

Visual navigation and control of a mobile robot

Fredrik Rudbeck

Supervisors

Anders Heyden

Anders Robertsson

Examiner

Magnus Oskarsson

Abstract

As cameras become cheaper and better, using them as sensors have become increasingly more common. This is especially true in robotics and for autonomous robot navigation. This thesis suggests a method where inputs from a single camera are used for making a robot self driving. The robot is assumed to move along planar surfaces, common in indoor environments.

The method is based on matching image features to a prebuilt navigation map also made up of image features. The method is evaluated by making a successful implementation on a mobile robot.

Since a camera is the only input, it proves that the quality of the camera is of high importance. All cameras in motion suffer from distortion. It is referred to as motion blur and arises when the scene changes during the recording of a single exposure. To try and exploit that cameras are cheap, it is investigated if a simple USB camera is sufficient for the method. It proves to be, but to keep the motion blur low, the maximum velocity of the robot is heavily limited.

Acknowledgments

I would like to thank my supervisors Anders Heyden och Anders Robertsson for their invaluable help and guidance. They have shared their knowledge, listened and understood when I shared my concerns and finally, shared my passion for the work.

I would also like to give a special thanks to Mårten Wadenbäck for two things. Firstly, for allowing me to access and use parts of his software. Secondly, for showing a genuine interest in my progress throughout the thesis.

Contents

1	Introduction	5
1.1	Background	5
1.2	Basic idea	6
2	Problem formulation	7
3	Camera and image theory	9
3.1	Pinhole perspective camera model	9
3.2	Homography	11
3.3	Features and feature extraction	12
3.4	Feature matching	12
3.5	RANSAC	12
3.6	Image registration	13
4	Hardware and software	16
4.1	Robot	16
4.2	Camera	17
4.3	ROS	18
4.4	OpenCV	19
4.5	Matlab	19
5	Extracting image data	20
5.1	Floor	20
5.2	Image preprocessing	21
5.3	Camera distortion	22
5.4	Camera calibration	23
5.5	Chessboard camera calibration	23
5.6	Camera tilt estimation	25
5.7	Feature extraction	27
5.7.1	SURF	27
6	Finding the camera pose	29
6.1	Navigation map	29
6.2	Removing bad matches	31

6.3	Robust image registration with RANSAC	31
6.4	The camera pose	32
6.5	Speeded-up matching	32
6.5.1	Incorrect estimates	33
7	Robot control	34
7.1	Problem definition	34
7.2	The camera movement algorithm	35
7.3	The robot movements	35
7.3.1	Finding the relative camera position	36
8	Results	37
8.1	Matches and pose estimation	38
8.2	Speeded-up matching	38
8.3	Motion blur	39
8.4	Robot movements	40
9	Discussion	42
9.1	Matching results	42
9.2	Motion blur	43
9.3	Speeded-up matching	43
9.4	Path following	44
9.5	Map	46
9.6	Conclusion	47
10	Future work	48
10.1	Path	48
10.2	Robot control	49
10.3	Map coordinates	49
10.4	Map	49

Chapter 1

Introduction

1.1 Background

Robotics and computer vision used to be very different research areas. Over time, cameras have become better and cheaper and is therefore a more common sensor in robotics, coupling the two fields into something often referred to as robot vision. A camera on a robot is what the eyes are to a human. It allows us to see and collect information about the world. There are other sensors such as radars, sonars, laser scanners and many more which may observe the structure of the world, but to fully capture texture and color a camera is needed.

Autonomous navigation is a heavily researched area where cameras play an important role. One of the more advanced systems one can think of is probably self driving cars. Other simpler autonomous robots have already made their way into our homes. The "Christmas present of the year" in Sweden 2015 was awarded to the robot vacuum cleaner [13], proving that most people find robots not only useful but also fascinating.

The goal of this thesis is to make a mobile robot autonomous where the navigation is based only on information from a single camera. The idea is to try and exploit that cameras are cheap and easy to come by.

It is assumed that the robot will move on a planar surface. This might seem limited but while 3D movements are in some sense more desirable, planar movements are common in indoor environments. Given this assumption, other methods may be used for navigation and some of the uncertainty which 3D motion allows can be eliminated [1].

1.2 Basic idea

This thesis is based on earlier work by Wadenbäck and Heyden [1] and Brange [2]. Wadenbäck and Heyden considered navigation of a mobile robot equipped with a single rigidly mounted camera, directed towards the floor. This setup means that the camera will move in a plane at a constant height above the floor. It allows for estimating the camera motion based on planar homographies given by images of the floor.

Brange used this theory for building a navigation map. The input to his algorithm is a set of pairwise partly overlapping images taken by the camera on the robot. The images are stitched together, forming a larger image that describes the traversable area of the floor. In practice, the navigation map is made up of image features and not full images.

In this project a mobile robot will be equipped with a camera. Images of a floor will be used for building a navigation map using Brange's algorithm. The ultimate goal is to make the robot autonomous. The first step will be to find where the robot is located within the navigation map. The pose can be found by taking an image of the floor and matching image features to the features of the prebuilt map.

Once the pose is known, moving the robot to a desired position can be done in various ways. This thesis will be centered around making a robust, fast and reliable pose estimation. Therefore, the actual driving will be made in a fairly simplistic way. This will still demonstrate the performance and usefulness of the pose estimation.

Chapter 2

Problem formulation

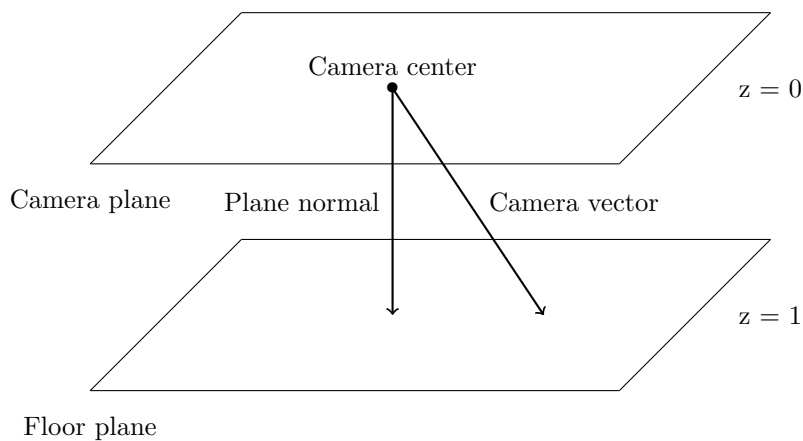


Figure 2.1: The camera moves freely in the plane $z = 0$ and is facing downward at the floor plane located in $z = 1$.

A mobile robot is equipped with a single downward looking camera which is described by the standard pinhole perspective camera model. The camera is rigidly mounted onto the robot and will therefore be at a constant height above the floor, in the plane defined as $z = 0$, see Figure 2.1. The floor will be at a constant distance from the camera and is assumed to be in the plane $z = 1$. This is not a limitation but only defines the scale of the global coordinate system [1].

The camera viewing direction, or camera vector, will not be parallel to the plane normal. It is unavoidable to mount the camera without some undesirable tilt. To describe the direction of the camera three rotations are used; \mathbf{R}_ψ , \mathbf{R}_θ and

\mathbf{R}_ϕ . The rotation \mathbf{R}_ϕ is the rotation about the fixed z-axis, see Figure 2.1. The rotations \mathbf{R}_ψ and \mathbf{R}_θ describe the unwanted rotations about the fixed x- and y-axis respectively. The camera tilt is defined as $\mathbf{R}_{\psi\theta} = \mathbf{R}_\psi\mathbf{R}_\theta$. Since the camera is assumed to be rigidly mounted to the robot, this tilt will be the same for all images.

The three remaining degrees of freedom, see Figure 2.2, the rotation about the z-axis \mathbf{R}_ϕ and the camera center position $\mathbf{t} = (t_x, t_y)$ in the plane $z = 0$ are the variable parameters of the camera and in extension also of the robot. The camera orientation is estimated at any given time by comparing images from the camera to a prebuilt navigation map.

The self driving capabilities will be analyzed by trying to reach a desired camera position given an estimated current position \mathbf{t} and rotation \mathbf{R}_ϕ .

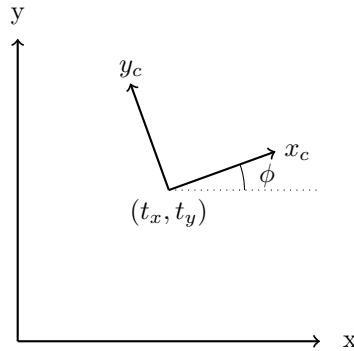


Figure 2.2: The camera position $\mathbf{t} = (t_x, t_y)$ and rotation angle ϕ described in the plane $z = 0$. (x_c, y_c) is the local coordinate system of the camera.

Chapter 3

Camera and image theory

3.1 Pinhole perspective camera model

The most commonly used mathematical model of a camera is the pinhole camera model. It is inspired by the the very simplest cameras. The ideal physical camera model, shown to the left in Figure 3.1 [19], has the shape of a box with a small hole, the *pinhole*. Light from an object passes through the hole and forms an image on the back wall of the box where the image sensor is assumed to be located.

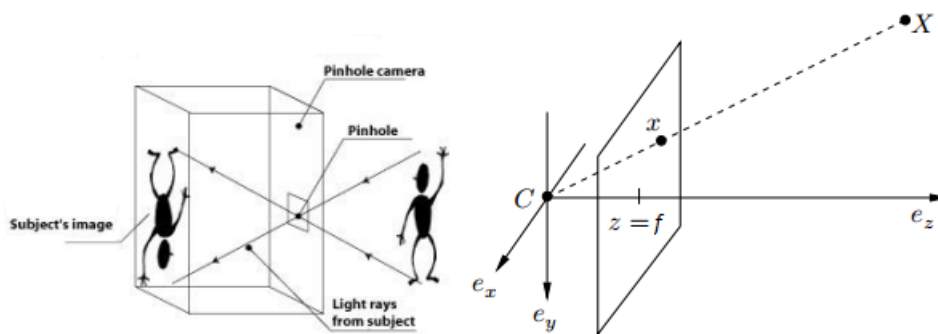


Figure 3.1: Left: Physical model of a pinhole camera.
Right: Mathematical model.

To make a mathematical model, an orthonormal coordinate system (e_x, e_y, e_z) is introduced at the pinhole with e_z pointing in the viewing direction of the camera. This is the camera coordinate system and places the camera center \mathbf{C} at the origin

$$\mathbf{C} = (0, 0, 0) \tag{3.1}$$

The image created on the back wall is horizontally and vertically flipped compared to the real object. To mathematically undo the inverting is equivalent to virtually moving the image sensor to the front of the camera, to what is called the *image plane*. The distance between the pinhole and the image plane is the *focal length* f . The ray from the camera center that intersects the image plane perpendicular to the plane (e_z in Figure 3.1) is called the *optical axis* and the intersection point is called the *principal point*. [1]

To generate a projection $\mathbf{x} = (x_1, x_2, f)$ in the image plane, of a scene point $\mathbf{X} = (X_1, X_2, X_3)$, a line from \mathbf{X} to \mathbf{C} intersecting the image plane is generated. The line intersects the image plane in $z = f$. Similarity of triangles gives that

$$\begin{aligned} \frac{x_1}{f} &= \frac{X_1}{X_3} \\ \frac{x_2}{f} &= \frac{X_2}{X_3} \end{aligned} \tag{3.2}$$

or equivalently

$$\mathbf{x}_{ip} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f \frac{X_1}{X_3} \\ f \frac{X_2}{X_3} \end{pmatrix} \tag{3.3}$$

where \mathbf{x}_{ip} is the (x, y) -coordinates of the projection in the image plane.

In the pinhole model the image plane is located in \mathbb{R}^3 . This means that the image projections are given by (x, y, z) -coordinates while image coordinates are given by (x, y) -coordinates. The image coordinate $(0, 0)$ is located in the center of the image. Most images have a coordinate system with $(0, 0)$ located at the top left corner of the image. Changing to pixel coordinates means scaling the x-coordinate with a factor d_x and the y-coordinate with a factor d_y . In the optimal case $d_x = d_y$. However, the sensor pixels are not always completely square. Defining $f_x = d_x f$ and $f_y = d_y f$ the mapping to pixels is given by

$$\begin{aligned} x_1 &= f_x \frac{X_1}{X_3} + c_x \\ x_2 &= f_y \frac{X_2}{X_3} + c_y \end{aligned} \tag{3.4}$$

where (c_x, c_y) is the pixel coordinate of the principal point. This point describes where the principal axis (e_z in Figure 3.1) intersects the image plane [1]. Optimally, this is in the image center but physical properties of the camera may offset this. Multiplying both sides of (3.4) by X_3 , this can be expressed using matrices as

$$\begin{pmatrix} x_1 X_3 \\ x_2 X_3 \\ X_3 \end{pmatrix} = X_3 \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} \quad (3.5)$$

where

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.6)$$

is the camera calibration matrix that transforms scene point coordinates to an image coordinate system with units in pixels. This invertible matrix contains the *intrinsic* (or *inner*) parameters of the camera.

Often it is advantageous to work in a global coordinate system which is not necessarily aligned with the camera coordinate system. Assuming the camera has the coordinates $\mathbf{t} = (t_x, t_y, t_z)$ in the global coordinate system and is rotated by a rotation matrix \mathbf{R} the mapping from global coordinates (X, Y, Z) to pixels (x, y) is given by [1]

$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{KR} [\mathbf{I} \mid -\mathbf{t}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (3.7)$$

The parameter λ describes the depth of the image points and the 3×4 matrix

$$\mathbf{P} = \mathbf{KR} [\mathbf{I} \mid -\mathbf{t}] \quad (3.8)$$

is called the *camera projection matrix*.

3.2 Homography

In the pinhole camera model, two images of the same planar surface are related by a *homography*. It is a transformation that may be used for transforming points in one image into another. Every homography can be represented by a non singular 3×3 matrix which is determined up to scale.

3.3 Features and feature extraction

In image processing and computer vision a *feature* is an especially important piece of information in an image. What is considered important is determined by the algorithm used for finding the feature. Some commonly used algorithms, such as the Harris corner detector and the Canny edge detector, find corners and edges in an image. These features are easy to understand. There are, however, a lot more abstract features.

The process of finding features in an image is known as *feature extraction*. Depending on the algorithm this process may vary widely. Still, the resulting feature is normally expressed in a similar way. A feature generally consists of a *feature point* describing the location of the feature in the image in x-y coordinates and a *descriptor* (or *feature vector*), a vector describing the extracted information. Naturally, the size and values of this vector varies since it describes information specific to the algorithm used.

An important property of a feature extracting algorithm is its ability to find the same features under different image conditions. It should find features which are invariant to rotation, translation, and illumination. Also, the descriptors have to be distinctive and at the same time robust to noise [6].

3.4 Feature matching

Extracting features from two images results in two feature sets. Given that these images are partially overlapping some corresponding features should be present in the two sets. Finding and pairing features together is known as *feature matching*. The matching is normally done by comparing distances, e.g., Euclidean distances, between the descriptors [6]. The dimension of the descriptor vectors therefore has a direct impact on the matching speed. A good feature extractor has to compromise between a long descriptor for distinctiveness and a short for matching speed.

3.5 RANSAC

The main idea for matching features is fairly straight forward. However, some of the matches will be false matches and must be removed. The *random sample consensus* (RANSAC) algorithm proposed by Fischler and Bolles [3] is a method that may be used to remove these outliers caused by false matches. RANSAC is a resampling method that generates candidate solutions by using the least amount of data points needed to estimate the underlying model parameters. The main outline of the algorithm is as follows [4]

1. Randomly select a smallest possible subset sufficient to determine the model parameters.
2. Solve the problem using this subset.
3. Evaluate the *error residuals* for the remaining measurements given the above solution.
4. The *consensus set* for this solution is the set of measurements with error residuals less than some predefined threshold.
5. Repeat 1-4 a number of times and at the end select the solution that gives the largest consensus set.

There are two things that must be predetermined, the number of iterations and the threshold for the residuals. The higher the iteration count, the more likely it is to find a completely outlier-free set. If the ratio of inliers to outliers is known, usually the iteration count is chosen so that the probability of finding an outlier-free set is 0.99 [4]. The error threshold depends completely on the problem being solved.

3.6 Image registration

Image registration is the process of transforming image coordinates from one image to another. Given two matching feature sets, the matching feature points are defined as \mathbf{x}_i and \mathbf{y}_i so that $i = 1, 2, \dots, n$, where n is the size of the sets. The images in this thesis are taken with the same camera at the same distance to the floor. This means the scale is the same between the images and the matching points are related with only a rotation and a translation. Thus, the transformation can be described as a rigid transformation.

A rigid transformation in \mathbb{R}^2 includes a rotation ϕ and a translation $\mathbf{t} = (t_x, t_y)$ so that the transformation from image point \mathbf{x}_i to corresponding image point \mathbf{y}_i is given by

$$\mathbf{y}_i = \mathbf{R}\mathbf{x}_i + \mathbf{t} \tag{3.9}$$

where \mathbf{R} is a rotation matrix

$$\mathbf{R} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \tag{3.10}$$

and fulfills $\mathbf{R}\mathbf{R}^T = I$ and $|\mathbf{R}| = 1$. This can also be expressed with a homography as

$$\mathbf{y}_i = \mathbf{H}\mathbf{x}_i \tag{3.11}$$

where

$$\mathbf{H} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \quad (3.12)$$

The image registration problem using rigid transformations is called *orthogonal Procrustes*. It means finding the best rotation \mathbf{R} and translation \mathbf{t} between the two point sets. This can be expressed as a minimization problem where the goal is to find the \mathbf{R} and \mathbf{t} that minimizes the *reprojection errors* $\mathbf{y}_i - (\mathbf{R}\mathbf{x}_i + \mathbf{t})$ for all matches i as

$$\min_{\mathbf{R}, \mathbf{t}} \sum_{i=1}^n \|\mathbf{y}_i - (\mathbf{R}\mathbf{x}_i + \mathbf{t})\|^2 \quad (3.13)$$

The reprojection error can be described as the distance between the transformed \mathbf{x}_i and the corresponding matching \mathbf{y}_i . The least squares fitting algorithm suggested by Arun et al. [8] solves this minimization problem. The algorithm is described below.

Find the centroids $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ of the respective sets by

$$\begin{aligned} \bar{\mathbf{x}} &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \\ \bar{\mathbf{y}} &= \frac{1}{n} \sum_{i=1}^n \mathbf{y}_i \end{aligned} \quad (3.14)$$

Define new coordinates as

$$\begin{aligned} \tilde{\mathbf{x}}_i &= \mathbf{x}_i - \bar{\mathbf{x}} \\ \tilde{\mathbf{y}}_i &= \mathbf{y}_i - \bar{\mathbf{y}} \end{aligned} \quad (3.15)$$

Let \mathbf{C} be the covariance matrix defined as

$$\mathbf{C} = \sum_{i=1}^n \mathbf{x}_i \mathbf{y}_i^T \quad (3.16)$$

The single value decomposition of \mathbf{C} is given by

$$\mathbf{C} = \mathbf{U} \mathbf{D} \mathbf{V}^T \quad (3.17)$$

The solution to the minimization problem is then the rotation and translation given by

$$\mathbf{R} = \mathbf{U} \mathit{diag}(1, \det(\mathbf{UV}^T)) \mathbf{V}^T \quad (3.18)$$

$$\mathbf{t} = \bar{\mathbf{y}} - \mathbf{R}\bar{\mathbf{x}} \quad (3.19)$$

Chapter 4

Hardware and software

This chapter will introduce the robot, the camera and the software used for this thesis. A discussion on why a certain equipment was used and in some cases a suggestion on other alternatives is given.

4.1 Robot

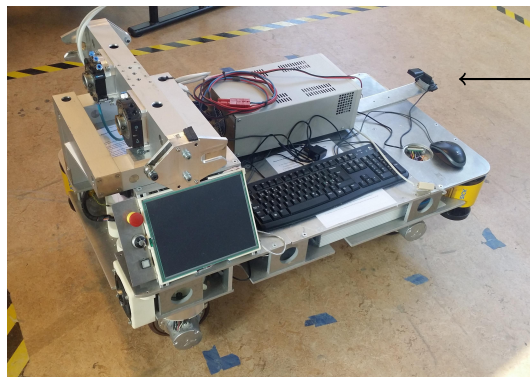


Figure 4.1: Image of the mobile robot rob@work3 used for experiments. The arrow marks the camera.

The robot, seen in Figure 4.1, is a mobile robot called rob@work3 manufactured by Fraunhofer IPA [18]. At the control department it is called Sleipner after the Norse mythological horse of Odin. Sleipner is said to have been an eight legged horse able to travel by land, sea and even able to take flight. Sadly, the

robot did not get its name because of its ability to fly. Instead, it comes from the eight motors controlling the four wheels.

Each wheel can be controlled independently. The two motors for each wheel make it possible to both roll the wheel, like you would expect of any wheel, but also to change the heading angle to go in any direction.

At the start of this project it was not obvious that this would be the robot of choice. The much smaller TurtleBot [16] was also considered. As was mentioned in the problem formulation (Chapter 2), and as will be emphasized more later, it is important that the robot is stable. It was found that the TurtleBot would jiggle back and forth after a sudden deacceleration. This caused the camera to have a varying distance to the floor. Sleipner, on the other hand, is steady and much more predictable.

4.2 Camera

The choice of camera was made rather casually. Simply, a convenient and easy to reach USB camera was chosen. This was not careless, however. One important part of this thesis is to investigate how well a cheap camera will perform. The camera used for experiments ended up to be a Plexgear 720p USB webcam [20], seen in Figure 4.2.



Figure 4.2: The Plexgear 720p camera used for experiments.

Another AXIS camera was considered. This is by all means a better camera but it has two big disadvantages. It costs a lot more than the USB camera and it is much more clunky to use. It requires a network cable and a router that needs power.

The camera was mounted on a metallic plate and screwed onto the robot. It is important that the camera can be assumed to be mounted rigidly. The mounted camera is marked by the arrow in Figure 4.1.

4.3 ROS

The *Robot Operation System* (ROS) is a framework for writing and configuring robot software. It includes libraries and tools for use on almost any robotic platform. This general purpose software is open source and is compatible with both C++ and python [15]. This had led to ROS being widely used within a big community. Online guides and support from the community are important assets when working with ROS.

Since ROS was already the main program installed on the robot, it was natural to use it for this project. Creating a robot model, constructing regulators for controlling the robot hardware and integrating this with ROS are full-blown projects by themselves. Having this already set up was not only convenient but also crucial for this project. Given this setup, moving the robot in a desired direction can be done by sending commands to move with a desired velocity along a certain axis. ROS also allows for easy communication with the robot. Publishing or reading information locally on the robot or on the network to another computer is just as easy.

Some programs, however, had to be implemented for ROS. An executable program in ROS is called a *node*. This means a node contains a main function that interacts with the ROS library in a certain way. Three nodes have been implemented. They can be seen, together with a node already existing on the robot, in Figure 4.3.

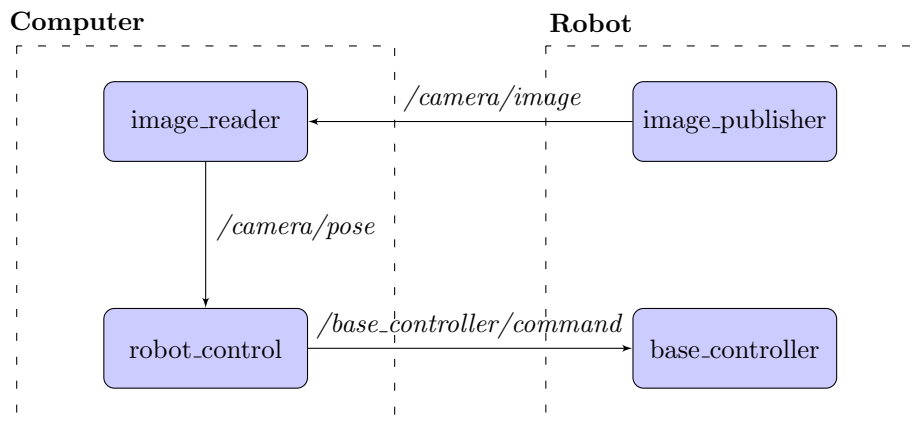


Figure 4.3: Flowchart of the information flow between the different ROS nodes. The node at the arrow base publishes information, and the node at the arrow head reads information, from the topic named near the arrow.

This first node, `image_publisher.cpp`, runs on the robot computer and is responsible for capturing images. It reads the camera stream from USB, repackages the images into a format fit for use in ROS and publishes this image to a ROS *topic*

called */camera/image*. Topics are basically buses over which nodes exchange messages. In other words, the image is made available on the network.

The second node, *image_reader.cpp*, is run on a separate computer. This computer has more computational power than the robot computer and is also not bothered by the robot control. This node is the main node responsible for estimating the camera pose, making use of the image analysis described throughout this thesis. As the name suggests, it also handles the ROS communication for reading images over the network by reading from the topic */camera/image*. Finally, it publishes the estimated camera pose to another topic, */camera/pose*.

The third node, *robot_control.cpp*, is also run on the separate computer and is responsible for determining the robot movements. It reads the current camera pose from the */camera/pose* topic and then determines the robot velocities. These are published to the topic */base_controller/command*. The node *base_controller* (not implemented for this thesis) on the robot reads from the same topic and uses the information to control the robot.

4.4 OpenCV

Open Source Computer Vision (OpenCV) is a collection of image analysis and computer vision interfaces that supports, among other languages, C++. With an estimated number of over nine million downloads, it has become a standard for developing image analysis software [14]. OpenCV is especially designed to be used for real time image processing which suits this project's needs well. All the image analysis that is run online, while the robot is controlled, is made in OpenCV. Both ROS and OpenCV have support for C++ which therefore ended up to be the code language used.

4.5 Matlab

Matlab with the Image Processing Toolbox is used for building the navigation map [2]. The high level language of Matlab was convenient in the early stages of this project since it was easy to test robot images against the generated map. Therefore, some code was initially implemented in Matlab. Later on, to avoid implementing code twice, some Matlab functions were code-generated with the Matlab plugin Matlab Coder. The Coder outputs optimized and fast C/C++ code, often much faster than the initial Matlab code [17].

Chapter 5

Extracting image data

This chapter will describe how to process the raw image from the USB camera and the steps needed to extract useful features. The chapter will include image preprocessing, different distortion effects and techniques to counteract these effects.

5.1 Floor

The downward looking camera sees what is currently underneath the robot. This application is mainly meant to be used indoors so that means the camera will most likely see a floor. This means there are some requirements on the floor texture. A completely uniformly colored floor with no texture holds very little information. It would be impossible to tell a certain slice of the floor apart from another slice. However, this is not the case for most floors. A wooden floor have a lot of texture. A repetitive tiled floor normally have variance among tiles. The experiments in this project are made on a textured plastic floor which proves sufficient. Figure 5.1 shows a typical image used for experiments. This is the raw three color channel image taken by the USB camera, stored as a 640x480 pixel image.



Figure 5.1: A typical robot view of the floor used for experiments.

5.2 Image preprocessing

The raw images from the USB camera are preprocessed to significantly increase the amount of information that may be extracted from them. Firstly, the image is transformed to a grayscale image. This is required since the feature extractor (Section 5.7.1) only considers grayscale images. This results in Figure 5.2.

To increase the global contrast, the image is histogram equalized. This will bring out the texture of the floor. As can be seen in Figure 5.3, a lot more details are now evident.



Figure 5.2: Robot view image transformed to grayscale image.



Figure 5.3: Robot view histogram equalized.

5.3 Camera distortion

Most cameras, and this is especially true for cheap ones, will be affected by significant distortion. The distortion will be the same between images and will therefore only need to be estimated once for each camera. With known distortion, image points may be remapped to compensate for the defects.

It is common to take two different distortions into account [12]; radial and tangential. The radial part is caused by the camera lens. The camera used in this project suffered from the radial distortion called *barrel* (or *fish-eye*) distortion. It causes the image to look like it has been wrapped around a spherical object. Figure 5.4 [21] shows an example of how an image of parallel lines are affected by barrel distortion.

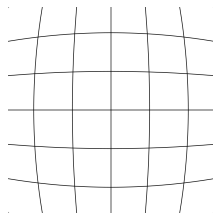


Figure 5.4: Barrel distortion applied to an image of parallel lines.

To compensate for the radial distortion an image point (x, y) may be corrected to (x_c, y_c) by the model [12]

$$\begin{aligned} x_c &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\ y_c &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) \end{aligned} \tag{5.1}$$

The tangential distortion occurs when the camera lens is not perfectly parallel to the imaging plane. These defects can be compensated for by the model [12]

$$\begin{aligned}x_c &= x + (2p_1xy + p_2(r^2 + 2x^2)) \\y_c &= y + (p_1(r^2 + 2y^2) + 2p_2xy)\end{aligned}\tag{5.2}$$

where $r = x^2 + y^2$.

These two corrections result in five unknown distortion parameters; k_1 , k_2 , k_3 , p_1 and p_2 .

5.4 Camera calibration

If a camera is to be used for taking measurements in the real world the camera must be calibrated. In the classic pinhole perspective camera model this corresponds to finding the intrinsic parameters in the camera matrix \mathbf{K} (see also Section 3.1).

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}\tag{5.3}$$

5.5 Chessboard camera calibration

There are different methods for estimating the *calibration parameters* (intrinsic and distortion parameters). In this project the classical chessboard detection technique is used for estimating both types of parameters. Two typical images for this calibration method can be seen in Figure 5.5.

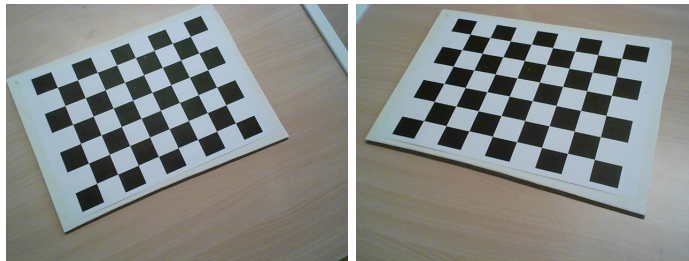


Figure 5.5: Two typical chessboard images.

The reason a chessboard pattern is used is because it is easy to recognize. It has sharp corners and distinctive edges. Normally, somewhere between ten and twenty images of the board are taken from various angles. The algorithm identifies the board in each image. The key elements of the algorithm are that the chessboard is known to be a flat surface and the amount of squares and the square side lengths are also known. This information is used in the algorithm for generating the calibration parameters. This technique is commonly used and is therefore implemented in many different image processing toolboxes. For this project an OpenCV implementation was used. A detailed explanation of the algorithm is left out in this thesis but may be found in [5].

After finding the calibration parameters they are assumed to be constant and never change. This is reasonable for the distortion parameters since they are based on the physical properties of the camera. The camera used for experiments has auto-focus, but to keep the intrinsic parameters constant the camera must not refocus. As long as the robot is stable and is traversing a flat surface this should not happen randomly and is therefore a reasonable assumption too.

$$\mathbf{K} = \begin{pmatrix} 790.60 & 0 & 310.58 \\ 0 & 791.90 & 230.79 \\ 0 & 0 & 1 \end{pmatrix} \quad (5.4)$$

Table 5.1: Estimated distortion parameters.

Variable name	Estimated value
k_1	$3.952 \cdot 10^{-2}$
k_2	1.3204
k_3	-6.006
p_1	$7.429 \cdot 10^{-3}$
p_2	$-1.188 \cdot 10^{-3}$

Using the chessboard method on the USB camera results in the intrinsic parameters that can be seen in (5.4), and the distortion parameters that can be seen in Table 5.1. Given these, the images may be warped to compensate for the distortion. If the histogram-equalized image seen in Figure 5.3 is warped with the estimated parameters the resulting image is shown in Figure 5.6. The distortion was not very big to begin with so the correction may be hard to notice. What can be seen is the dark areas along the edges of the image. They arise when the image is pressed together to counteract the barrel effect. They are set to black because of lack of information.



Figure 5.6: Robot view after distortion correction.

5.6 Camera tilt estimation

In the specific setup with a downward looking camera it is important that the images really are parallel to the floor. However, when mounting the camera it is unavoidable to not mount it slightly tilted. This will also affect the images with a distortion effect. It is assumed that the tilt is constant, and this is true if the camera is rigidly mounted to the robot. If the tilt remains the same between images it only needs to be estimated once. Just like with the barrel and tangential distortion, the measured image points can be rectified to counteract the tilt. The remaining part of this section will derive how to find this correction.

The camera projection matrix (see also (3.8)) is

$$\mathbf{P} = \mathbf{K}\mathbf{R} [\mathbf{I} \mid -\mathbf{t}] = \mathbf{K}\mathbf{R}_{\psi\theta}\mathbf{R}_{\phi} [\mathbf{I} \mid -\mathbf{t}] \quad (5.5)$$

where in the last equality the rotation matrix \mathbf{R} has been split into tilt $\mathbf{R}_{\psi\theta}$ (introduced in Chapter 2) and rotation about the z -axis \mathbf{R}_{ϕ} . Since every image is studied separately it can be assumed that the camera is located at the origin with no rotation around the z -axis. This means the matrix can be expressed as

$$\mathbf{P} = \mathbf{K}\mathbf{R}_{\psi\theta} [\mathbf{I} \mid \mathbf{0}] \quad (5.6)$$

To remove the distortion caused by the tilt a correction to the image points must be found, i.e., a homography \mathbf{H} to be applied to the image coordinates to make them look like they come from a camera \mathbf{P}_{\perp} with no tilt

$$\mathbf{P}_\perp = \mathbf{K} [\mathbf{I} \mid \mathbf{0}] \quad (5.7)$$

A point in the floor plane $z = 1$ in homogeneous coordinates $\mathbf{X} = (X, Y, 1, 1)$ is projected by the tilted camera \mathbf{P} in (5.6) as

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim \mathbf{P}\mathbf{X} = \mathbf{K}\mathbf{R}_{\psi\theta} (\mathbf{I} \mid \mathbf{0}) \begin{pmatrix} X \\ Y \\ 1 \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{R}_{\psi\theta} \begin{pmatrix} X \\ Y \\ 1 \\ 1 \end{pmatrix} \quad (5.8)$$

The same point \mathbf{X} is projected by the untilted camera \mathbf{P}_\perp in (5.7) as

$$\begin{pmatrix} x_\perp \\ y_\perp \\ 1 \end{pmatrix} \sim \mathbf{P}_\perp\mathbf{X} = \mathbf{K} (\mathbf{I} \mid \mathbf{0}) \begin{pmatrix} X \\ Y \\ 1 \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} X \\ Y \\ 1 \\ 1 \end{pmatrix} \quad (5.9)$$

Using (5.8) and (5.9) results in

$$\begin{pmatrix} x_\perp \\ y_\perp \\ 1 \end{pmatrix} \sim \mathbf{K} \begin{pmatrix} X \\ Y \\ 1 \\ 1 \end{pmatrix} \sim \mathbf{K}(\mathbf{K}\mathbf{R}_{\psi\theta})^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{R}_{\psi\theta}^T \mathbf{K}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (5.10)$$

This means that in order to rectify the image coordinates $\mathbf{x} = (x, y)$ the homography

$$\mathbf{H} = \mathbf{K}\mathbf{R}_{\psi\theta}^T \mathbf{K}^{-1} \quad (5.11)$$

must be applied to get the corrected coordinates $\mathbf{x}_c = (x_c, y_c)$ as

$$\begin{pmatrix} x_c \\ y_c \\ 1 \end{pmatrix} = \mathbf{K}\mathbf{R}_{\psi\theta}^T \mathbf{K}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (5.12)$$

where \mathbf{K} is the camera calibration matrix estimated in the previous section and the tilt matrix $\mathbf{R}_{\psi\theta}$ is

$$\mathbf{R}_{\psi\theta} = \mathbf{R}_\psi \mathbf{R}_\theta = \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ \sin \psi \sin \theta & \cos \psi & -\sin \psi \cos \theta \\ -\cos \psi \sin \theta & \sin \psi & \cos \psi \cos \theta \end{pmatrix} \quad (5.13)$$

Heyden and Wadenbäck [1] have shown that the undesired tilt angles ψ and θ can be estimated. For this project software developed by Wadenbäck is used for

the tilt estimation and no further details will be explained here. For a detailed explanation see [1].

The estimated tilt angles for the camera used in experiments can be seen in Table 5.2. The homography \mathbf{H} that may be applied to untilt the image points is shown in (5.14).

Table 5.2: Estimated camera tilt parameters.

Variable name	Estimated value (rad)
ψ	$-1.0716 \cdot 10^{-2}$
θ	$7.9801 \cdot 10^{-3}$

$$\mathbf{H} = \mathbf{K} \mathbf{R}_{\psi\theta}^T \mathbf{K}^{-1} = \begin{pmatrix} 1.003 & 4.117 \cdot 10^{-3} & -8.250 \\ 2.330 \cdot 10^{-3} & 1.003 & -9.938 \\ 1.009 \cdot 10^{-5} & 1.353 \cdot 10^{-5} & 0.9937 \end{pmatrix} \quad (5.14)$$

Considering the flow of the paper so far it would seem natural to include an image where the robot view has been corrected by the tilt. However, this image is never calculated. Instead, only the detected feature points are transformed. This makes the code much faster.

5.7 Feature extraction

Picking a suitable feature extractor for this thesis comes down to a few different criteria. Since the robot navigation is based on computer vision, the image analysis is made in real time. This puts a speed requirement on the feature extractor. The faster the extractor, the better. It is also important that the extracted features are in-plane rotation invariant. The robot may move freely within the map and therefore the matching must not depend on rotation.

5.7.1 SURF

Speeded-Up Robust Features (SURF) by Bay et al. [6] is a feature detector designed with the goal of being rotation-invariant, computationally fast but still give robust and distinctive descriptors. The algorithm has been successfully used for numerous topics, such as image registration and object recognition. Since SURF is widely used, both the feature extraction and matching have been implemented in many image analysis toolboxes and programs, including OpenCV. All of these reasons together have lead to SURF being the suitable choice for this project.

Extracting features from images results in feature points and descriptors. The feature points may be drawn on top of the image they were extracted from, at the location where they were extracted. Such an example can be seen in Figure 5.7 where the image used as source is the one in Figure 5.6.

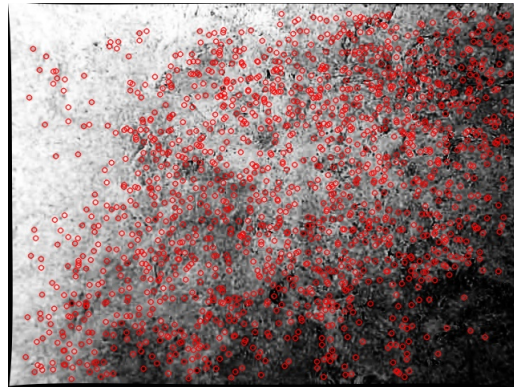


Figure 5.7: The location of feature points (red circles) on top of the undistorted robot view image.

Chapter 6

Finding the camera pose

The goal is to find a position and a rotation of the camera expressed in the map coordinate system. As described in the previous chapter, given an image, the feature points and descriptors are extracted. Assuming the camera is located somewhere in the map, most of these feature should also be present among the map features. Matching the image features with the map features results in matching pairs. Finding the pose of the camera comes down to finding the rigid transformation from the image feature points to the matched map points and using this rotation \mathbf{R} and translation \mathbf{t} to transform the camera position to a pose in the map.

6.1 Navigation map

The navigation map is very important to this project. During navigation this map describes the world to the robot. Brange's algorithm [2] builds the map from a set of pairwise partly overlapping images. In practice these images are acquired by manually driving the robot and taking images of the floor.

The algorithm extracts features from each image, match the features between images and stitch them together to form one large image of the floor. One such image may be seen in Figure 6.1. However, this full image is not used for the navigation itself. Instead the extracted features from each image are used. These make up the actual navigation map. The map used for experiments can be seen in Figure 6.2. The map consists of 32,640 features.

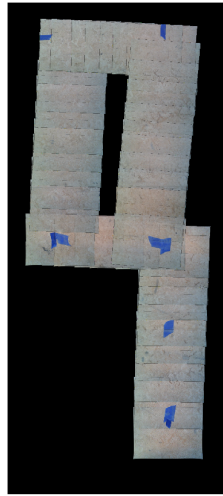


Figure 6.1: 40 images stitched together to form one large image of the traversable floor.

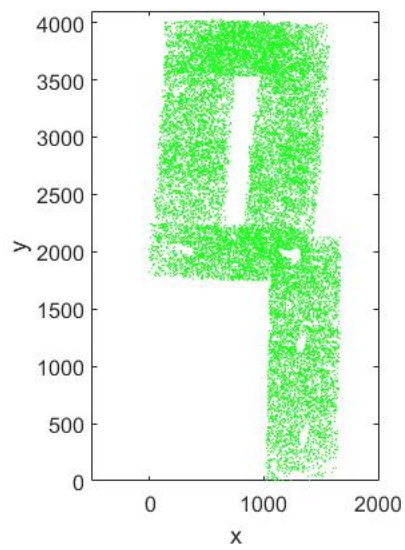


Figure 6.2: Real navigation map made up of 32,640 feature points, plotted as green crosses.

6.2 Removing bad matches

When the image points from an image is to be matched against the navigation (Section 3.5) some of the matches will be false matches. A RANSAC solution was suggested to removed the outliers. Before this is done, however, another more crude method is used for removing some of the false matches.

The distance d between two matched feature vectors \mathbf{p} , \mathbf{q} of length k , can be described as the Euclidean distance between the vectors as

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^k (q_i - p_i)^2} \quad (6.1)$$

This serves as a similarity measure of the vectors. The more similar the vectors, the smaller the distance. Given a set of matches, define the smallest distance d_{min} as the smallest distance of all the matches in the set. Also define d_{min}^* as

$$d_{min}^* = \max(d_{min}, 0.02) \quad (6.2)$$

The crude method for eliminating some false matches is to eliminate all matches where

$$d > 2d_{min}^* \quad (6.3)$$

This idea is to remove all matches that are considered too different. The reason d_{min}^* is used is to not limit the minimum distance too much. If one match is particularly good too many other matches will be removed if d_{min} is used. The threshold of 0.02 in (6.2) and the multiplier of 2 in (6.3) is chosen to give a decent amount of remaining matches each time. Notice that this method will also remove some true matches but the strongest ones will remain.

6.3 Robust image registration with RANSAC

Even if some bad matches were removed in the previous section, some will still be there. They will have a large negative effect on the estimated rotation and translation found by solving the image registration problem, (3.18) and (3.19). To eliminate the false matches, RANSAC (Section 3.5) is used in conjunction with the least squares fitting algorithm (Section 3.6). The combination is as follows.

1. At every iteration select two random matching pairs from the feature points. Only two point correspondences are needed to estimate a rigid transformation.
2. Find \mathbf{R}^* and \mathbf{t}^* with the least squares fitting algorithm given these two point pairs.
3. Rigidly transform all image feature points with \mathbf{R}^* and \mathbf{t}^* .
4. If a transformed point is closer than the error-threshold to the corresponding map point, count as inlier. Otherwise as outlier.
5. Select the rotation \mathbf{R} and translation \mathbf{t} that gives the most inliers.

6.4 The camera pose

The rotation and translation are found by finding the best fit between matching feature points. However, this rotation and translation holds true for all pixels in the image. They may all be transformed to find their position in the navigation map. More specifically, this is also true for the principal point (c_x, c_y) which can be seen as the location of the camera center within the image coordinate system. This gives the camera pose (x_c, y_c, ϕ_c) in the map as

$$(x_c, y_c) = (c_x, c_y) + \mathbf{t} \tag{6.4}$$

and the rotation angle ϕ_c can be found in the rotation matrix \mathbf{R} .

6.5 Speeded-up matching

The matching of image features to map features can be done in different ways. The most robust but also the slowest method is to use a brute force matching algorithm. This means that every image feature is matched against every feature in the map and the best match is chosen. The best match is the match with the lowest Euclidean distance between the feature descriptors. It was noticed that the biggest bottle neck of the code was the matching. The method means that the matching speed is very heavily dependent on the amount of features both in the map and in the images.

The method described above assumes that nothing is known about the camera position and to find the position the whole map must be searched. This is only true at the very first iteration. To speed up the matching the old estimated position within the map can be taken into account. Assuming the position at the last iteration is known, only features close to this point needs to be considered for matching. A simple idea is to only match the image features to map features shorter than a distance l from the old position.

The problem is now to find which features are closer than l from the old point. This is done in a similar brute force way as before. The distance from the old camera position to every map feature is measured. Only those closer than l are selected. This still requires a full pass through of the map features. However, this is still a great reduction from matching every single image point, usually somewhere 900 - 1500 points per image, which would require equally many full passes through all the map features.

6.5.1 Incorrect estimates

Depending on the old position for the matching is not always a good thing. Sometimes the estimated position will be wrong. Assume a worse case scenario where the old position is estimated so far from the correct one, that at the next iteration the whole circle with radius l is far away from the current true position (which has not been estimated yet). This could mean that none of the features inside l are present in the new image. This would result in either a new terrible estimation or a failed estimation. In case of a really bad estimation, which sometimes occur from noise or distortion effects, the whole matching algorithm may fail.

To safeguard against this kind of behavior a new distance m may be introduced. This variable is the maximum allowed distance that the camera positions may vary between estimations. If the distance is larger than this, the estimation is discarded and the system is reset so that the next estimation is found by comparing to the whole map.

Chapter 7

Robot control

Once the camera pose is known from estimation, it can be used for controlling the robot. This can be done in many different ways. This chapter will describe how the robot uses the camera pose to reach a desired goal position, by following a path described by minor goals, called waypoints. The method presented will be quite simplistic but will still be sufficient to investigate the precision and fitness for use of the pose estimator. For a detailed model of the dynamics and kinematics of the robot, see [7].

7.1 Problem definition

Assume the camera is placed at a known position $\mathbf{t} = (t_x, t_y)$ with a known rotation ϕ in the global map coordinate system. The camera is to reach a certain goal position where the path to the goal is described by discrete waypoints given as x-y coordinates. The difference in angle between the current pose and the next waypoint is $\delta\phi$ and the distance to the waypoint is δd , see Figure 7.1.

The final orientation ϕ of the camera at each waypoint is not considered. That means, how and from which direction each waypoint is reached is not considered important.

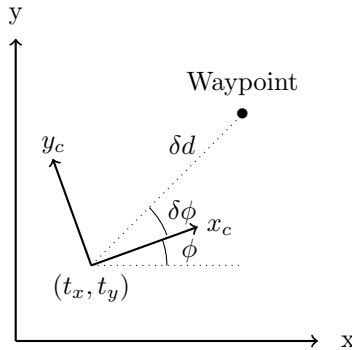


Figure 7.1: The estimated camera pose and the next waypoint placed in the global map coordinate system. The angle between them is $\delta\phi$ and the distance between them is δd .

7.2 The camera movement algorithm

1. Find the distance between the current position and the position of the next waypoint δd . If δd is smaller than a given threshold ϵ_D :
 - (a) if the waypoint is the final goal, terminate the algorithm and consider the goal reached.
 - (b) else select the next waypoint in the path.
2. Find the difference in angle $\delta\phi$. If $\delta\phi$ is larger than a given threshold ϵ_ϕ :
 - (a) rotate around the camera z-axis.
 - (b) else the angle is within the threshold, go to 3.
3. The camera is facing the next waypoint. Move forward and return to 1.

7.3 The robot movements

As stated in the previous section under note 2 (a), it is desired to rotate around the camera. This will keep the camera fixed in the map coordinate system while the ϕ angle changes. The robot used for experiments is controlled by setting reference velocities in the x-, y- and ϕ -directions. These are, naturally, specified in the robot coordinate system. In probably all practical applications the camera and robot coordinate systems will not align. However, since the camera is rigidly mounted to the robot there will be a rigid transformation between the two coordinate systems. Below will follow a way to derive the

velocity references that should be applied to the robot in order to rotate around the camera.

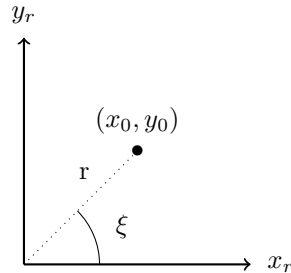


Figure 7.2: The camera position $\mathbf{x}_0 = (x_0, y_0)$ in the robot coordinate system where r is the distance from the origin and ξ is the angle from the x-axis.

Assume that the camera is placed a distance r and an angle ξ from the origin in the robot coordinate system, see Figure 7.2. Further assume the camera and robot z-axis are pointing in the same direction. This means that the camera position in the robot coordinate system $\mathbf{x}_0 = (x_0, y_0)$ can be expressed as

$$\begin{aligned} x_0 &= r \cos(\xi) \\ y_0 &= r \sin(\xi) \end{aligned} \tag{7.1}$$

The velocities are

$$\begin{aligned} \dot{x}_0 &= -r \sin(\xi) \dot{\xi} \\ \dot{y}_0 &= r \cos(\xi) \dot{\xi} \end{aligned} \tag{7.2}$$

This means that to generate a rotation about the z-axis of the camera, the velocity references to send to the robot are $(\dot{x}_0, \dot{y}_0, \dot{\xi})$ where $\dot{\xi}$ is some free to choose rotational velocity.

7.3.1 Finding the relative camera position

To do this in practice, the distance r and rotation ξ must be determined. This can be done in various ways. For the robot used it was fairly straight forward to take measurements of the wheel positions. Their center proved to be in the robot coordinate system origin. Measuring the distance from the center point to the camera gave r . Similar measurements also gave ξ .

Chapter 8

Results

For all experiments the navigation map seen in Figure 8.1 is used. Notice that this is the same map as was shown in Figure 6.2, only rotated 90 degrees and plotted in different software. This map consists of 32,640 SURF features where the descriptors are of size 64. The map is built from 40 pairwise overlapping images, acquired by manually driving the robot. The floor is the plastic floor of the room called *Robotlab* located at the control department at Lund University.

The image height is known to be 480 pixels. The height of the image is measured to be about 19 cm. That means the real world length per pixel is roughly 0.04 cm/pixel. The map is 4023 pixels long and 1667 pixels wide, see Figure 6.2. This means the map size is about 1.6 m long and 0.7 m wide.

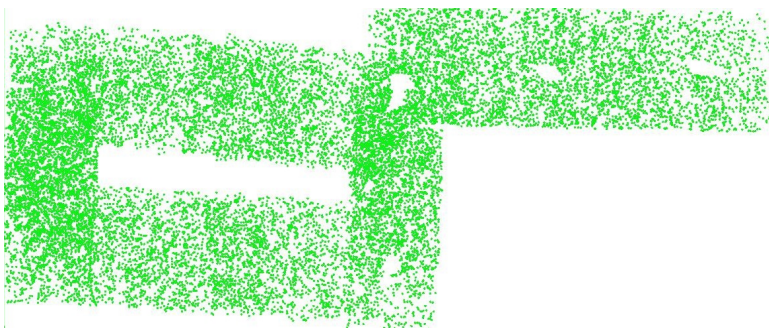


Figure 8.1: Navigation map used for experiments (green circles), made up of 32,640 SURF features.

8.1 Matches and pose estimation

To analyze the feature matching and the pose estimation the robot is placed in the map. Data from a specific location and for one specific estimation is as follows.

- The amount of features extracted from the camera image is 913.
- The amount of features in the map is 32,640.
- Feature matching and reduction of features by removing bad matches (Section 6.2) results in 45 corresponding matches.

The remaining 45 feature correspondences are used for estimating the rotation \mathbf{R} and translation \mathbf{t} (Section 6.3). If the image feature points are transformed with \mathbf{R} and \mathbf{t} and placed within the map, the result can be seen in Figure 8.2. The map is heavily zoomed in to be able to see something of interest. This means only a small slice of the full map and only a few of the 45 image features are shown in Figure 8.2. The green circles are the map feature points and the red circles are the transformed image feature points.

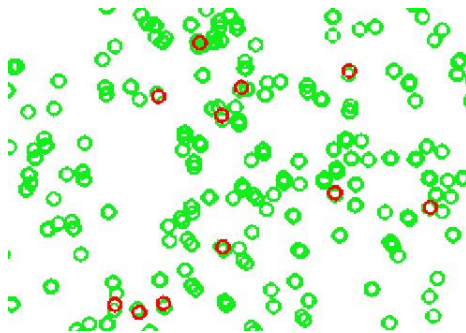


Figure 8.2: A slice of the full navigation map (green circles) and the matched and transformed image feature points (red circles).

8.2 Speeded-up matching

To analyze the speeded-up matching (Section 6.5) the average sampling frequency, the frequency at which images are acquired and camera positions are estimated, is measured with and without the optimization. The experiments are made by placing the robot within the map and keeping it still. Apart from turning the optimization on and off, everything else in the algorithm is kept the same.

An example of when the speeded-up algorithm is used is illustrated in Figure 8.3. The estimated camera position at a certain time as a small yellow circle,

the previous estimated position as a small blue circle and the circle, generated by choosing $l = 300$ map units, as a large blue circle. Since the robot is kept at a fixed position, the current and previous positions are overlapping and the small yellow circle is barely visible in this image.

For $l = 300$ the amount of features to consider for matching (those inside the large blue circle) are reduced from 32,640 of the whole map to 2,407.

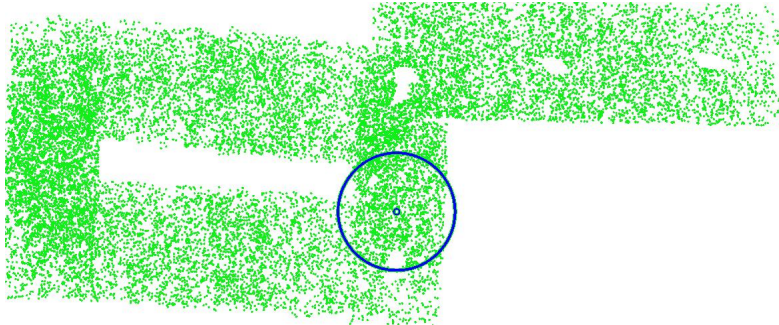


Figure 8.3: Navigation map features (green), current camera position (small yellow circle), old camera position (small blue circle) and the edge of the area used for feature search in the map (large blue circle with radius $l = 300$ map units).

At this certain location the average sampling frequency is measured with and without the speeded-up matching algorithm.

- Without, the average sampling frequency is 0.7 Hz.
- With, the average sampling frequency is 5.2 Hz.

This is an increase of

$$\frac{5.2Hz}{0.7Hz} = 7.4 \tag{8.1}$$

8.3 Motion blur

It was noticed that the images were affected by heavy motion blur while the camera was moving. To illustrate the effects, images of the same piece of warning tape was captured while driving at different velocities. Four such images are shown in Figure 8.4.



(a) Image while standing still.



(b) Image while moving at 0.1 m/s.



(c) Image while moving at 0.2 m/s.



(d) Image while moving at 0.3 m/s.

Figure 8.4: Images affected by motion blur taken at different velocities. Notice that the velocities are given as rough estimates.

8.4 Robot movements

The robot is placed somewhere in the map. Given four waypoints the camera movement algorithm (Section 7.2) is run. While driving, the estimated camera positions are stored. The estimated camera movements are plotted as the blue line in Figures 8.5 and 8.6. Figure 8.5 emphasizes the movements and also the waypoints relative to the navigation map while Figure 8.6 includes arrows pointing at regions of interest, allowing for easier discussion.

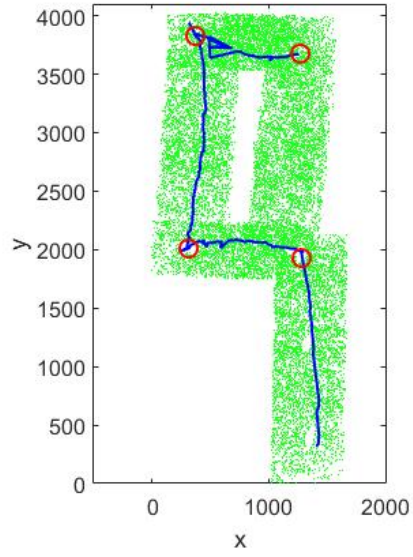


Figure 8.5: Estimated camera positions (blue path) with target waypoints (red circles) in the navigation map (green dots).

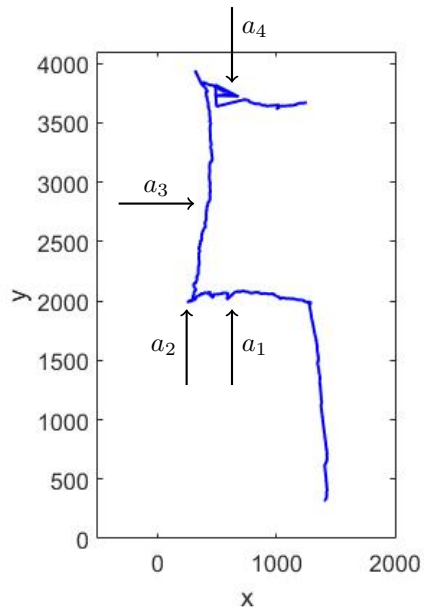


Figure 8.6: Estimated camera positions (blue path) and arrows (a_1, a_2, a_3, a_4) marking regions of interest which will be discussed later.

Chapter 9

Discussion

9.1 Matching results

For the specific example presented in the results, 913 features were extracted from an image. The number of image features depend on what is seen in the image and how the feature extractor is tuned. The more image features, the more likely it is to find good matches to the map. However, more features also means slower matching. This project suffered from robustness issues so quite a lot of image features were extracted from each image to increase the probability of good matches.

After matching, the number of feature matches were reduced to 45 by removing matches which were considered bad (Section 6.2). Using this method before the image registration with RANSAC (Section 6.3) proved to both make the code faster and more robust. The increased speed is based on that the amount of iterations in the RANSAC method could be reduced. Only selecting matches considered good (according to the method in Section 6.2) means increasing the relative amount of inliers to outliers.

The result of transforming image feature points to the map is shown in Figure 8.2. Notice that only matches considered good are plotted. This means that for every image feature, there is expected to be a corresponding map feature close by. This is exactly what is seen in the figure; every red circle overlaps its corresponding green circle.

It is important to notice that the zoom of Figure 8.2 is not specified. Even if it was, it would be related to lengths in the map coordinate system which is specified in *map units*. For this reason, no explicit measurement of how close the image points are to the map points is presented. What this image really shows is that the estimated rigid transformation described by \mathbf{R} and \mathbf{t} correctly transforms image points into the map.

9.2 Motion blur

As can be seen in Figure 8.4 the camera used for experiments suffered from heavy motion blur. The higher the velocity, the more blurred images. The maximum velocity of the robot was set to 0.03 m/s. The more blurred an image is, the less details are seen and the less image features are extracted. The features which still are extracted, will be distorted. From a pose estimation point of view the maximum allowed velocity should be as low as possible. If the velocity is chosen too low however, the robot movements will be quite uninteresting. One important part of this project was to see if any simple camera could be used. The motion blur of the USB camera proved to be so prominent that the performance was very limited. A rather strict limitation was needed on the maximum velocity.

An important analysis would be to evaluate different cameras. There are expensive cameras designed to reduce the amount of motion blur. Testing a camera like this is of high interest. It is, however, fairly extensive work to test different cameras since for each camera everything must be set up from scratch. Acquiring cameras and physically mounting them to the robot is only the initial step. The camera positions relative to the robot coordinate system must be found. Every camera must be calibrated, the distortion must be estimated, the tilt angles must be estimated, and the feature extractors and matchers must be tuned to get a reasonable amount of features and matches. Additionally, the navigation map must be adjusted based on each camera.

There are also mathematical methods to counteract the motion blur. This is yet again a way to remove undesired distortion effects. An important analysis would be to investigate if these techniques could be used to improve the image quality while driving. The restoration methods generally introduce artifacts in the de-blurred images that might make the feature matching inaccurate. It is important to remember that every procedure that is carried out at every iteration takes time. The sampling frequency is lowered which may lead to worse control of the robot.

9.3 Speeded-up matching

The bottle neck of the code proved to be the feature matching and it had to be made faster. Apart from the computational power, the sampling frequency depend on quite a few parameters. Therefore, the actual presented sampling frequencies are more importantly considered in relation to each other, and not for their actual values. The speeded-up matching algorithm resulted in a reduction of features from 32,640 to only 2,407. This reduction resulted in an increase of the average sampling frequency from 0.7 Hz to 5.2 Hz which corresponds to a 7.4 increase. This is a great performance improvement and proves

that taking the previous estimated position into account during matching is of high value.

In spite of great improvement in time, the algorithm became a lot more unstable. The sampling frequency analysis was made when the robot was standing still. While moving, the motion blurred images increase the probability of incorrect matches. As already mentioned (Section 6.5.1) an incorrect match could result in the algorithm depending on previous data that is completely wrong and this could cause the algorithm to crash. To safeguard against the behavior the new distance m was introduced as the the maximum allowed distance that the camera positions may vary between estimations. Before the introduction of m , the speeded-up algorithm was almost unusable while driving. After, for some values of m , the performance was greatly improved and could be used while driving. Still, the problem remained and the algorithm would sometimes fail.

Another disadvantage of the method is when the new estimated position has a higher value than the maximum allowed m . This would reset the algorithm so that the next estimation was found by comparing to the whole map. This resulted in irregular sampling times which may be undesirable from a control perspective.

Finding a good value of m is also not trivial. An initial idea was to include the kinematics of the robot, so that with a known sampling frequency and a maximum allowed velocity, m could be determined. Since the method introduced varying sampling times this proved to be rather uncertain. In the end the parameter m was treated as a user parameter.

Since robustness was an issue, a brute force matching algorithm was used for matching image features to map features. If a better camera with much less motion blur proves to give better image features, the algorithm may benefit from using some other faster matching algorithm. For example FLANN (fast approximate nearest neighbor) is an approximate method that is faster than brute force matching [9]. Because of the approximate nature, it is not as robust as a brute force algorithm.

9.4 Path following

The result of following a path of given waypoints is shown in Figures 8.5 and 8.6. The initial placement of the camera was unknown and had to be estimated by matching image features to the whole map. After the initial iteration the speeded-up matching (Section 6.5) was used for this experiment. This clearly shows that even though a simple movement algorithm (Section 7.2) was used, the goal can successfully be reached.

The estimated camera positions, seen in Figure 8.6, got some strange artifacts.

These are pointed out with the arrows (a_1, a_2, a_3, a_4) and discussed below.

- a_1 - This indicates a sudden bump in the camera position. The plot shows how the robot interprets its position within the map, and not the actual true position in the world. The sudden bump comes from a misplaced position estimation. The wrong estimates will always occur at some times, and they are increasingly probable with higher velocities with more motion blur.
- a_2 - This is another sudden bump while the robot is rotating around the camera to face the new waypoint. It appears for the same reasons as a_1 .
- a_3 - At first glance it may appear strange that the paths between waypoints are curved. However, this is a natural consequence of the movement algorithm. Assume the robot is located in the waypoint at a_2 . The robot then turns until the difference in angle between the camera direction and the next waypoint (the top left one) is smaller than the threshold ϵ_D (Section 7.2). This means the robot will not move straight at the waypoint. After driving forward for a bit, the angle will eventually get larger than ϵ_D , the robot will realign to the waypoint and start driving forward again. This will happen again and again until the waypoint is reached and this results in the curved paths.
- a_4 - This is a more extreme case of misplaced position estimations. Here a few misplaced estimations followed after one another. This is typically what happens when the speeded-up matching algorithm depends on old bad estimates. In this specific experiment the matching algorithm eventually, after a few iterations, managed to get back on track. This is a lucky case and will not always happen. This is a time where the speeded-up matching algorithm could have crashed.

There is another behavior that is not really captured in Figure 8.6. Sometimes when the robot stops to rotate towards the goal an oscillating behavior arises. This happens when the robot is rotating and is just about to come inside the threshold ϵ_D . If right at this moment the robot makes a wrong estimation, the robot will keep rotating in some direction and might miss the threshold. At the following iteration, assuming a correct estimation is made, the robot will have to correct its mistake and rotate back towards the waypoint. This manifests as oscillations. The slower the rotation and the higher the sampling frequency, the less likely this is to happen.

9.5 Map

An effect of building the map as suggested by Brange [2] is that the environment is assumed to be static. Consequently, the navigation in this thesis is also assumed to be in a static setting. This limits the use of the suggested method. When building the map it must be considered with great care where the robot is allowed to move and this area must be kept free of any obstacles. This might be fine in industry where the robot is assigned a specific task in a controlled environment. For home use, e.g., for use on a robot vacuum cleaner, this is likely a hindrance.

The floor itself which is the source of the map information must also be kept in the same state. As has been mentioned briefly earlier, some changes are allowed. In theory, as long as the robot never is able to take a picture of a completely unidentifiable piece of the floor, some part of the image should be present in the map and allow for feature matching. Still, it should be analyzed how big changes really are allowed. A thing to consider is if the map will still work when the floor is covered with dirt, oil spill or other unpredictable messiness.

Not only immediate changes to the structure or color of the floor are of importance. Different lighting conditions might affect how the floor is perceived by the robot. In some ways this comes down to an analysis of SURF features and is outside the scope of the method. However, the preprocessing of the images in this project is also important. If parts of the image are very different in light levels from when the map was built, the histogram equalization may result in images very different than expected. As of yet, this has not been considered with too much care since the room used for experiments had consistent light levels. Nonetheless, some complications were noted. Viewing the same point on the floor from different angles resulted in varying performance when matching to the map. This could cause wrong matches and misplaced estimations of the camera position as a result. This seemed to be caused by the robot itself casting shadows onto the images. A simple solution could be to mount a light near the camera to light up the floor. This would lead to more consistent light levels.

9.6 Conclusion

The ultimate goal of this thesis has been reached. A mobile robot was made autonomous by using feedback for positioning and path following from nothing but a single camera. From an unknown initial position, the robot is able to reach a desired goal position. The suggested method, where image features are matched against a feature navigation map, proved to be successful.

The measurements comes in the shape of images and they must be of high quality. Even though it has been shown that a cheap USB camera is sufficient, it limits the maximum velocities of the robot rather heavily in order to reduce motion blur.

Initially, the computations were rather time consuming. The suggested speeded-up matching algorithm increased the computational speed but also introduced instability in the software. Still, it shows how old estimation data can be used for improving the computational time.

Chapter 10

Future work

Already in the previous chapter areas where more research is needed were proposed. The idea is that they were closely related to the methods or to the results. This chapter will suggest other enhancements and improvements that would have been implemented if there was enough time. These are not crucial for the suggested method or implementation but may be important for optimization or from a user point of view.

10.1 Path

The robot movements are based on following a path of waypoints. For experiments, this path was manually chosen. In a more complete implementation you would expect to only give the desired goal position and the robot to navigate there itself. For this to work two things must be implemented.

The first thing is to find a way to interpret the feature navigation map to generate a graph, or roadmap. This roadmap should describe where the camera is allowed to move within the 2D world of the camera plane. Allowed areas would be where there are feature points in the map and not allowed areas would be anywhere outside.

Once a roadmap is found, a shortest path from the current position to the goal position could be generated by a shortest path algorithm such as Dijkstra's [10] or the A* search algorithm [11].

10.2 Robot control

The robot movement algorithm used in this thesis was a very simplistic one. It was sufficient to investigate the performance of the pose estimation from images, but proved to not work very well and a more sophisticated control algorithm is desired. If a path is given, there are path following algorithms that would be interesting to investigate.

10.3 Map coordinates

Throughout the project the measurements are often given in the global coordinate system as *map units* without any real interpretation in the real world. This is based in the problem formulation where it was chosen that the camera moves in the plane $z = 0$ and sees the floor in the plane $z = 1$. If the actual distance between the camera plane and floor plane is found, the map coordinates could be translated to real length units in meters.

Using map units over world units is not a problem per se. It is more important from a user perspective where it might be more intuitive to point out the goal position of the robot in the real world. A simple measurement was made to find the real world length per pixel. It was done to show the size of the navigation map and was not made with very high precision.

10.4 Map

The map used for experiments consisted of 32,640 features. That is a lot of features for such a small map. The coverage of features, as can be seen in Figure 8.1, seems to be a bit excessive. The sampling frequency, even with the suggested speeded-up matching method, is heavily dependent on the amount of map features. Therefore, reducing the amount of map features is of high interest. An attempt was made to reduce the number of features but it proved to make both Brange's algorithm [2] and this project's algorithm less stable. Since this project had robustness issues, mostly because of the motion blur, this was not analyzed further.

Bibliography

- [1] M. Wadenbäck and A. Heyden. 2014b. Ego-Motion Recovery and Robust Tilt Estimation for Planar Motion Using Several Homographies. In *Proceedings of VISIGRAPP 2014*, pages 635-639, Lisbon, Portugal, SCITEPRESS.
- [2] E. Brange. 2016. Efficient and robust map building from a downward looking camera with loop closing. *Master's thesis* at Faculty of Engineering, Centre for Mathematical Sciences, Lund University.
- [3] M. Fischler and R. Bolles. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Communications of the ACM*, 24(6), pages 381-395, New York, USA, ACM.
- [4] K. Derpanis. 2010. Overview of the RANSAC Algorithm. http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf (Available 2017-02-13)
- [5] A. de la Escalera, J. Armingol. 2010. Automatic Chessboard Detection for Intrinsic and Extrinsic Camera Parameter Calibration. In *Sensors 2010*, 10(3), pages 2027-2044, Madrid, Spain, Sensors.
- [6] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool. 2006. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110(3), pages 346-359.
- [7] B. Olofsson, K. Berntorp, A. Robertsson. 2015. A Convex Approach to Path Tracking with Obstacle Avoidance for Pseudo-Omnidirectional Vehicles. In *Report TFRT*, 7643. Department of Automatic Control, Lund Institute of Technology, Lund University.
- [8] K. Arun, T. Huang, S. Blostein. 2009. Least-Squares Fitting of Two 3-D Point Sets. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(5), pages 698-700, IEEE.
- [9] M. Muja and D. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11), pages 2227-2240, IEEE.

- [10] E. Dijkstra. 1959. A note on two problems in connexion with graphs. In *Numerische Mathematik*, 1(1), pages 269–271.
- [11] P. Hart, N. Nilsson, B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. In *IEEE Transactions on Systems Science and Cybernetics*, 4(2), pages 100-107, IEEE.
- [12] OpenCV. 2017. Camera calibration With OpenCV. http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html (Available 2017-02-13)
- [13] O. Carp. 2015. Årets julklapp är robotdammsugaren. *Dagens Nyheter*. 2015-11-24. <http://www.dn.se/ekonomi/arets-julklapp-ar-robotdammsugaren/> (Available 2017-02-13)
- [14] OpenCV. 2017. <http://opencv.org/> (Available 2017-02-13).
- [15] ROS. 2017. About ROS. <http://www.ros.org/about-ros/> (Available 2017-02-13).
- [16] Turtlebot. 2017. <http://www.turtlebot.com/> (Available 2017-02-13).
- [17] Matlab Coder. 2017. Generate C and C++ code from MATLAB code. The Mathworks Inc. <https://se.mathworks.com/products/matlab-coder/> (Available 2017-02-13).
- [18] Fraunhofer IPA. 2017. Hardware. <http://www.care-o-bot.de/en/robot-work/hardware.html> (Available 2017-02-27).
- [19] M. Oskarsson. 2015. Lecture 1: The Pinhole Camera Model. [Image]. <http://www.ctr.maths.lu.se/media/FMA270/2015/alllectures.pdf> (Available 2017-02-13).
- [20] Unknown photography. 2017. Plexgear 720p Webbkamera. [Image]. <https://www.kjell.com/se/sortiment/dator-natverk/datortillbehor/webbkameror/plexgear-720p-webbkamera-p61271> (Available 2017-02-28).
- [21] "Wolfwings". 2008. Barrel distortion visual example. [Image]. [https://en.wikipedia.org/wiki/Distortion_\(optics\)#/media/File:Barrel_distortion.svg](https://en.wikipedia.org/wiki/Distortion_(optics)#/media/File:Barrel_distortion.svg) (Available 2017-02-13)