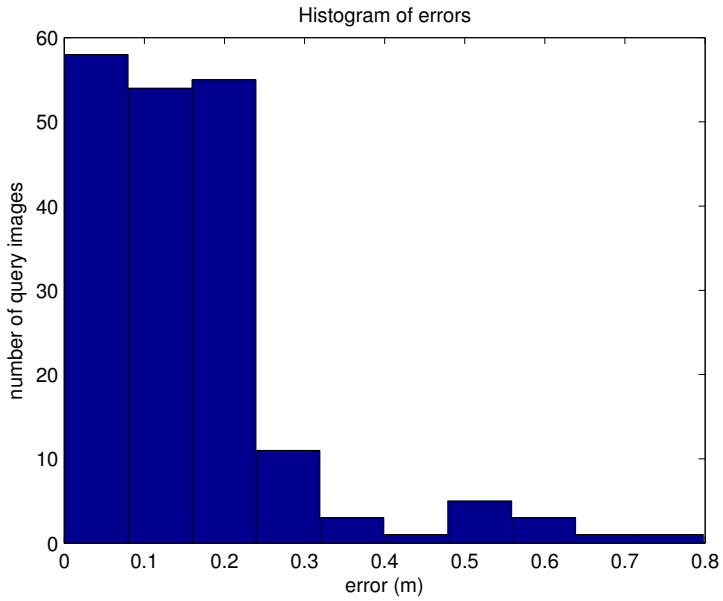



Figure 19: Histogram of position estimation errors in the overall experiment. The used algorithm is nearest neighbor, clustering was not applied.

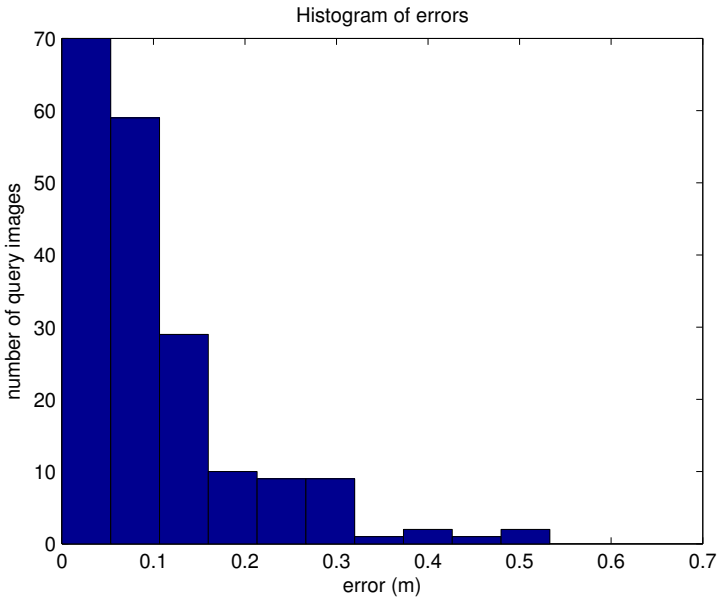



Figure 20: Histogram of position estimation errors in the overall experiment. The used algorithm is weighted $k$-NN with $k = 10$, clustering was not applied.

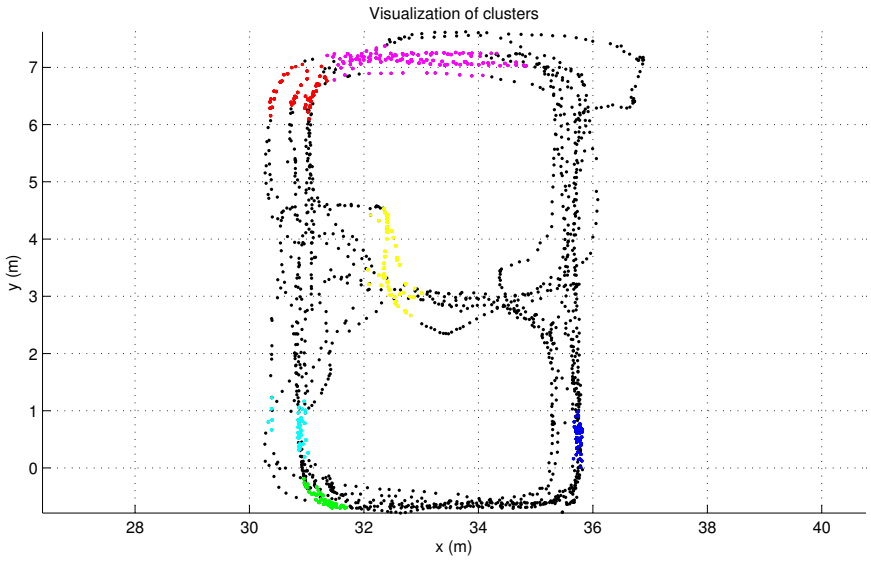



Figure 21: Examples of clusters in the overall experiment. Different colors represent 6 different clusters (out of the total of 100 clusters).

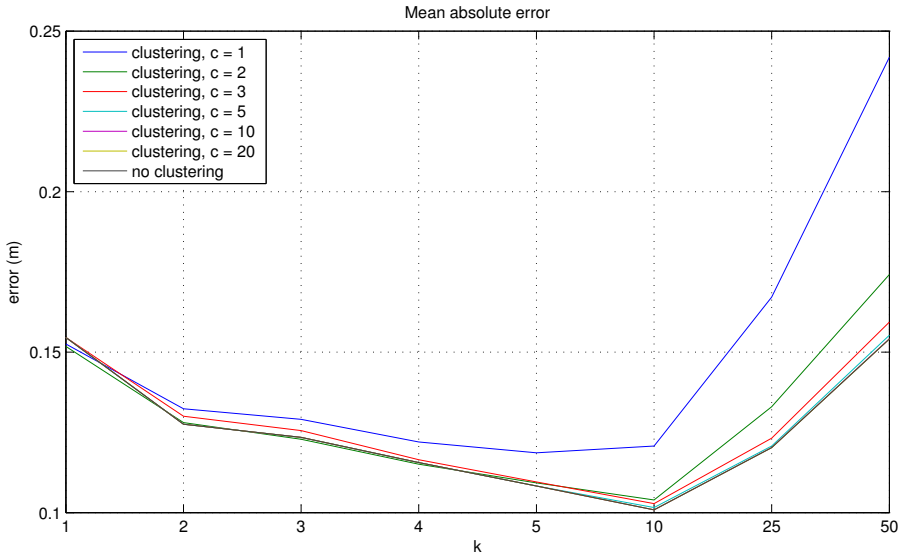



Figure 22: Comparison of mean absolute errors in the overall experiment for different number of used closest clusters $c$ for selected values of $k$ in the weighted $k$-NN algorithm.

| **# clusters** | 1 | 2 | 3 | 5 | 10 | 20 | Without clustering |
|---|---|---|---|---|---|---|---|
| **Time (s)** | 0.025 | 0.040 | 0.055 | 0.082 | 0.156 | 0.302 | 1.455 |

Table 7: Average time of position estimation per query observation in the overall experiment for different number of closest clusters $c$ taken into account. Nearest neighbor method was used. Measured on a machine with CPU Intel Core i5-3317U @ 1.7 GHz and 4 GB RAM.

## 4.4 Invariance to rotation

It is desirable that the algorithm is invariant to rotations, as the robot may access previously visited places in a different orientation. As described earlier, the process of building datasets generates for each panoramic image 36 rotated (shifted) variants with a step of 10°, which become observations. These observations may be used all, but in large image databases, it might be favorable to reduce the number of rotations to save memory. It will be shown in two experiments, how the algorithm performs for different steps of rotation.

As can be seen in Figure 7, the arm holding the paraboloidal reflector interferes with the scene when applying artificial rotation to the image. It is remarkable to mention that the presence of the arm in the panoramic images does not represent a significant drawback to the performance of the algorithm.

### 4.4.1 On-spot turning

The first experiment on rotation was performed with the *ViCoS lab* datasets. The last three datasets from this area (see Table 6) contain on-spot turns at three different locations in the laboratory, as can be seen in Figure 12. Examples of images from one of the on-spot turning datasets are shown in Figure 13 (d), (e) and (f). Since the datasets contain around 80–90 images each, the average rotation step between the query images is approximately 4–5°.

Observations from the datasets *lab2*, *lab3* and *lab4* were joined together to form the set of database observations, while the datasets *lab5*, *lab6* and *lab7* form a set of query observations. The query observations were filtered so that only observations in the default position (0° artificial rotation) remained. Measuring the performance of the algorithm for different levels of filtering of the database observations (by differently large steps of rotation) is the aim of this experiment.

Figure 23 shows the box plot capturing how precisely does the algorithm estimate the positions of the query observations for different levels of filtering of the database observations. Figure 24 shows the visualization of errors of position estimation when using all 36 rotation steps, whereas Figure 25 shows the errors of position estimation after filtering out all rotations but the default 0°. It is obvious that the artificial rotations improve the precision of position estimation significantly. In addition to the figures, it is worth mentioning that the mean absolute error is 0.17 m using artificial rotations (step 10°) and 0.57 m without artificial rotations, which means that using artificial rotations can reduce the average error more than 3×. Even when using only 4 rotations per image (step 90°), the performance is still better by a large margin than without using artificial rotations. The mean absolute error is in that case 0.23 m. This experiment shows therefore also that it is possible to remove some of the rotations from the database, when it is necessary to reduce the amount of the used memory, without a significant impact on the performance. The weighted $k$-NN algorithm with $k = 5$ was used in this experiment. Clustering was not applied.

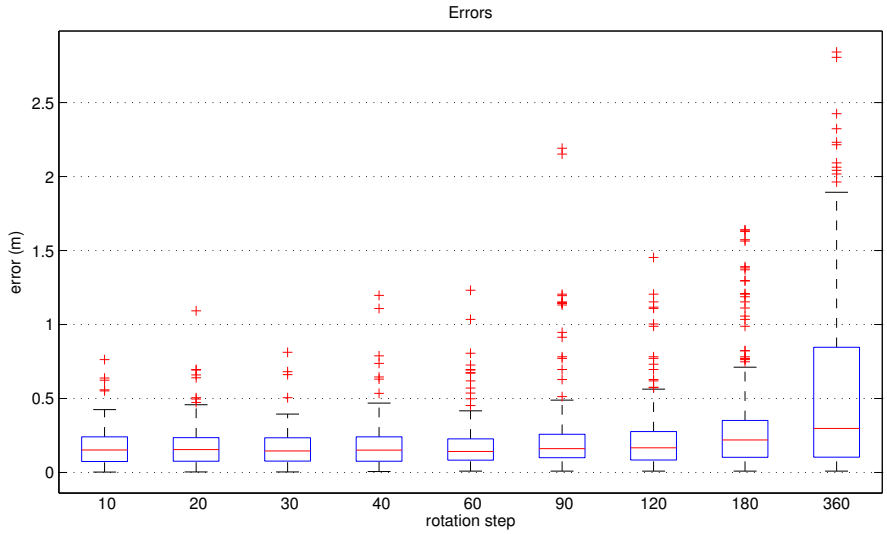



Figure 23: Box plot of errors for different rotation steps in the *on-spot turning* rotation experiment. The rotation step of 360° represents that the artificial rotations were not used, only the default 0° rotation.

### 4.4.2 Way there and back

In addition to the on-spot turning experiment, which is rather an artificially prepared experiment to test the robustness of the algorithm to rotation, a more real-world experiment was performed. The dataset *corridor2* serves as the source of both the database and query observations. This dataset covers a common foreseen scenario, when the robot passes through some path there and then back (see Figure 10). The observations recorded on the way there (1–290) form a database, the observations recorded on the way back (355–653) build up a query set. The observations close to the turning point (291–354) were discarded. As always, only the query observations with the default 0° rotation were used.

Box plot of the position estimation errors can be seen in Figure 26. It is notable that a significantly worse performance among the artificial rotation steps occurs in the case of the 120° step. The most likely explanation is that the images captured on the way there and on the way back differ by a 180° rotation, which is skipped by a too large margin when using the 120° step. Figures 27 and 28 show the errors of position estimation when using all 36 rotation steps and with only the default 0° rotation, respectively. The mean absolute error is 0.24 m with artificial rotations (step 10°) and 1.03 m without artificial rotations. These results again confirm that using artificial rotations improves the precision of the position estimation significantly.

Like in the on-spot turning experiment, the weighted $k$-NN algorithm with $k = 5$ was used, also without clustering.

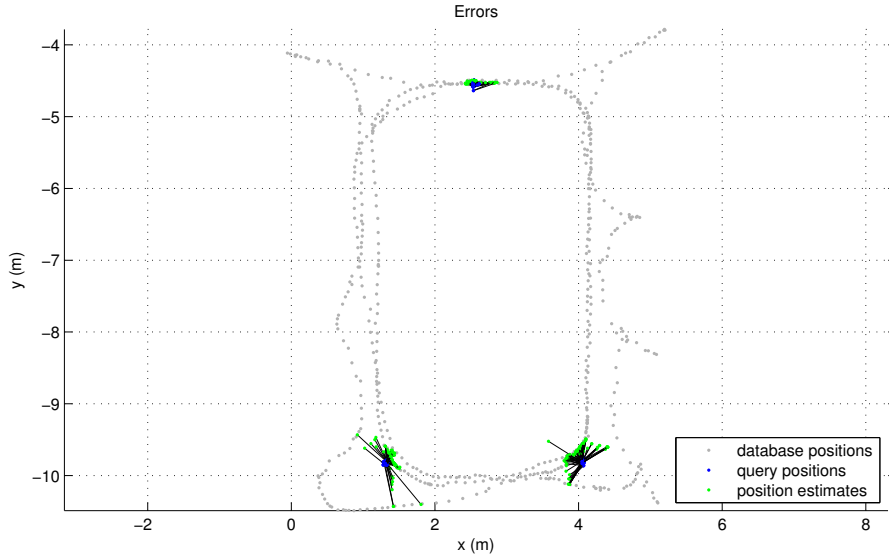



Figure 24: Visualization of errors in the *on-spot turning* rotation experiment using the weighted $k$-NN algorithm with $k = 5$ and a 10° rotation step for the database observations.

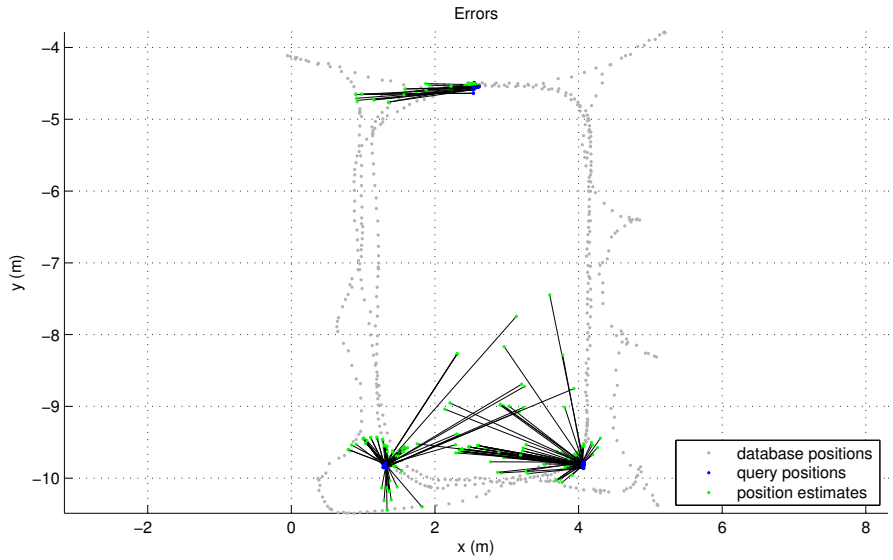



Figure 25: Visualization of errors in the *on-spot turning* rotation experiment using the weighted $k$-NN algorithm with $k = 5$ and only the default 0° rotation, i.e., without any artificial rotations of the database observations.

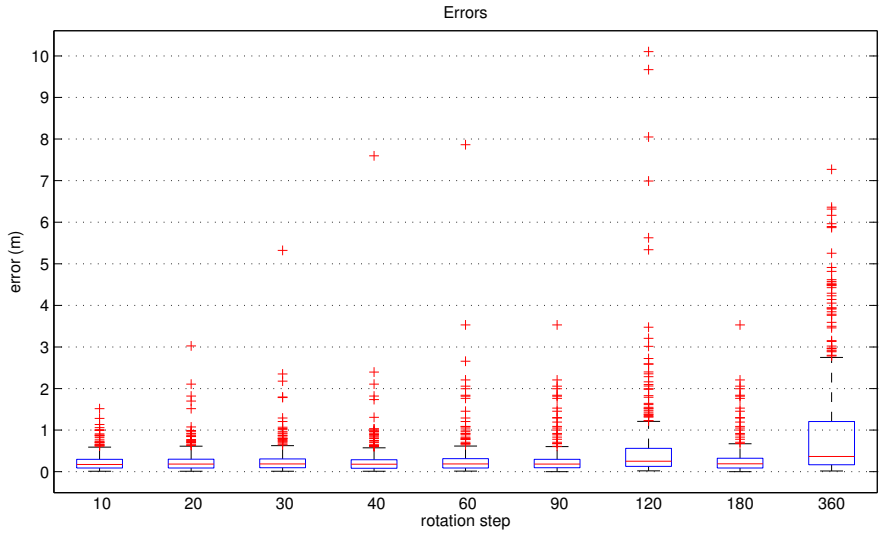



Figure 26: Box plot of the errors for different rotation steps in the *way there and back* rotation experiment. The rotation step of 360° represents that the artificial rotations were not used, only the default 0° rotation.

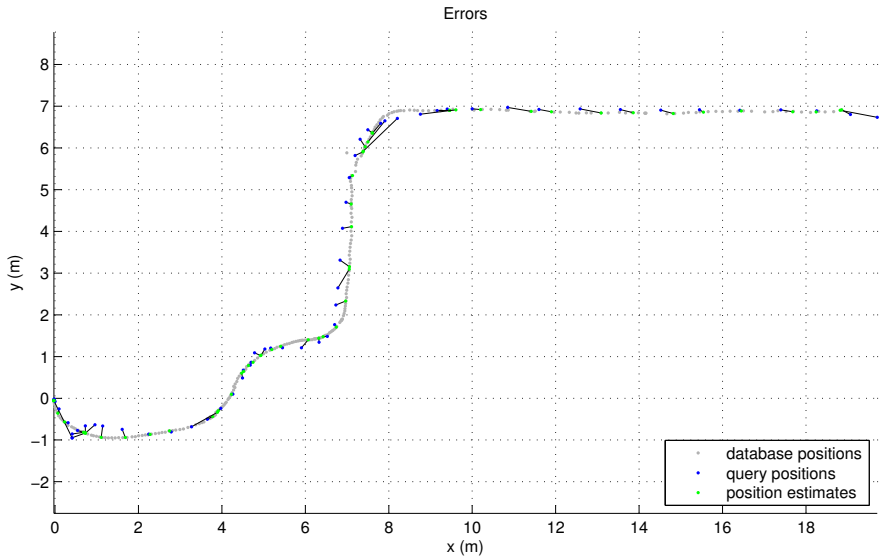



Figure 27: Visualization of errors in the *way there and back* rotation experiment using the weighted $k$-NN algorithm with $k = 5$ and a 10° rotation step for the database observations. Only every 5th query observation is shown to maintain the readability of the image.

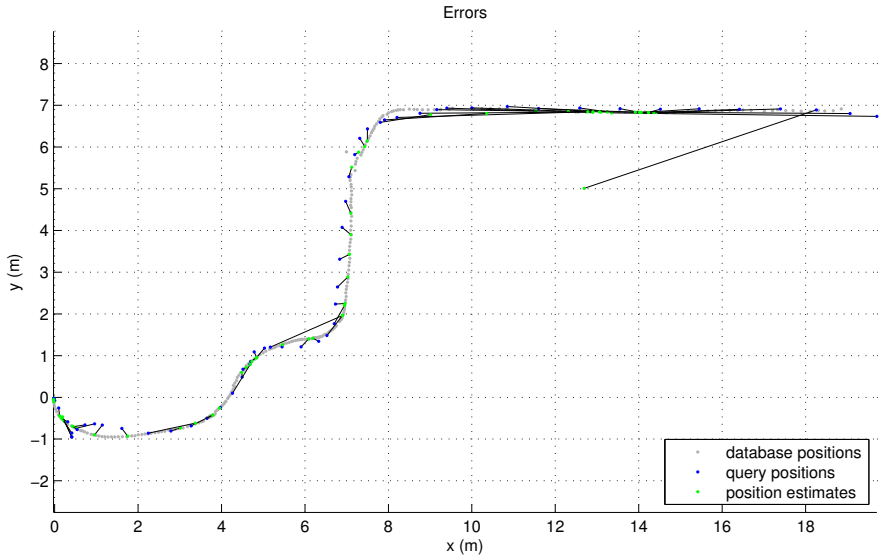



Figure 28: Visualization of errors in the *way there and back* rotation experiment using the weighted $k$-NN algorithm with $k = 5$ and only the default 0° rotation, i.e., without any artificial rotations of the database observations. Only every 5th query observation is shown to maintain the readability of the image.

## 4.5 Variable lighting conditions

The robot might enter previously visited areas at different times of a day and therefore under various lighting conditions. Two experiments were performed to test the robustness of the algorithm to lighting changes, one in the *Corridor* area, another one in the *Coffee place* area.

### 4.5.1 Corridor area

The first experiment was performed in the *Corridor* area. The union of the observations in the datasets *corridor1* and *corridor2*, which were captured under artificial light, serves as the database. These observations were filtered to achieve a 20° rotation step. There are in total 24048 database observations at 1336 positions. The query set is formed by taking all 0° observations from the *corridor3* dataset, which was captured in daylight. The query set contains 749 observations.

This experiment tests also the robustness of the algorithm to variable environment and occlusion, as there are people passing by through the corridor during capturing the sequence.

The query path is approximately 65 m long (there + back). The used algorithm was the weighted $k$-NN with $k = 5$, clustering was not applied. The experiment yielded

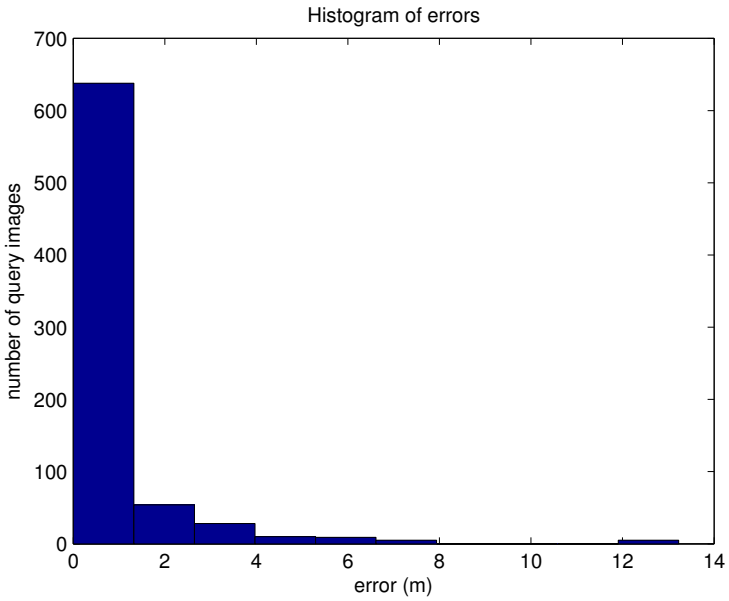



Figure 29: Histogram of errors in the lighting conditions experiment in the *Corridor* area using the weighted $k$-NN algorithm with $k = 5$ and a 20° rotation step for the database observations.

errors with the mean of 0.85 m, standard deviation 1.53 m and median 0.36 m. The histogram of errors is in Figure 29, the visualization of errors can be seen in Figure 30. The vast majority of the errors is rather low, considering that the same images taken in different lighting conditions show non-negligible differences, as can be seen in Figure 11 – it is in particular significant for the pair (c) and (f).

### 4.5.2 Coffee place area

The second experiment was performed in the *Coffee place* area. The datasets *coffee1–3* taken in artificial light represent the database, the datasets *coffee4–7* captured at daylight form the query set. The database observations were filtered to achieve a 20° rotation step, the query observations are only these with the 0° artificial rotation. There are in total 20646 database observations at 1147 positions. The query set contains 773 observations.

The weighted $k$-NN algorithm with $k = 5$ and without clustering was used. The resulting mean absolute error on the area of $7 \times 9$ m is 1.64 m, with standard deviation 1.97 m and median 0.60 m. These errors are rather large, but the histogram in Figure 31 shows that the significant majority of the errors is below 1 m. If we have a look at Figure 32, we can see that most of the incorrectly localized query observations are from the right side of the area. This side separates the corridor from the balcony by a glass wall and it can be seen in Figure 15 on the pair of images (a), (d) that the difference in appearance is significant. At night, the scene is mirroring in the glass

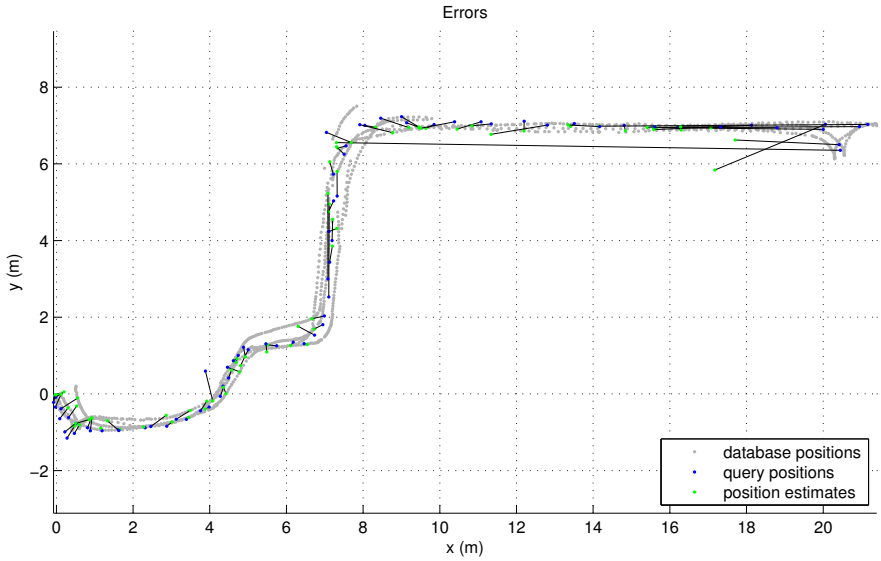



Figure 30: Visualization of errors in the lighting conditions experiment in the *Corridor* area using the weighted $k$-NN algorithm with $k = 5$ and a 20° rotation step for the database observations. Only every 10th query observation is shown to maintain the readability of the image.

under artificial light, whereas during the day, the glass looks as a uniform white surface. Such a significant difference in appearance is behind the capabilities of the algorithm.

## 4.6 Database density

The database of images may become very large and it might be desirable to reduce its size. One of the options how to save memory is to reduce the number of artificial rotations per position, which was discussed in Section 4.4. Another way, which will be tested in this experiment, is to reduce the number of positions. This experiment shows, how the reduction of the number of positions affects the position estimation accuracy.

The configuration of the database and query set was re-used from the *way there and back* experiment on rotations, described in Section 4.4.2. The rotation step for the database observations is fixed to 20°.

In this experiment, various portions of the database are used. The positions are filtered as a whole, with all observations belonging to them. In other words, if an observation is removed from the database, then all the other observations with different artificial rotations originating from the same position are removed too. The results of the experiment when keeping only every $s$-th database position are summarized in the box plot in Figure 33. The visualization of the errors is shown in Figure 34 for using all database observations ($s = 1$) and in Figure 35 for using only observations belonging to every 20th database position.

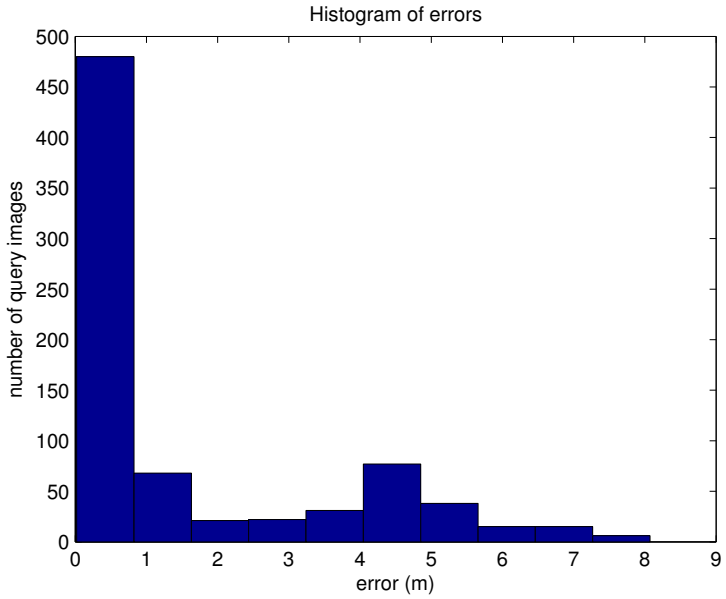



Figure 31: Histogram of errors in the lighting conditions experiment in the *Coffee place* area using the weighted $k$-NN algorithm with $k = 5$ and a 20° rotation step for the database observations.

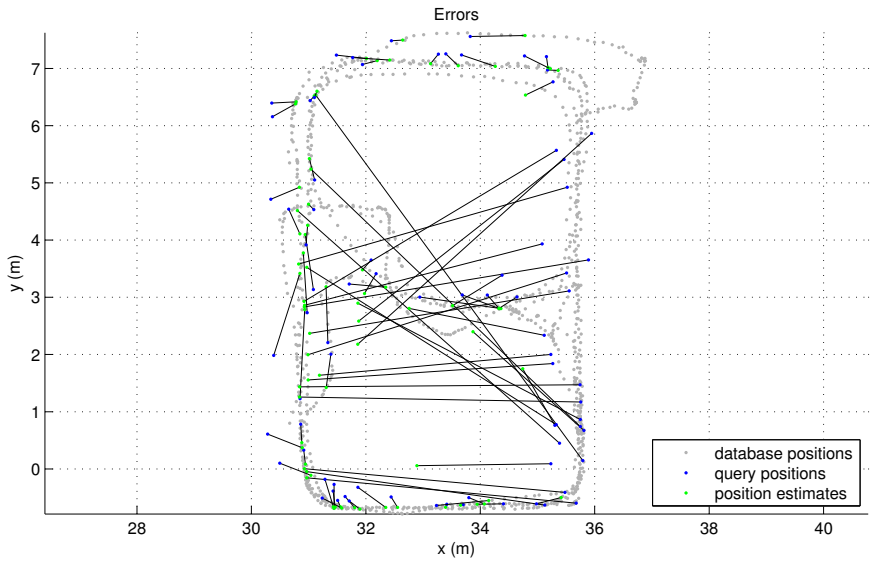



Figure 32: Visualization of errors in the lighting conditions experiment in the *Coffee place* area using the weighted $k$-NN algorithm with $k = 5$ and a 20° rotation step for the database observations. Only every 10th query observation is shown to maintain the readability of the image.

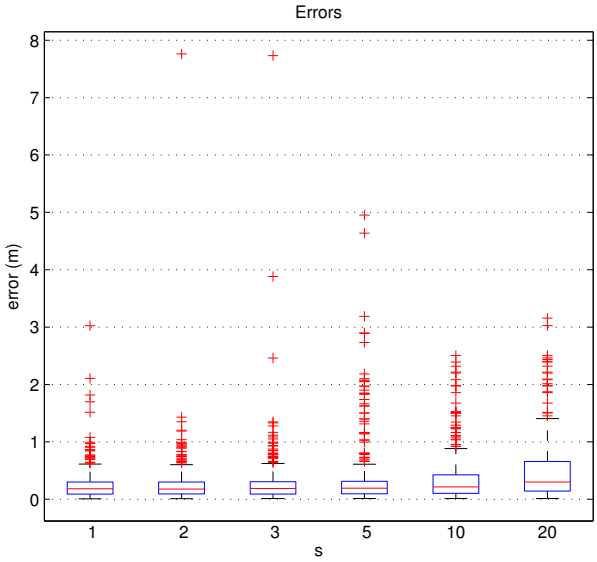



Figure 33: Box plot of errors in the database density experiment using only observations belonging to every $s$-th database position.

The database that we are using is obviously unnecesarilly dense, as the precision of the position estimation is still almost the same when using only every 5th position. The memory can be therefore saved by deleting up to 4/5 of the database. It is necessary to mention that this holds only for using the weighted $k$-NN algorithm, as leaving out the positions would have more considerable effect on the simple nearest neighbor algorithm.

## 4.7 Place recognition

The aim of this experiment is to test, how well does the algorithm identify the rough location of the robot in terms of areas. We are therefore not interested too much in the position estimation accuracy in that case, but rather in the classification of the area. The set of database observations was constructed by joining the datasets *corridor1*, *lab2* and *coffee4*, each from a different area. The set of query observations consists of another three datasets, *corridor2*, *lab3* and *coffee5*, again one from each area. The database observations were filtered to achieve the rotation step of 20°. Only the query observations with the default 0° rotation were used. This yields the total of 18864 database observations at 1048 positions and 960 query observations in this experiment.

The experiment was run with the weighted $k$-NN algorithm ($k = 5$) and without clustering. Figure 36 shows the visualization of the errors. It can be seen that the accuracy of the place recognition is 100 %, all images were classified to the correct area. It can be also seen that the position estimation accuracy is the worst in the *ViCoS lab* area. It is probably the most difficult one among the captured areas, as it contains many similarly looking structures.

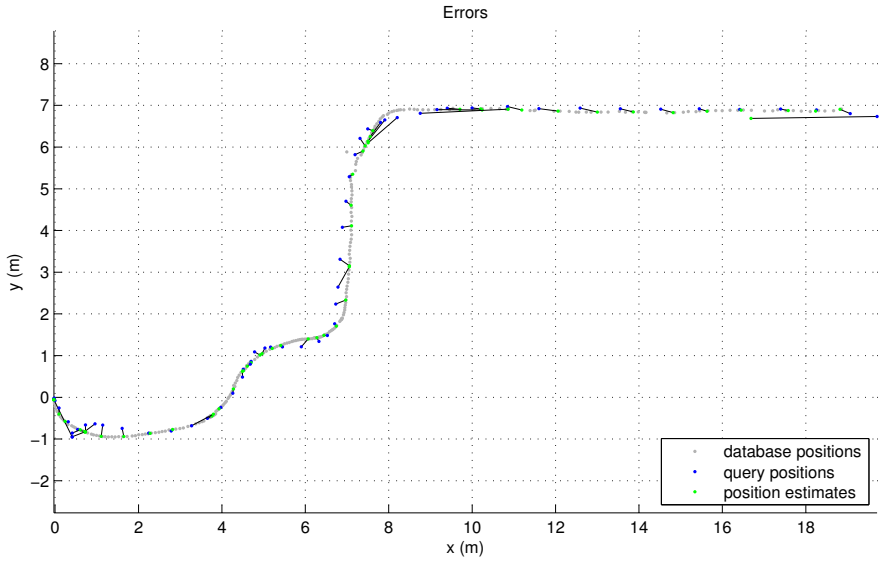



Figure 34: Visualization of errors in the database density experiment using all database observations.

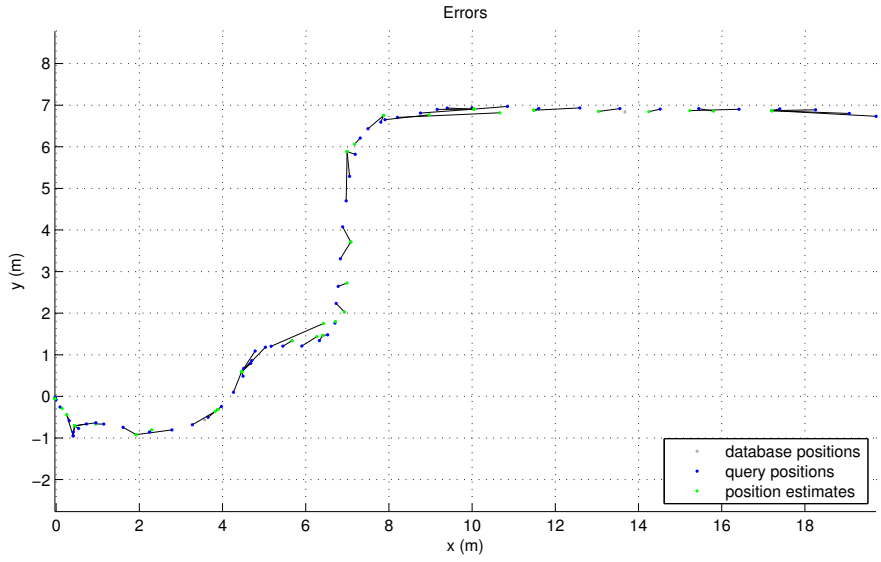



Figure 35: Visualization of errors in the database density experiment using only observations belonging to every 20th database position.

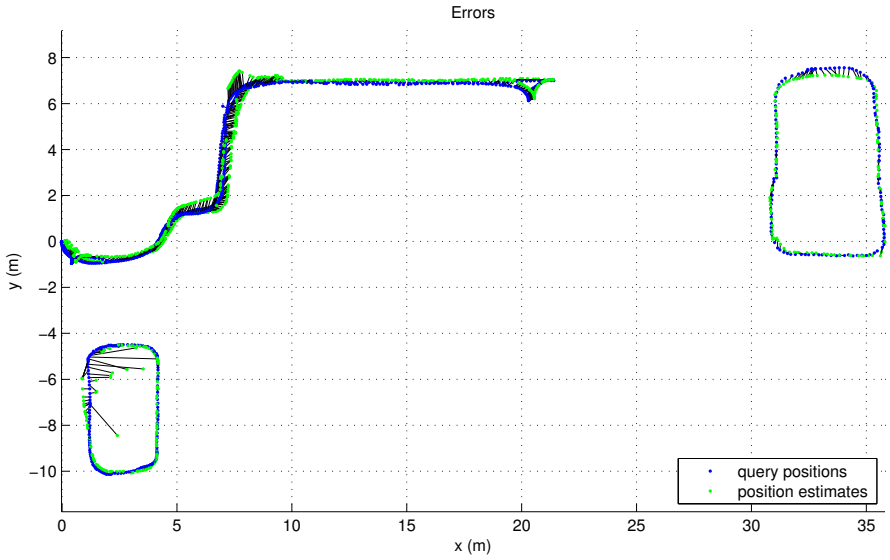



Figure 36: Visualization of errors in the place recognition experiment using the weighted $k$-NN algorithm with $k = 5$ and a 20° rotation step for the database observations.

# 5 Conclusion

A method for vision-based localization using omnidirectional images and convolutional neural networks was proposed in this work and tested in several experiments. A total of 17 datasets were captured in 3 different areas and then combined to database and query sets in various scenarios, testing different capabilities of the proposed algorithm. The concept of observations was introduced as a wrapper for an image with its position and other relevant parameters. The position of a query image is estimated by making use of descriptors computed by a convolutional neural network. The CNN approach proved to be suitable and powerful for the vision-based robot localization task, as expected based on the results from the previous and related work.

The proposed method was tested on a mobile robot in the Visual Cognitive Systems laboratory at the Faculty of Computer and Information Science at the University of Ljubljana. The existing framework featuring robot control and mapping, implemented mainly in ROS by the members of the ViCoS laboratory, was extended in this work by adding functionality for saving the position of the robot using laser rangefinder and for saving 360° images from the camera.

A collection of Matlab functions and scripts was created to process the recorded data, build a database of observations and carry out the position estimation task in various experiments.

## 5.1 Summary of results

The basic version of the position estimation algorithm consists in finding the nearest neighbor of the query image by minimizing the dissimilarity measure, defined as the square of the Euclidean distance between the CNN descriptors. The dissimilarity measure is minimized over all database observations by linearly iterating through the whole database. Even the simple nearest neighbor method offers precise localization, which can be demonstrated on the result of the experiment testing the overall performance of the algorithm. Provided that the observations in that experiment cover an area of approx. $6 \times 9$ meters, the average position estimation error of 0.15 m can be considered a good result, moreover taking into account also possible inaccuracies of the measured positions in the order of centimeters.

Two extensions of the nearest neighbor method were implemented and both proved to be beneficial. One extension consists in using more observations with the most similar descriptors by the means of the weighted $k$-nearest neighbor method. A significant improvement in position estimation was achieved – in the overall experiment, the mean absolute error decreased by 33 % and the median of the errors decreased by 50 % when using the weighted $k$-NN algorithm with $k = 10$, as compared to the simple nearest neighbor method.

Another extension that was implemented and tested is clustering of database observations. Although the partitioning takes a long time (several hours for a database size

of the order of $10^4$ observations), it has to be done only once and the speed-up of the position estimation is significant. The results of the overall experiment show that the nearest neighbor method is more than 50× faster when using clustering, considering only one closest cluster to be searched through when looking up the nearest neighbor.

The database of recorded images is enriched by rotating (shifting) the images to simulate different orientations of the robot. Although this process takes additional time in the dataset building stage, it proved to be very useful, as the mean absolute error decreased by more than 70 % in both performed experiments when using 36 rotations of each database image (step 10°) in comparison to using only images in the default orientation, without artificial rotations.

The experiments also showed that the memory used by the database can be reduced substantially by leaving out some of the artificial rotations or even whole positions without a significant influence on the position estimation.

The achieved results of the experiment on variable lighting conditions are considered satisfactory, provided that the appearance of many omnidirectional images taken at the same positions at different lighting conditions differs significantly.

## 5.2 Future work

The implementation of the algorithm is focused on performing experiments and visualization of results. For that reason, several computations could be avoided in a real application, which would decrease the time needed for the position estimation. For instance, significant speed-up could be achieved by dewarping the 360° images directly to the size used by the convolutional neural network, which was not desirable in our case in order to build high-resolution datasets that can be used beyond this work.

Since the implemented framework using the concept of observations is quite general, it could be easily extended to use other descriptors and methods. It would be convenient to compare the results of the algorithm on the same datasets with other approaches in the future.

The algorithm estimates the position of the robot as the $x$, $y$ coordinates. A possible extension would be to estimate also the orientation of the robot. This could be done by modifying the dataset recording process to save the real-world orientation of the robot together with the $x$, $y$ coordinates. The concept of observations is already designed to store the information about the orientation of the robot. The rotation angle, which is currently used only for artificial rotations and is set to 0° for the original recorded images, could be set to the actual rotation of the robot. The artificial rotation angles would be then set by adding the step (e.g. 10°) to the real-world orientation.

The datasets recorded by the mobile robot were captured so far only indoors. Although the framework is prepared to handle outdoor scenes, the algorithm could not be tested on outdoor sequences in this work due to technical issues with the GPS receiver. It would be convenient to capture also outdoor datasets to confirm our assumption that the proposed algorithm should work well both on images captured indoors and outdoors.

# A Data capturing manual

In order to make the created framework for capturing images available also to other users of the robot, the step-by-step procedure of capturing the data from the robot's sensors will be described in the following text.

All the commands are supposed to be run in a standard Linux shell console.

## A.1 Robot control

First of all, `roscore` has to be run to start the ROS framework services. After that, we can launch the main operation controller `atrv.launch` by the command
`roslaunch atrv_mini_ros atrv.launch`
This launch file runs the Rumblepad controller and enables us to drive the robot.

## A.2 Position

Next, we launch the `gmapping_hokuyo.launch` file by running the command
`roslaunch atrv_mini_ros gmapping_hokuyo.launch`
This launch file runs several nodes. It operates the laser scanner, runs the ROS visualization tool RViz, which displays the gradually built map, and runs the `gmapping` node [19]. Altogether, this launch file is responsible for building a 2D map of the explored area by reading the odometry data from the robot's actuators and correcting them by the data received from the laser scanner.

After that, we can run the `laser_pose_saver` node to save the position of the robot. The node is run by its launch file:
`roslaunch atrv_mini_ros laser_pose_saver.launch`
This node calculates the position of the robot using the `tf` messages produced by the `gmapping` node. The output text file has each row in the format
`[timestamp];[x];[y]`
The floating point values $x$, $y$ represent the position and they are in meters. There are two parameters that can be set in the `laser_pose_saver.launch` file: `file_path`, which is the full path to the file, into which the robot position is saved, and `skip_rate`, which is an integer $n$ specifying that only every $n$-th relevant transformation message published on the `/tf` topic is processed and the calculated position is saved.

## A.3 Images

The camera installed on the robot is not compatible with the ROS system, in particular with the `camera1394` package, so it was necessary to build a standalone program to capture images. The program `TimestampImages` is implemented in C++ using the

FlyCapture SDK that comes with the Point Grey cameras [20]. The program has the following synopsis:
`sudo ./TimestampImages [target directory] [delay between images in ms]`
The program saves the images named by the timestamp when they were captured, i.e., `[seconds].[nanoseconds].jpg`.

## A.4 Parameters used in experiments

The two parameters controlling the data acquisition process were set in all experiments performed in this work in the following way:

- the `skip_rate` parameter in the `laser_pose_saver.launch` file (Section A.2) was set to 2, i.e., position of the robot is saved every 200 ms,
- the `delay` parameter of the `TimestampImages` image capturing program (Section A.3) was set to 935, which means that an image is captured every 1000 ms (it was experimentally tested that the image saving process takes 65 ms).

# B Dataset building manual

The following text describes the step-by-step procedure of building datasets from the captured data.

The data from sensors are by default saved into subdirectories of a directory named as `[year][month][day]_[hour][minute]`, which represents one dataset. Images produced by the program `TimestampImages` are saved to the `img` subdirectory, positions of the robot are saved to a text file in the `laserPose` subdirectory.

## B.1 Converting 360° images to wide panoramic images

First of all, the 360° circular images have to be converted to panoramic images. The process of dewarping images is performed by the following two Matlab scripts, which have to be run for each dataset:

1. `align_circles.m` – an interactive tool for calibration, i.e., selection of the center and radii of the inner and outer circles limiting the 360° image; it saves the alignment data to the file `alignment.mat` in the `img` subfolder of the dataset,

2. `dewarp_folder.m` – reads the alignment data from the file `alignment.mat` and transforms the 360° images from the `img` subfolder to wide panoramic images and saves them to the `pano` subfolder with the same filenames.

## B.2 Computing descriptors

The descriptors are calculated for all images in the dataset using the Matlab function `describe_allrot()`. Each panoramic image is rotated in 36 steps, i.e., with a step size 10°, and for all of these rotations, the descriptors are calculated. This function creates in the `pano` subfolder a file `desc.mat` with the descriptors of all images present in the `pano` subfolder.

## B.3 Generating observations

Everything is prepared now to generate observations. Observations are from the implementation point of view instances of a custom Matlab class `Observation`.

The Matlab script `save_observations.m` finds for each image in the `pano` subfolder the matching (time-nearest) position in the text file with the positions present in the `laserPose` subfolder and fills in all the fields of the current observation (`im_pathname`, `dataset`, `position`, `desc`, `rotation` and `timestamp`). Each image yields 36 observations, one for each of the rotation steps (10°), which is specified in the field `rotation`.

The `timestamp` is the timestamp of the image (not of the position). Both the `timestamp` and the `dataset` fields could be extracted from the field `im_pathname`, yet it is advantageous to have them stored separately for performance reasons.

The script `save_observations.m` produces two arrays, `observations` containing all generated instances of the `Observation` class, and `positions`, a structure with fields `x`, `y` and `timestamp`. The latter is used only for faster and easier visualization of results and it is not necessary for the position estimation algorithm itself. Both arrays are saved to the file `observations.mat` in the `pano` subfolder.

## B.4 Captured datasets

The datasets were renamed in this work to improve readability. The matching between the names of the datasets used in this work and the names used in the code is presented in Table 8.

| Name in the code | Name in the text |
|---|---|
| 20150313_2012 | *corridor1* |
| 20150313_2032 | *corridor2* |
| 20150318_0902 | *corridor3* |
| 20150326_0735 | *lab1* |
| 20150327_2013 | *lab2* |
| 20150327_2017 | *lab3* |
| 20150327_2022 | *lab4* |
| 20150327_2025 | *lab5* |
| 20150327_2028 | *lab6* |
| 20150327_2031 | *lab7* |
| 20150326_1852 | *coffee1* |
| 20150326_1858 | *coffee2* |
| 20150326_1903 | *coffee3* |
| 20150327_1712 | *coffee4* |
| 20150327_1715 | *coffee5* |
| 20150327_1720 | *coffee6* |
| 20150327_1725 | *coffee7* |

Table 8: Matching between the names of the datasets used in this work and the names used in the code.

# C Contents of the enclosed DVD

The DVD contains 4 folders:

- `matlab` – Matlab codes – implementation of the algorithm,
- `text` – the text of this thesis – PDF and LaTeX source files,
- `capturing` – contribution to the framework developed in the ViCoS laboratory for data capturing on the ATRV mini robot – recording of images and positions,
- `data` – all datasets used in this work, containing original 360° images, dewarped images, ground truth positions and computed CNN descriptors.

# References

[1] Winston Churchill and Paul Newman. Experience-based navigation for long-term localisation. *The International Journal of Robotics Research (IJRR)*, 32:1645–1661, 2013.

[2] Michael Milford and Gordon Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649, May 2012.

[3] Tomáš Krajník, Jaime P. Fentanes, Oscar M. Mozos, Tom Duckett, Johan Ekekrantz, and Marc Hanheide. Long-term topological localisation for service robots in dynamic environments using spectral maps. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4537–4542, September 2014.

[4] Ana C. Murillo, Gautam Singh, Jana Košecká, and José J. Guerrero. Localization in urban environments using a panoramic GIST descriptor. *IEEE Transactions on Robotics*, 29(1):146–160, February 2013.

[5] Zetao Chen, Obadiah Lam, Adam Jacobson, and Michael Milford. Convolutional neural network-based place recognition. *CoRR*, abs/1411.1509, 2014.

[6] Erik Derner and Tomáš Svoboda. Indexing images for visual memory by using DNN descriptors – preliminary experiments. Research Report CTU–CMP–2014–25, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, December 2014.

[7] Otakar Jašek and Tomáš Svoboda. Visual-based memory using GIST descriptor. Research Report CTU–CMP–2014–10, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, October 2014.

[8] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence – Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint 1408.5093*, 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Csaba D. Toth, Joseph O'Rourke, and Jacob E. Goodman. *Handbook of Discrete and Computational Geometry, Second Edition*. Discrete and Combinatorial Mathematics Series. CRC Press, 2004. Pages 877–892.

[12] Anand Rajaraman and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011. Pages 69–73.

[13] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. Pages 284–288.

[14] ROS.org. About ROS. http://www.ros.org/about-ros/. Accessed: 2015-04-30.

[15] Iztok Oder. Lokalizacija mobilnega robota s pomočjo večsmerne kamere. Bachelor's Thesis, Faculty of Computer and Information Science, University of Ljubljana, Slovenia. http://eprints.fri.uni-lj.si/2703/1/63110328-IZTOK_ODER-Lokalizacija_mobilnega_robota_s_pomo%C4%8Djo_ve%C4%8Dsmerne_kamere_.pdf, 2014.

[16] Point Grey Research, Inc. Chameleon 1.3 MP color USB 2.0 camera (Sony ICX445). http://www.ptgrey.com/chameleon-13-mp-color-usb-2-sony-icx445-camera. Accessed: 2015-04-29.

[17] Hokuyo Automatic Co., Ltd. Scanning range finder URG-04LX. https://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html. Accessed: 2015-04-29.

[18] The MathWorks, Inc. Box plot – MATLAB boxplot. http://www.mathworks.com/help/stats/boxplot.html. Accessed: 2015-05-10.

[19] ROS.org. Gmapping – ROS Wiki. http://www.ros.org/gmapping/. Accessed: 2015-04-30.

[20] Point Grey Research, Inc. FlyCapture SDK. http://www.ptgrey.com/flycapture-sdk. Accessed: 2015-04-30.