Minimum Hardware SfM/SLAM for Sparse Data Point Mapping of Retail Stores

Daniel Falk

Master's thesis
2016:E43

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Master's Theses in Mathematical Sciences 2016:E43
ISSN 1404-6342

LUTFMA-3303-2016

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/

Minimum hardware SfM/SLAM for sparse data point mapping of retail stores

Falk, Daniel[1]

[1]*Lund University, Faculty of Engineering, Sweden*
Email: daniel@da-robotteknik.se

Abstract. This report aims to compare methods that can be used to efficiently and accurately map digital price labels position in $\mathbb{R}^3$ using a mobile robotic platform equipped with camera vision in a retail environment. The robotic platform built during the project uses four wide angle cameras to cover the full circular view in the horizontal plane. Installation of external systems for e.g. navigation can be very costly and time demanding. The accuracy, repeatability and time consumption have been evaluated for a set of methods and combinations thereof with and without external hardware. The result of the tests shows a proof-of-concept with good results but also emphasizes the need for a robust system. The mapping is done with high accuracy even when the external hardware for navigation is completely removed. As a part of the project a novel line following algorithm has been developed which shows results far better than any published results found and yet capable of running in real time utilizing limited hardware.

keywords
structure from motion, simultaneous localization and mapping, generalized imaging device, camera pose estimation, sparse data, point clouds, path planning, artificial intelligence, intelligent agent

Master's thesis, 30hp

Lund University, Sweden
Faculty of Engineering
Centre for Mathematical Sciences

Date of publication: September 2, 2016

Carried out at *Cognimatics AB*, Lund, Sweden
In cooperation with *Pricer AB* digital shelf-edge solutions

Author: Falk, Daniel
Affiliation: Department of Mechanical Engineering
Email: daniel@da-robotteknik.se
Phone: 0046 76-80 156 12

Supervisors

Examiner:	Åström, Kalle Mathematical Imaging Group Centre for Mathematical Sciences
1st supervisor:	Ardö, Håkan CRO, Cognimatics
2nd supervisor:	Haner, Sebastian Mathematical Imaging Group Centre for Mathematical Sciences
3rd supervisor:	Lindstedt, Gunnar Industrial Electrical Engineering and Automation Depertment of Mechanical Engineering

Author's preface

As an optimist I am sure that in the future robots will work back-to-back with humans in every field of work to ensure a comfortable living for us. The interest in robotics has always been a big part of me but despite my love for artificial intelligence my studies led me to the mechanical engineering programme. With an almost finished master specialized in mechatronics I searched my way towards studies in artificial intelligence at *Nanyang Technological University* in *Singapore*. The combination of my studies finally led me to my thesis at the *Centre for Mathematical Sciences* at *Lund University, Sweden*. For six months I have been working in an inspiring environment at the office of *Cognimatics AB* in *Lund* to finalize my thesis work. In the area of simultaneous localization and mapping there are always large efforts going on to push the boundaries, notwithstanding I am convinced that I have presented information complementing the work of others. With the final results in my hands my supreme and most important conclusion is that it has been a worthwhile experience for me and hopefully for all other parties involved.

The thesis is partly based on the following paper supplemented in appendix.

- Daniel Falk. *Cognitive vision for line following using stroke width vectorization*, 2016. Manuscript submitted for publication.

Acknowledgments

This project was made possible by the expertise of my colleagues at *Cognimatics AB*, the project supervisors and my examiner. I would foremost like to thank my primary supervisor Håkan Ardö for invaluable guidance in cognitive vision processing, mathematics and proofreading of the final report. The work in the project is based on the readings of hundreds of conference papers, theses and books among others written by my examiner Kalle Åström and supervisors Sebastian Haner, Håkan Ardö and Gunnar Lindstedt without which the results would not be achievable.

Introduction

Pricer AB has delivered more than 110 million digital price labels to over 13,500 stores world wide. Although the wireless system includes a method for geolocalization the accuracy is measured in meters. The system developed and presented in this report uses a mobile robot equipped with multiple cameras to build an accurate 3D map of the price labels' positions where the precision is in centimeter-range.

The main area of research in the field of robotics today concerns the awareness of the robot. To be aware of the environment the choice of sensors are of highest interest. Today digital cameras are available at low prices with previously unmatched performance. The main problem consists of understanding the constant stream of data. For decades industry and academia have been trying to improve the computers understanding of their environment based on single or multiple view camera images. Although the advances have been great the community still lies in a big data - low understanding situation.

This project has been focused on using multiple cameras on a mobile robotic platform to scan retail stores and find the position of digital price labels on the shelves. To maximize the sale and revenue much effort is spent planning the layout of retail stores, both using data mining techniques [52], [23], [33] and subjective experience. This process is often done remotely, especially in larger chains of stores. The physical placement of the products are performed by other personnel in the retail environment. The latter often introduces uncertainties in the placement and deviations from the plans possibly resulting in large reduction of revenue and profit.

When using multiple cameras as one the mathematics is commonly not fully understood. When a non-projective general imaging device is utilized the problem is often split up into several stereo vision problem albeit important information is lost. By evaluating and proposing a set of methods that can be used for Simultaneous Localization And Mapping (SLAM), or as often called in cognitive vision academy: Structure from Motion (SfM), a baseline for implementation and usage is introduced.

To make the result of the project as applicable as possible, avoiding purely theoretical solutions that do not reflect good results in reality, the project has been carried out in close cooperation with Pricer AB. Pricer AB is the global leader in digital shelf-edge solutions. With thousands of retail stores using their equipment the use for methods developed in this project is huge.

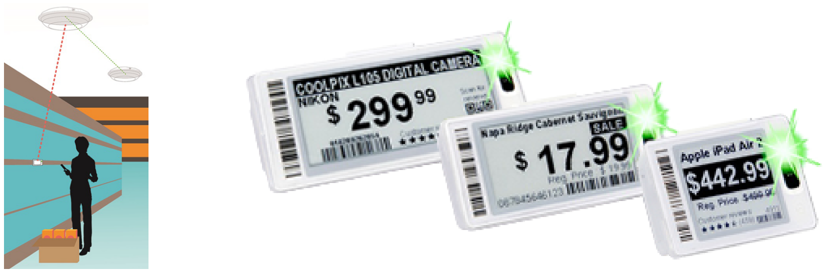


Figure 1: The Pricer Electronic Shelf Label system replaces paper price labels with digital screens connected to the network. In the right image the IR transmitters and the controllable green LEDs can be seen on each price label.

The methods implemented in the report combines the classical Simultaneous Localization And Mapping problem or the related Structure from Motion with an evaluation of sensors for navigation and estimation. In Figure 2 a simplified overall drawing of the experimental environment is visualized. The mobile robot is equipped with four wide angle cameras, a computer with WiFi and a set of sensors. Roof mounted cameras are evaluated to improve the localization of the mobile robot. A digital price label system has been installed in the laboratory together with other equipment to simulate a retail store as closely as possible.

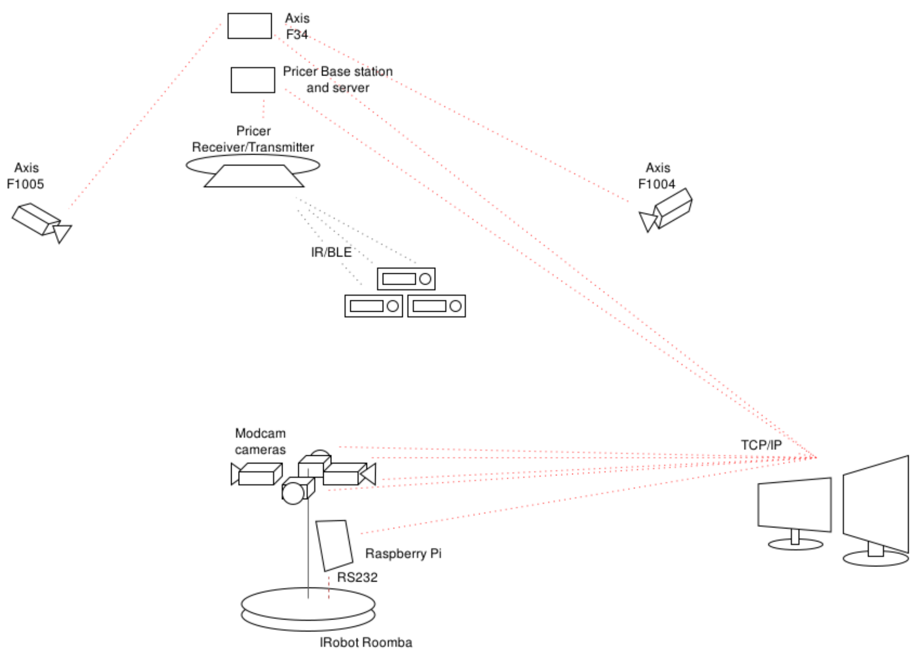


Figure 2: Experimental setup to develop and evaluate the system and methods. A mobile robotic platform equipped with vision systems can scan a retail environment to find digital price labels and map them in three dimensions.

Related work

The area of Simultaneous Localization And Mapping (SLAM) has been much researched and many papers focusing on different aspects of it are available such as *Aulinas Josep, et. al.* taking a rather wide view over the problem [24]. A paper by *Andrew J. Davison et. al.* presents an algorithm for SLAM using a single camera [39]. In the paper by *Micheal Kaess and Frank Dellaert* a method for SLAM using eight arbitrarily placed cameras was presented [56].

Much of the research within SLAM and SfM has focused on relatively dense data sets extracted either from laser scanners [40], [45], [50] or from dense feature point extraction in images [43], [32], [54]. With low-cost sonars as sensors *Teddy N. Yap, Jr. and Christian R. Shelton* entered the area of sparse data sets [73]. In their paper a minimum of 5915 data points were used which although sparse compared to the normal case is approximately hundred times denser than what is evaluated in this report.

The detection and identification of digital price labels based on a blinking characteristic is an area that has not been found in any papers. However the topic of detecting a blinking human eye has seen quite a bit of research. In a paper by *Kristen Grauman et. al.* communication based on the blinking pattern [47] was investigated and performed with many methods related to what is used in this report. Further research and implementation within the same topic have been done by *Michael Chau and Margrit Betke* [36].

Indoor navigation for route optimization is briefly investigated in this report. Some research of related navigation has been performed but not many publications covering autonomous in store navigation have been found. A patent by *Zimmerman, Thomas Guthrie* [75] shows as many other sources that the map building is automatic but the navigation through the store is manually done by an operator. Many indoor navigation systems relies on extensive RF tagging and accurate floor plans. The work carried out by *Aveek Purohit, Zheng Sun, Shijia Pan and Pei Zhang* [64] differs by using customers radio frequency and magnetic signatures to automatically build a map of movable pathways. Using this approach a success rate of above 85 percent has been reached with an accuracy of 0.7 meters. This method could be combined with the force field navigation that is briefly touched in this report to create an optimal route. The lack of precision can easily be bypassed by doing small real time modifications to the path using simple distance measurement sensors on the robotic platform.

Line detection and following have been part of the methods developed in this project. Much academic effort has been put with the focus to find robust techniques to detect lines and curves in images. A common method is to use Hough Line Transform [57], [53] to identify the dominant lines in the image. This was used in the Johnny-5 robot [25], first prize winner in the 2004 Intelligent Ground Vehicle Competition. This method removes all information about the curvature of the lines which holds much of the wanted knowledge. Another further developed method uses the vanishing point and combines it with a lane-curve function [65]. This method is good for outdoor navigation where the line/road has a clear vanishing point but would struggle with the indoor line following where the curves can be sharp and closely following each other. The method implemented during the project has surprisingly many similarities with the text detection problem in the Stroke Width Transform [41] developed by *Boris Epshtein, Eyal Ofek and Yonatan Wexler*.

Notations

To simplify the understanding of the equations some notations have been used throughout the report. Scalars are written as non bold characters while sets, vectors and matrices are written in bold notation. Sets of numbers are written as upper case letters while vectors and matrices are either upper or lowercase bold letters. The absolute value of a scalar s is written as $|s|$. A vector, $\boldsymbol{v}$, in $\mathbb{R}^n$ is written as $\boldsymbol{v} = (v_1, v_2, ..., v_n)^T$ while a matrix $\boldsymbol{m} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{bmatrix}$ is written with square brackets and a set $\boldsymbol{S} = \{1...S\}$ is written with curly brackets. L_2 norm of a vector $\boldsymbol{v}$, i.e Euclidean distance, is written as $\|\boldsymbol{v}\|_2$ and cardinality of a set, $\boldsymbol{S}$, is written as $\overline{\overline{\boldsymbol{S}}}$. The right arrow is used when defining new functions based on a specified input variable such as the $\mathbb{R}^3$ to $\mathbb{R}^1$ Euclidean distance

$$\boldsymbol{v} = (a, b, c)^T \rightarrow L_2(\boldsymbol{v}) = \sqrt{a^2 + b^2 + c^2}.$$

Contents

1 Terminology and methods: SLAM, SfM, BA

As a preceding step to the development and evaluation of algorithms the terminology and choice of method have to be explained. The terminology of Simultaneous Localization And Mapping (SLAM) and Structure from Motion (SfM) may sometimes be used in a confusing mixture and may or may not mean the same. Another less known term related with similar meaning is as an example Structure and Motion.

1.1 Terminology

The terms of Structure from Motion and monocular Simultaneous Localization And Mapping are often used interchangeably. SfM has its heritage in the photogrammetry community where the aim was to create scale models of a 3D object or environment based on an often sparse set of images. In these cases a projective model was often used and an optimization method such as Bundle Adjustment (BA). SLAM on the other hand was developed by the mobile robot industry with the same goal of creating a 3D model of the surroundings. In the cases with mobile robotics the sensor readings were often denser and covered a larger area. It could consist of cameras or other sensors such as lasers or ultra sonic transceivers. In SLAM related cases the filtering methods have historically been more common. In the latter years the two areas of research have in many cases merged and their terminology are therefore often used interchangeably.

1.2 Methods

Filtering methods, such as Kalman Filters (KF) or Monte Carlo Localization (MCL, particle filter), dismiss all previous poses and represent the gained information as a probability distribution. Key-frame methods such as BA make use of all poses' observations or a subset thereof. The subset can be defined by a sliding window or distributed key-frames. In (smaller) problems where all observations are used in the BA it is called a global Bundle Adjustment. In a wide and general case a successful method is to use the assumption of Gaussian distribution such as in BA or Extended Kalman Filters (EKF).

1.2.1 Complexity

In a simple case where only the core of the method is evaluated and bordering tasks such as the detection of tracking points are omitted the complexity of the BA and the EKF can be described based on the number of used frames f and the number of tracked points included in the map, n. The complexity is dependent of the structure of the mapping. In a dense mapping every tracked point is visible in each and every frame. This is normally not the case resulting in a sparser problem. As for an initial estimation to understand the methods the complexity derived by *Strasdat, Hauke et. al.* [68] was used

$$\mathcal{O}_{\mathrm{BA}}(f^2 n), \tag{1}$$

and

$$\mathcal{O}_{\text{EKF}}(n^3). \tag{2}$$

The total number of tracked points is a fixed value given by the number of price labels used in the store. In Figure 3 the complexity of the calculation for a frame given the two methods is visualized as a function of the total frames, $f \in \{0...8000\}$, and total detected labels, $n \in \{0...10000\}$.

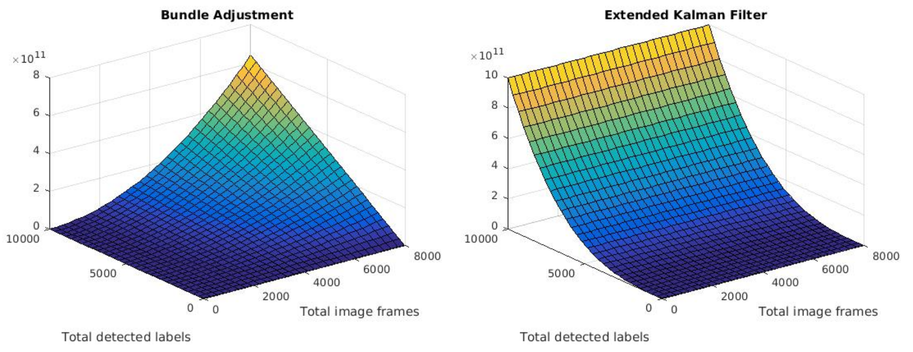


Figure 3: Visualization of the methods complexity response to input size of total frames and landmarks used. Left: Bundle Adjustment, right: Extended Kalman Filter.

In a retail environment where the price labels are dense the Bundle Adjustment method might be preferred over the Extended Kalman Filter.

1.2.2 Analysis of efficiency

The efficiency of the algorithms have been investigated in a deeper sense by *Strasdat, Hauke et. al.* [68]. The efficiency is affected by factors such as the accuracy of the algorithm and the search time to detect labels in the images. One conclusion indicates that it is better to increase the number of tracked points before the number of poses the tracking is conducted at. In the retail environment the number of tracked points is given by the number of installed price labels and is therefore not a changeable parameter. Their conclusion also states that in general Bundle Adjustment is superior to Extended Kalman Filtering. When only a small processing hardware is available the filtering methods might be better but when high quality results are requested the Bundle Adjustment is preferable. In accordance with *1.2.1 Complexity* the choice of method was therefore decided in favor for the Bundle Adjustment method.

2 Theory of 3D projection and optimization

The following section starts with a fast explanation of the basic math behind world projection to an image and image planes correspondence to each other. Camera calibration and pose estimation are explained. The section focuses mainly on the calibration of the robotic platform, that is: the relative location of the four cameras, the two beacons locations and pose of the statically mounted roof cameras. The theory is here explained as generically as possible but is in the *6 Implementation* section applied to the specific environment used in this project.

2.1 A world point's projection in the image plane

The image plane of the camera is placed a distance f behind the cameras focal point where f is the focal distance. For simplification the image plane can be assumed to be placed in front of the focal point, thus inverting the image plane axes i.e. rotating the image π radians within the plane. In addition to the world coordinate system lets define a coordinate system with its origin in the cameras focal point such that the image plane is located at $\boldsymbol{i'} = (0, 0, f)^T$. Lets further say that its z-axis is perpendicular to the cameras image plane. The relation between the world coordinate system and this system can be described by a rotational matrix $\boldsymbol{R}$ and a translation $\boldsymbol{t}$. A point $\boldsymbol{X}$ in the world coordinate system can thus be transformed to coordinates $\boldsymbol{X'}$ in the cameras coordinate system according to

$$\boldsymbol{X'} = \boldsymbol{R}\boldsymbol{X} + \boldsymbol{t}. \tag{3}$$

By increasing the dimensionality of the point into the projective space the point will be scale invariant. In homogeneous coordinates a point in $\mathbb{R}^3$ $\boldsymbol{X} = (X_x, X_y, X_z)^T$ is defined as $\boldsymbol{X} = (X_1, X_2, X_3, X_4)^T$ where

$$X_x = \frac{X_1}{X_4}, X_y = \frac{X_2}{X_4} \quad \text{and} \quad X_z = \frac{X_3}{X_4}.$$

Thus $\boldsymbol{X}$ is the same point as $2\boldsymbol{X}$ or $n\boldsymbol{X}$, where n is any real number. A point $\boldsymbol{X} = (X_1, X_2, X_3, 0)^T$ is a point at infinity described by the direction of the first three parameters. Using homogeneous coordinates Equation 3 can be further simplified to

$$\boldsymbol{X'} = \boldsymbol{T}\boldsymbol{X}, \quad \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}. \tag{4}$$

By modeling the camera as a pinhole camera the projected point in the image plane can be found by simple geometry. Many books and publications seek to punctiliously describe the process of projecting the world to the image plane. Considering that, this report will not cover the details but instead refer the interested readers to books such as *Multiple View Geometry* by *Hartley R. et al* [51]. Instead it can be ascertained that given a 3x3 projection matrix $\boldsymbol{K}$ known as the camera matrix the homographic projection onto the image plane is described by

$$\boldsymbol{x} \propto \boldsymbol{K}\boldsymbol{T}\boldsymbol{X}, \tag{5}$$

where $\boldsymbol{x}$ is a homogeneous 2D coordinate, i.e. a 1x3 vector. The camera matrix $\boldsymbol{K}$ is consisting of the focal length and optic center described in x and y pixel lengths and might also include a pixel skew factor.

However the pinhole camera model does not model any lens distortion. Lens distortion can be non linear and thus the coordinate system of an uncalibrated camera is not Euclidean. When using wide angle cameras the image will be distorted in what is sometimes called a barrel shape. In this report the lens distortion is modeled using three radial and two tangential parameters. The radial terms removes the so called barrel effect while the tangential terms compensates for lenses not parallel to the image plane. Let u' and v' describe the estimated coordinates in the distorted image based on the coordinates u and v in an Euclidean image

$$\begin{aligned} u' &= u * (1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6) + 2\tau_1 uv + \tau_2(r^2 + 2u^2) \\ v' &= v * (1 + k_1 * r^2 + k_2 * r^4 + k_3 * r^6) + \tau_1(r^2 + 2v^2) + 2\tau_2 uv \\ r^2 &= u^2 + v^2. \end{aligned} \tag{6}$$

In some cases the inverse of the lens distortion is needed. It is not suitable to find an algebraic inverse to Equation 6 however making use of Banach fixed-point theorem, [62],[28], an iterative solution can be used

Algorithm 1 Inverse of lens distortion for an image point $x = (u', v')^T$

```
$u := u'$
$v := v'$
while change < converge criteria do
    $r_2 := u^2 + v^2$
    $c_{\text{inv}} := 1/(1 + ((k_3 r_2 + k_2) r_2 + k_1) r_2)$
    $l_{\text{x}} := 2\tau_1 uv + \tau_2(r_2 - 2u^2)$
    $l_{\text{y}} := \tau_1(r_2 + 2v^2) + 2\tau_2 uv$
    $u := c_{\text{inv}} * (u' - l_{\text{x}})$
    $v := c_{\text{inv}} * (v' - l_{\text{y}})$
return $u, v$
```

2.2 Solving the cameras position

With a set of known points in the world and their correlating projections in an image it is possible to estimate the pose of a calibrated camera [44][60]. Every known point introduces three new equations which are up to scale and thus also introduces one unknown scale factor, λ. The pose of the camera is described by a 3x3 rotational matrix and a 1x3 translation, totally including 12 parameters. The number of known points, N, needed to solve the cameras position is thus given by

$$3N - N \geqslant 12 \quad \Leftrightarrow \quad N \geqslant 6.$$

However the rotational matrix has to be orthogonal and it has been proved [37] that a cameras position can be found with a least squares manner with

$$N > 3, \quad \text{given rotational matrix constrains.}$$

Because of noise in the detection and discretization in the image sensor the reprojections will have no exact solution. Thus the optimal solution is obtained by minimizing the error. It is possible to use an iterative algorithm to find the pose minimizing the sum of squared L_2 reprojection errors for all points. Either the actual detected point can be undistorted by using the iterative inverse of the distortion or the projected point can be distorted using the faster Equation 6, albeit in the latter case it needs to be calculated on every iteration's calculation of the cost function. From hereon it will be assumed that all image coordinates are undistorted and thus represented in an Euclidean space. The minimizationn is done over the camera pose, i.e. the cameras rotation and translation from the world coordinates. To project the points the transformation matrix, including the rotation matrix, is needed. To minimize a function with a rotational matrix as input demands that the two conditions for the rotational matrix are fulfilled; a rotational matrix is orthogonal and has a unit determinant

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I} \quad \text{and} \quad \det \boldsymbol{R} = 1. \tag{7}$$

Lets define the optimization matrix $\boldsymbol{Q}$ as an Euler rotation and a translation vector

$$\boldsymbol{Q} = [\boldsymbol{r}|\boldsymbol{t}] = \begin{bmatrix} r_x & t_x \\ r_y & t_y \\ r_z & t_z \end{bmatrix}. \tag{8}$$

The vector with rotations around x,y and z axis can be converted to a rotational matrix according to:

$$\boldsymbol{r} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \rightarrow$$

$$\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{r}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(r_x) & s(r_x) \\ 0 & s(r_x) & c(r_x) \end{bmatrix} \begin{bmatrix} c(r_y) & 0 & s(r_y) \\ 0 & 1 & 0 \\ -s(r_y) & 0 & c(r_y) \end{bmatrix} \begin{bmatrix} c(r_z) & -s(r_z) & 0 \\ s(r_z) & c(r_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{9}$$

where c denotes the cosine and s denotes the sine function. From a matrix $\boldsymbol{Q}$ it is thus possible to calculate a transformation matrix $\boldsymbol{T}$ as

$$\boldsymbol{Q} = [\boldsymbol{r}|\boldsymbol{t}] \rightarrow \boldsymbol{T} = \boldsymbol{T}(\boldsymbol{Q}) \overset{\text{Equation 9}}{=} \boldsymbol{T}(\boldsymbol{R}, \boldsymbol{t}) = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix}. \tag{10}$$

With N known points with the detected image coordinates $\boldsymbol{x}_n$ and the reprojected coordinates $\tilde{\boldsymbol{x}}_n$ the minimization problem can be described as

$$\underset{\boldsymbol{Q}}{\operatorname{argmin}} \sum_{n=1}^{N} v_n \left\| \tilde{\boldsymbol{x}}_n - \boldsymbol{x}_n \right\|_2^2, \tag{11}$$

where v_n is a binary variable stating if the point n can be seen in the image. Defining function

$$\boldsymbol{v} = (\alpha, \delta, \omega)^T \quad \rightarrow \quad \text{pflat}(\boldsymbol{v}) = (\frac{\alpha}{\omega}, \frac{\delta}{\omega}, 1)^T, \tag{12}$$

and combining with the equation for projection, Equation 5, the minimization problem can be written as

$$\underset{\boldsymbol{Q}}{\text{argmin}} \sum_{n=1}^{N} v_n \left\| \text{pflat}(\boldsymbol{KT}) - \boldsymbol{x}_n \right\|_2^2 . \tag{13}$$

2.3 Finding cameras relative positions

If the relative pose and intrinsics of the cameras are known it is possible to see them as what is called a single generalized image device [63], [26]. A projective camera only has one focal point which all light rays pass through. A general image device on the other hand can have multiple focal points at different locations but can still in many situations be considered as a single camera.

Lets again use the matrix $\boldsymbol{Q}$ defined in Equation 8 i.e. a 3x2 matrix and define $\boldsymbol{O}$ as a matrix containing the translation and rotation vectors for all used cameras, $c \in \{1...C\}$, at all positions, $f \in \{1...F\}$, where a frame was taken

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{Q}_{11} & \boldsymbol{Q}_{12} & \cdots & \boldsymbol{Q}_{1C} \\ \boldsymbol{Q}_{21} & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{Q}_{F1} & \cdots & \cdots & \boldsymbol{Q}_{FC} \end{bmatrix}, \tag{14}$$

with the size $3F$x$2C$. Equation 11 can be expanded to cover the reprojection error from all cameras and image positions:

$$\underset{\boldsymbol{O}}{\text{argmin}} \sum_{f=1}^{F} \sum_{c=1}^{C} \sum_{n=1}^{N} v_{f,c,n} \left\| \tilde{\boldsymbol{x}}_{f,c,n} - \boldsymbol{x}_{f,c,n} \right\|_2^2 . \tag{15}$$

We can thus in a single optimization problem find the pose for all the cameras at the time of each frame. The same result as is achieved by solving Equation 15 can be found by solving Equation 11 one time for each camera and frame. The reason that Equation 15 is more powerful comes with the power to introduce further constrains.

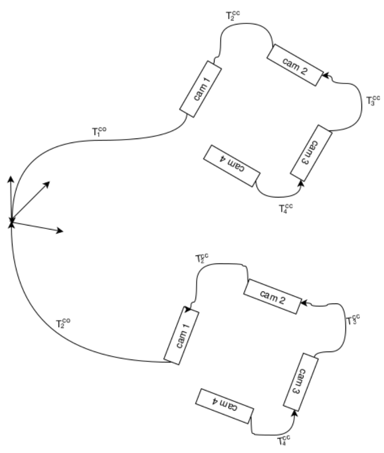


Figure 4: Mounted on a fixed platform the cameras can be modeled with static relative pose to each other. Even if the pose of the robot changes the transformation between the cameras stay the same, albeit the transformation from the first camera to the world origin changes. Using this knowledge the degrees of freedom reduces from $6FC$ to $6(F + C - 1)$ where F is the number of robot poses and C is the number of cameras.

With the cameras mounted fixed on the robot we can make use of the knowledge that even if the world coordinates are not known the relative pose between the cameras will always stay the same. This can be considered as a kinematic chain. We can define F different transformations from the first cameras coordinate system to the world system, $\boldsymbol{T}_f^{\text{co}}$, and $F - 1$ transformations, $\boldsymbol{T}_{i,i-1}^{\text{cc}}$, between the camera i and $i - 1$ where $i \in \{F, .., 3, 2\}$. Given a camera, c, and a frame, f, the transformation to the world system can now be calculated as

$$\boldsymbol{T}_{c,f} = \Big(\prod_{j=0}^{F-1} \boldsymbol{T}_{F-j,F-j-1)}^{\text{cc}}\Big)\boldsymbol{T}_f^{\text{co}}. \tag{16}$$

Using this approach the optimizing matrix $\boldsymbol{O}$ in Equation 15 can now be reduced to

$$\boldsymbol{O} = \big[\boldsymbol{Q}_1^{\text{CO}}, \boldsymbol{Q}_2^{\text{CO}}, ..., \boldsymbol{Q}_F^{\text{CO}}, \boldsymbol{Q}_{2,1}^{\text{CC}}, \boldsymbol{Q}_{3,2}^{\text{CC}}, ..., \boldsymbol{Q}_{C,C-1}^{\text{CC}}\big], \tag{17}$$

thus reducing the degrees of freedom from $6FC$ to $6(F + C - 1)$. The minimum case to the relative pose problem with two generalized cameras has been investigated by *Stewnius, Henrik, et al.* [66] using the intersection of six image rays. With a large number of dimensions to optimize over the possibility to get stuck in local extremities increases when using iterative algorithms. The cameras on the robot built in this project are expected to be placed on a circle in the horizontal plane, and with its image plane parallel to the circles tangent and perpendicular to the cameras next to it. It can be assumed that the error marginal for the angle and position is small. With this assumption the pose of the image planes in the model can be defined to be tangential to a sphere around a common center point. All the cameras pose in relation to the center point can thus be described using one three degree of freedom rotation vector for each camera and a common radius. With this approach the total degrees of freedom in the search problem is reduced to $3(F + C) + 1$. To make sure that the model is as accurate as possible this simplified model can be used to find the initial guess for the more complex and accurate model thus minimizing the possibility of getting stuck in a local minimum.

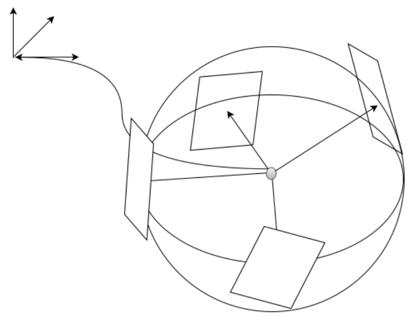


Figure 5: To further reduce the degrees of freedom all cameras can be assumed to have their image plane as a tangent on the same sphere. The distance in the cameras z-axis to the center of the sphere is then all the same for the cameras. The degree of freedom is then reduced to $3(F + C) + 1$.

Lets use the vector $\boldsymbol{S}_f^{\text{CCO}}$ as the translation from the camera spheres center point at location f to the world origin and the vector $\boldsymbol{S}_c^{\text{CCC}}$ as the rotation vector from the camera c to the spheres center. The radius of the sphere is represented by r. The optimization vector $\boldsymbol{O}$ can now be defined as

$$\boldsymbol{O} = \left[\{\boldsymbol{S}_1^{\text{CCO}}\}^T, \{\boldsymbol{S}_2^{\text{CCO}}\}^T, ..., \{\boldsymbol{S}_F^{\text{CCO}}\}^T, \{\boldsymbol{S}_1^{\text{CCC}}\}^T, \{\boldsymbol{S}_2^{\text{CCC}}\}^T, ..., \{\boldsymbol{S}_C^{\text{CCC}}\}^T, r\right], \tag{18}$$

where the transform from camera center to world origin is given by

$$\boldsymbol{S}_f^{\text{CCO}} = (S_{tx}, S_{ty}, S_{tz})^T \quad \rightarrow \quad \boldsymbol{Q}^{\text{CCO}} = \boldsymbol{Q}^{\text{CCO}}(\boldsymbol{S}_f^{\text{CCO}}) = \left[\begin{array}{cc} \mathbf{0}_{3\text{x}1} & \boldsymbol{S}_f^{\text{CCO}} \end{array}\right], \tag{19}$$

and transformation from camera c to sphere center is given by

$$\left.\begin{array}{r} \boldsymbol{S}_c^{\text{CCC}} = (S_{rx}, S_{ry}, S_{rz})^T \\ r \end{array}\right\} \quad \rightarrow \quad \boldsymbol{Q}_f^{\text{CCO}} = \boldsymbol{Q}_f^{\text{CCO}}(\boldsymbol{S}_c^{\text{CCC}}, r) = \left[\begin{array}{cc} S_{rx} & 0 \\ S_{ry} & 0 \\ S_{rz} & r \end{array}\right]. \tag{20}$$

Using this knowledge about the cameras intrinsics and extrinsics it is now possible to consider the setup as one single generalized image device.

2.4 Estimate 3D positions in partly unknown point clouds

Given a point cloud where the real world coordinates are known for a subset of the points, the location of the rest of the points can be estimated using the knowledge gained from the camera calibrations. Four known points are needed to find the pose of a camera in three dimensions, the same goes for the generalized imaging device created from the combination of the calibrated cameras [37]. With a perspective camera, i.e a camera with a single focal point an infinite number of solutions can be found to the perspective 3-point problem if all points are collinear, i.e on the same line through space. With a non perspective imaging device another non solvable case can be found where the rays from the point to the cameras focal points are parallel. In the first case the rotation around and the translation along the axis on which the points lies are unknown and in the second case the translation along the rays is unknown. If the cameras are overlapping and the unknown points can be seen in at least two cameras from the same robot location the points' 3D coordinate can be estimated from frames taken from a single robot position. In the other case at least two different locations need to be used to solve the points' locations. The method proposed consist of the following steps:

1. Start with an initial guess of the robots position
2. Rotate the robots pose estimation around the vertical axis in smaller and smaller steps until all points are in front of the image plane of all the cameras that can see them
3. Solve robot pose iteratively by minimizing the sum of the reprojection errors for all cameras
4. Solve all unknown points locations linearly by triangulation, use singular value decomposition to find an approximation of the nullspace
5. Optimize over all variables using bundle adjustment to find the maximum likelihood solution to the complete problem

2.4.1 Initial guess

The initial guess is of utter importance for the result. The cost function for a minimization problem of this scale has a large amount of local minima and an unrestricted search space. With the wrong initial guess it is thus possible to get an unacceptable solution. According to the experiments done during this project the initial guess of position is not as crucial and can easily be done by observed subjective estimate. The most restricting technique used at this stage is the rotation around the vertical axis. Using the stop condition that all points seen by the camera should be in front of image plane is fine as long as the position is correct. If the guess of the position is not correct there might be a situation where no rotation is fulfilling the stop criteria. No more time has been spent to this problem because using the techniques of either deduced reckoning or roof mounted cameras explained during the later chapters yields good estimation of both position and direction thus removing the need completely for step 2 in the enumeration above.

2.4.2 Solve robot pose

The robot's locations relative to each other can be solved by minimizing the reprojection error of all points, known or unknown, in the cameras' image planes while the absolute positions are yielded from the reprojection of the known points. To find the robots position from known points Equation 15 is reused but this time the minimization matrix is defined as

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 & ... & \boldsymbol{Q}_F \end{bmatrix}, \tag{21}$$

where $\boldsymbol{Q}_f$ is the six degree of freedom translation and rotation of the robot at the location of frame $f \in \{1...F\}$. In the case where the cameras are fixed on a robot traveling on a planar surface the number of degrees of freedom can be reduced to three, a two axis translation and a one axis rotation. With two known points seen from the generalized image source the possible positions of the robot is located on a circle in the plane which normal is parallel to the line going through both the points. It is thus possible to have a maximum of two intersections with the horizontal plane that the cameras are assumed to be traversing in. Further more if the identity of the two points are known one of the two intersections would result in the robot being up-side-down. Thus two identifiable points are enough to position the robot. The only exception to this is if the axis between the two points is perpendicular to the ground plane, the circle can then either be in the plane resulting in an infinite amount of solutions or parallel with the plane resulting in none solutions.

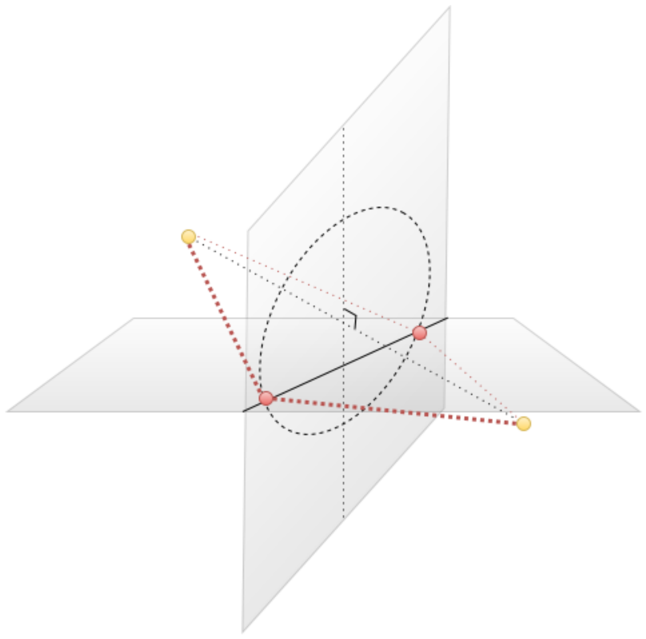


Figure 6: With two uniquely identifiable points the pose of the robot is bound to a circle. If the robot is restricted to traveling in a plane only a maximum of two positions are possible. If the robot is allowed to rotate only around the planes normal axis only one possible solution is left.

2.4.3 Triangulation of points

A point seen in a camera can be expected to be located close to the $\mathbb{R}^3$ line defined by the projection point in the image plane and the cameras optic center. Given a cameras limited resolution the theoretical space mapped to a specific pixel is described by a pyramid shape where the top of the pyramid is the focal point and each side of the pyramid goes through the edge of the pixel. This volume is further extended by the introduction of noise. Given this it can be understood that a number of lines going from different cameras optic center and through the center of the detected pixel are not likely to intersect each other in space. The relations between the coordinates can be solved using Direct Linear Transform [51]. Because of the homogeneous coordinates the relations are up to scale and we can thus introduce a scalar λ_n for each projection equation. Equation 5 can thereby be written as

$$\lambda \boldsymbol{x} = \boldsymbol{KTX}. \tag{22}$$

Applying Equation 22 to each camera, $c \in \{1...C\}$, seeing the same point $\boldsymbol{X}$ results in

$$\begin{bmatrix} \boldsymbol{K}_1\boldsymbol{T}_1 & -\boldsymbol{x}_1 & 0 & \dots & 0 \\ \boldsymbol{K}_2\boldsymbol{T}_2 & 0 & -\boldsymbol{x}_2 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ \boldsymbol{K}_C\boldsymbol{T}_C & 0 & 0 & \dots & -\boldsymbol{x}_C \end{bmatrix} \begin{bmatrix} \boldsymbol{X} \\ \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_C \end{bmatrix} = \mathbf{0}, \tag{23}$$

which also can be described as the $3C \times (4+C)$ $\boldsymbol{A}$ matrix, the $4+C$ row vector $\boldsymbol{v}$ and a $4+C$ null row vector

$$\boldsymbol{A}\boldsymbol{v} = \mathbf{0}. \tag{24}$$

Based on the volume being mapped to the same pixel it is most unlikely that Equation 23 has an exact nonzero solution. By using singular value decomposition the approximative nullspace can be found [59] as the eigenvector of $\boldsymbol{A}$ found as the column in $\boldsymbol{V}$ corresponding to the smallest eigenvalue which is found in the diagonal elements of $\boldsymbol{\Sigma}$

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*, \tag{25}$$

where $\boldsymbol{V}^*$ is the conjugate transpose which is the same as the transpose when $\boldsymbol{A}$ consist of non complex values

$$\boldsymbol{A}_{\text{real}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T. \tag{26}$$

2.4.4 Bundle adjustment

The quantity being minimized in the linear cost function is

$$\left\| \begin{bmatrix} \boldsymbol{K}_1\boldsymbol{T}_1\boldsymbol{X} - \lambda_1\boldsymbol{x}_1 \\ \boldsymbol{K}_2\boldsymbol{T}_2\boldsymbol{X} - \lambda_2\boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{K}_C\boldsymbol{T}_C\boldsymbol{X} - \lambda_C\boldsymbol{x}_C \end{bmatrix} \right\|_2, \tag{27}$$

and is thus dependent on the λ_c scalars which are values without meaning. Any nonzero scalar multiplication of both sides of the expression in Equation 5 will yield a new λ_c but still satisfy the equation. This error is thus not geometrically meaningful and the solution is not the optimal for the problem. Although there are many algebraic ways to solve the problem based on Gröbner basis [58] [35], bundle adjustment is a modern iterative method used to find the optimal solution. In bundle adjustment the squared L_2 reprojection error is minimized which yields the most desired solution. If the measurement errors are normally distributed, independent and have constant standard deviation this results in the maximum likelihood estimation. The probability distribution of the mean over several independent errors can be expected to converge to a normal distribution when the amount of variables increase [22]. Define $\boldsymbol{X}_m$ as the position in $\mathbb{R}^3$ of an unknown point $m \in \{1...M\}$. To minimize the reprojection error of all points in all cameras Equation 15 is used again with the minimization matrix defined as

$$\boldsymbol{O} = \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{Q}_2 & ... & \boldsymbol{Q}_F & \boldsymbol{X}_1 & \boldsymbol{X}_2 & ... & \boldsymbol{X}_M \end{bmatrix}. \tag{28}$$

Here M is the number of unique unknown points, which is a subset of all N points. If the environment is static and the robot has enough views of points, known or unknown, to get the relative transformation between the robots locations the combination of all the generalized cameras in the different locations can be seen as a single generalized camera and thus

$$M < N - 4 \tag{29}$$

is required to solve the position of all unknown points. The minimal case to find relative transformation between two poses of generalized image devices are 6 point correspondences [66]. This number is reduced if the transformation is limited to three degrees of freedom.

The non linear least squares is sensitive to outliers because of the thin tails of Gaussian distributions [71] and it is thus important that the data points is filtered for outliers before optimization or that an other robustness improving method is used [31].

2.5 Position feedback from roof mounted cameras

The roof mounted cameras are used to acquire non drift prone position estimation of the robot. The detection of the robot is done by identifying two flashing LEDs on top of the robot, hereafter called the beacons. The view from the cameras have a strong perspective warp of the floor but can be calibrated using a simple model. The robot is known to travel in the plane of the floor which is why a single camera is enough to find the full pose of the robot.

Given two arbitrary planes in $\mathbb{R}^3$ it is possible to calculate a transformation matrix from one plane to the other if enough correspondence is known. The projective transformation consist of a similarity transform, an affinity and a projectivity. A transformation from $\mathbb{R}^2$ to $\mathbb{R}^2$ is done by multiplication between the homogeneous coordinate and a 3×3 matrix. Transformation in the opposite direction can easily be done by multiplication with the inverted matrix. If several matrices are to be combined they can simply be multiplied together to form a new matrix of the same dimension and properties.

2.5.1 Finding homography transforms

The correspondence can consist of points, lines or conics known in both planes. Given a homogeneous coordinate in the cameras Euclidean image plane, $\boldsymbol{\rho} = (u, v, 1)^T$, and a 2D-position $\boldsymbol{\chi} = (x, y, 1)^T$ in the plane parallel with the floor at the height of the robots beacons the relationship can be written

$$\boldsymbol{\chi} \propto \boldsymbol{H}\boldsymbol{\rho}, \tag{30}$$

or by introducing a scalar λ which represent the arbitrary scale factor

$$\lambda \boldsymbol{\chi} = \boldsymbol{H}\boldsymbol{\rho}, \tag{31}$$

where $\boldsymbol{H}$ is a 3×3 matrix called the homography matrix. Rearranging Equation 31 to the form $\boldsymbol{A}\boldsymbol{h} = \mathbf{0}$ using Direct Linear Transform as before results in

$$\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & x_1u_1 & x_1v_1 & x_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ -u_2 & -v_2 & -1 & 0 & 0 & 0 & x_2u_2 & x_2v_2 & x_2 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & y_2u_2 & y_2v_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_N & -v_N & -1 & 0 & 0 & 0 & x_Nu_N & x_Nv_N & x_N \\ 0 & 0 & 0 & -u_N & -v_N & -1 & y_Nu_N & y_Nv_N & y_N \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix} = \mathbf{0}, \tag{32}$$

which can be solved using singular value decomposition in the same way as Equation 23 to 26. The approximative nullspace given by $\boldsymbol{h}$ can be reshaped to the 3×3 matrix $\boldsymbol{H}$. Because of the homogeneous coordinates, $\boldsymbol{H}$ is up to scale and thus has exactly eight degrees of freedom. Each point correspondence results in three new equations but also a new unknown scale factor, thus the number of points required to find the homography matrix, with the scale invariant considered, is given by

$$3N - N \geq 9 - 1 \Leftrightarrow N \geq 4. \tag{33}$$

2.5.2 Evaluating overlapping camera homography

If several roof mounted cameras are partly or fully overlapping and set up with a homography mapping to the same plane the calibration error between the cameras can be calculated. Equation 31 explains the transform from one camera to the calibrated plane. Based on this we find the projection from one camera to the other. Let $\boldsymbol{\rho}_a$ be the detected point in camera a, $\boldsymbol{\chi}$ the calibration points true 2D position in the plane and $\boldsymbol{\rho}'_b$ the point projected in camera b.

$$\boldsymbol{\rho}'_b \propto \boldsymbol{H}_b^{-1}\boldsymbol{\chi}, \quad \boldsymbol{\chi} \propto \boldsymbol{H}_a\boldsymbol{\rho}_a \quad \Rightarrow \quad \boldsymbol{\rho}'_b \propto \boldsymbol{H}_b^{-1}\boldsymbol{H}_a\boldsymbol{\rho}_a, \tag{34}$$

the symmetrical error between I cameras can thus be described as

$$\sum_{\substack{i=1\ldots I \\ j=1+I-i \\ i\neq j}} \frac{1}{2I}\Big(\left\|\boldsymbol{\rho}_i - \boldsymbol{H}_i^{-1}\boldsymbol{H}_j\boldsymbol{\rho}_j\right\|_2 + \left\|\boldsymbol{\rho}_j - \boldsymbol{H}_j^{-1}\boldsymbol{H}_i\boldsymbol{\rho}_i\right\|_2 \Big). \tag{35}$$

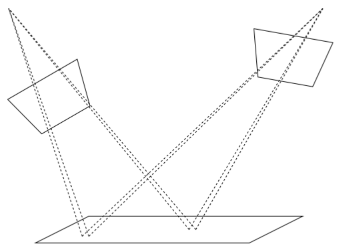


Figure 7: A projection from one cameras image plane to another plane is described by the homography matrix. If two cameras projection to the same plane is known the projection from one camera to the other can easily be calculated.

2.5.3 Improving homography through bundle adjustment

Just as discussed with the linear triangulation this combination of Direct Linear Transform and Singular Value Decomposition results in a suboptimal solution because of the geometrically uninteresting character of the optimized quantity. A better solution can be achieved by utilizing this method only to yield an initial guess of the homography for iterative bundle adjustment. The parameters of the homography matrix can be included in the optimization matrix when the camera relative pose calibration is done. Still using Equation 15 the optimization vector in Equation 18 is extended to

$$\boldsymbol{O} = \left[\{\boldsymbol{S}_1^{\mathrm{CCO}}\}^T, \{\boldsymbol{S}_2^{\mathrm{CCO}}\}^T, ..., \{\boldsymbol{S}_F^{\mathrm{CCO}}\}^T, \{\boldsymbol{S}_1^{\mathrm{CCC}}\}^T, \{\boldsymbol{S}_2^{\mathrm{CCC}}\}^T, ..., \{\boldsymbol{S}_C^{\mathrm{CCC}}\}^T, r, \hat{\boldsymbol{H}}\right], \tag{36}$$

where $\hat{\boldsymbol{H}} = (H_1, H_2, ..., H_{8R})$ contains the element one to eight of all R roof cameras' homography transformation matrices. Because of the scale invariance in the homography matrix $\boldsymbol{H}$, $\hat{\boldsymbol{H}}$ has only eight elements from each homography matrix while $H_{r,9}$ is fixed to e.g. unit value.

2.5.4 Finding beacons relative positions

After finding the optimal solution to Equation 32 for each roof mounted camera a point i with 2D coordinates $\boldsymbol{\chi}_i$ can be reprojected into the camera r's image coordinates $\tilde{\boldsymbol{\rho}}_{i,r}$ according to

$$\lambda_{i,r}\tilde{\boldsymbol{\rho}}_{i,r} = \boldsymbol{H}_r^{-1}\boldsymbol{\chi}_i. \tag{37}$$

Following the structure from the optimization problems earlier in the report the homogeneous transformation matrix $\boldsymbol{T}^{\mathrm{CCO}}$ from the robots camera sphere center to the world coordinates can be assumed known. Lets now define a new coordinate system called the beacon center, with a transform $\boldsymbol{T}^{\mathrm{BCO}}$ to the world coordinates. Origin of this coordinate system is placed in the plane of the beacons, i.e. the plane for which the homogeneous matrices are calibrated. Each beacon, i, can now be defined in the origin of their own coordinate system with the transform $\boldsymbol{T}^{\mathrm{BiBC}}$ to the beacon center. Given that each transformation matrix is defined as

$$\boldsymbol{T}^{\mathrm{j}} = \left[\begin{array}{cc} \boldsymbol{R}^{\mathrm{j}} & \boldsymbol{t}^{\mathrm{j}} \\ \mathbf{0} & 1 \end{array} \right] \quad and \quad \{\boldsymbol{R}^{\mathrm{j}}\}^{-1} = \{\boldsymbol{R}^{\mathrm{j}}\}^{T}. \tag{38}$$

The world coordinates for beacon i can be calculated as

$$\boldsymbol{t}_i^{\mathrm{world}} = -\big(\boldsymbol{R}^{\mathrm{BCO}}\boldsymbol{R}^{\mathrm{BiBC}}\big)^T\big(\boldsymbol{R}^{\mathrm{BiBC}}\boldsymbol{t}^{\mathrm{BCO}} + \boldsymbol{t}^{\mathrm{BiBC}}\big). \tag{39}$$

The beacons are expected to be located in the calibrated plane and assuming that the world coordinate system is defined such that the floor is parallel with the x-y plane Equation 37 can be expanded to

$$\lambda\boldsymbol{\rho}'_{r,i} = \boldsymbol{H}_r^{-1} \left[\begin{array}{cc} \boldsymbol{I}_{2x2} & 0 \\ \mathbf{0}_{1x2} & \frac{1}{a} \end{array} \right] \boldsymbol{t}_i^{\mathrm{world}} \quad \text{where} \quad a = (0,0,1)\cdot\boldsymbol{t}_i^{\mathrm{world}}, \tag{40}$$

where $\boldsymbol{I}_{\mathrm{2x2}}$ is the 2×2 identity matrix and $\mathbf{0}_{\mathrm{1x2}}$ is an 1×2 array of zeros. In this project two beacons are used, the center to center distance is measured manually to d^{cc} and the beacons center is defined as the midpoint between them with the z-axis parallel to the line going through both beacons and facing the robots direction. Their position in the beacon center coordinate system is thus

$$\boldsymbol{b}_{front}^{\mathrm{BC}} = (0,0,\frac{d^{\mathrm{cc}}}{2})^T \quad \text{and} \quad \boldsymbol{b}_{rear}^{\mathrm{BC}} = (0,0,-\frac{d^{cc}}{2})^T. \tag{41}$$

Equation 39 is thereby reduced to

$$\boldsymbol{t}_i^{\mathrm{world}} = -\big(\boldsymbol{R}^{\mathrm{BCO}}\big)^T\big(\boldsymbol{t}^{\mathrm{BCO}} + \boldsymbol{b}_i^{\mathrm{BC}}\big), \quad i \in \boldsymbol{I} = \{\text{front}, \text{rear}\}. \tag{42}$$

With this knowledge of how to reproject the beacons to the roof camera images the cameras relative pose calibration from Equation 15 can now be extended to include the calibration of the beacon center in relation to the cameras

$$\underset{\boldsymbol{O}}{\operatorname{argmin}} \sum_{f=1}^{F} \left(\sum_{c=1}^{C}\sum_{n=1}^{N} v_{f,c,n} \left\| \tilde{\boldsymbol{x}}_{f,c,n} - \boldsymbol{x}'_{f,c,n} \right\|_2^2 + \sum_{r=1}^{R}\sum_{i\in\boldsymbol{I}} v_{r,i,f} \left\| \tilde{\boldsymbol{\rho}}_{r,i,f} - \boldsymbol{\rho}_{r,i,f} \right\|_2^2 \right). \tag{43}$$

2.5.5 Using pose estimation to locate point clouds

With two beacons on the robot the full 2D pose, i.e translation and rotation, can be found. Multiple roof cameras may or may not overlap to average out detection noise. The method for locating points in a point cloud can now be done with the same minimization as in Equation 43 but with the optimization vector described in Equation 28. The bundle adjustment consist of the same optimization regardless if the point cloud is totally unknown, partly unknown or fully known and no matter if the roof cameras have located the beacons or not. Differences appear only in the initial guess of robot pose. If at least one roof camera detects the robot its estimation can be used as initial guess of heading and position, otherwise as before an arbitrary guess is used which might result in a false minimum during bundle adjustment. How this problem is avoided if no roof camera readings are available is described in the next chapter covering use of the robots relative movement.

3 Deduced reckoning

Deduced reckoning is a method used to estimate position and heading. In the robot used in this project pulse encoders mounted in the wheels measure the amount of rotation. By integrating the rotation, the movement can be calculated as the robots rotation and the distance traveled when the diameter of the wheels and distance between left and right wheel are known. By initializing the first location to either a known pose or to the zero vector all thereafter following poses can be estimated. Noise and errors in measurements are integrated and thus the error in pose estimation increases with the distance traveled, this is called drift. The drift can be corrected by looking at the relations of spatially fixed points seen from multiple locations. Especially a closed loop, where some of the points seen from the first location also is seen from the last location is good and can eliminate the drift altogether.

3.1 Chain method

The actual path traveled between the locations used for mapping is disregarded and the move between two on each other following poses can be described as a distance, Δ, along a straight line specified by an angle, α, from the last heading together with a post move rotation specified as an angle γ measured from the heading at the last location. By using this chain-method which minimizes the error distance and angles form the last location instead of distance and angle from the origin or first location the effect of the drift is minimized.

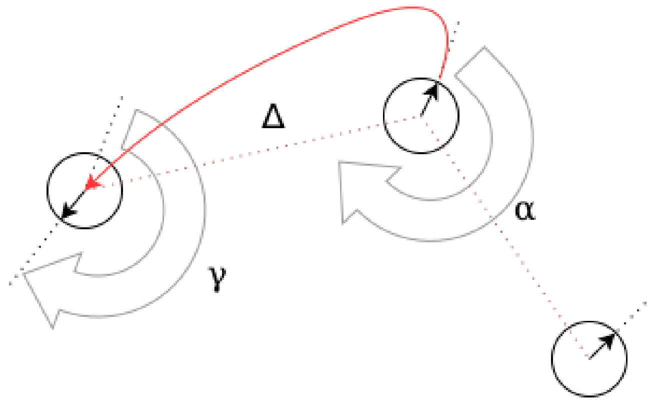


Figure 8: The deduced reckoning can be simplified to a chain where the move between two poses is described by three parameters, the shortest distance between poses Δ, the angle between last pose and direction of movement α and the angle between last pose and current pose γ. Red line symbolizes actual movement, dashed red line is the shortest distance between locations.

3.2 Extending world estimation equation

The cost function used to find the most probable locations of the robot and the price labels can now be modified, based on Equation 43. Let's define $\boldsymbol{d}_{i,j} = (\Delta_{i,j}, \alpha_{i,j}, \gamma_{i,j})^T$ as the three parameters describing the move from location j to i estimated with the deduced reckoning and $\tilde{\boldsymbol{d}}_{i,j}$ as the corresponding distance and angles calculated from the estimated poses of the robot. The optimization vector $\boldsymbol{O}$ is the same as described in Equation 28 containing the poses of the robot and the location of the shelf labels. Including the error between the measured and expected parameters for deduced reckoning the cost function can be written as

$$\begin{aligned}\underset{\boldsymbol{O}}{\operatorname{argmin}}\Bigg(a \sum_{f=1}^{F} \Bigg(& \sum_{c=1}^{C}\sum_{n=1}^{N} v_{f,c,n} \left\| \tilde{\boldsymbol{x}}_{f,c,n} - \boldsymbol{x}'_{f,c,n} \right\|_2^2 \\ & + \sum_{r=1}^{R}\sum_{i\in \boldsymbol{I}} v_{r,i,f} \left\| \tilde{\boldsymbol{\rho}}_{r,i,f} - \boldsymbol{\rho}_{r,i,f} \right\|_2^2 \Bigg) \\ & + \sum_{f=2}^{F} \left\| \left(\tilde{\boldsymbol{d}}_{f,f-1} - \boldsymbol{d}_{f,f-1} \right) \circ (b,c,d)^T \right\|_2^2 \Bigg),\end{aligned} \tag{44}$$

where the $\circ$ operator represents the Hadamard product, also known as entry-wise multiplication. Weights a, b, c and d are introduced as a result of the use of different units.

3.3 Estimating weights and unit correspondence

Reprojection error is measured in pixels while the deduced reckoning is measured in world units: millimeters and radians. There is no straight forward way to calculate the weight factors because the world units maps to a different amount of pixels in different cameras and in different locations in a camera's image. Different sensors also have different amount of measurement error which should affect the choice of weights.

A way of doing an estimation of the unit correspondence is to take the average scale factor of the mapping between the calibrated plane where the robot moves and its projection in the roof cameras. To do this lets define the point $\boldsymbol{a}$ in the calibrated plane as $\boldsymbol{a} = (x, y, 1)^T$ and further two points one world unit away from $\boldsymbol{a}$ along perpendicular axes, $\boldsymbol{b} = (x+1, y, 1)^T$ and $\boldsymbol{c} = (x, y+1, 1)^T$. As before R is the total number of roof mounted cameras and $\boldsymbol{H}_r$ is the homography transformation matrix from camera r to the calibrated plane. Let's also introduce $\boldsymbol{A}_r$ as the area of the region of interest in the calibrated plane seen by camera r. The ratio between the pixel unit weight factor, a, and the millimeter unit weight, b, is then described as

$$\frac{a}{b} = \frac{1}{R \cdot \boldsymbol{A}_r} \sum_{r=1}^{R} \iint_{\boldsymbol{A}_r} \frac{\left\|\boldsymbol{a} - \boldsymbol{b}\right\|_2 + \left\|\boldsymbol{a} - \boldsymbol{c}\right\|_2}{\left\|\boldsymbol{H}_r^{-1}\boldsymbol{a} - \boldsymbol{H}_r^{-1}\boldsymbol{b}\right\|_2 + \left\|\boldsymbol{H}_r^{-1}\boldsymbol{a} - \boldsymbol{H}_r^{-1}\boldsymbol{c}\right\|_2} \mathrm{d}x\mathrm{d}y. \tag{45}$$

To avoid calculating the integrals the area can be discretized into a point set in $\mathbb{R}^2$ and calculated as the average of a double sum.

The weights also represent the trust in each measurement. It can be expected that the camera detections have considerably higher accuracy than the deduced reckoning even if the resolution normalization is performed according to Equation 45. The trust has to be either estimated or measured. By measuring the standard deviation for each sensor and dividing the weight constants with this number good weight factors can be achieved. If the standard deviation approach is taken then there is no need to perform the resolution normalization, if the ratio is not of special interest.

3.4 Estimating increasing parts of the world

In the case where the drift is of such a magnitude resulting in false solutions because of local minima, it can be required that locations of the robot are estimated using increasing subsets of all used locations to yield more accurate initial guesses. This can be achieved by running the optimization $F^{\text{tot}} - 1$ times where F^{tot} is the total amount of locations from where the mapping has been done. By defining L as the number of locations used the optimization can be run with the locations $f \in \{1...L\}$ where L is decided each optimization cycle as $L \in \{2...F^{\text{tot}}\}$ and the initial guess of the pose for location f is the estimation from the previous optimization if $f < L$ and $L > 2$ otherwise the estimation from the deduced reckoning.

4 Detecting flashing LEDs

The robots beacons consist of two LEDs flashing, each with a constant frequency. The digital price labels are also detected by their flashing LEDs. These consist of a flash with two main peaks in the frequency spectrum. The LEDs are detected by finding triggers based on high temporal derivative change in the light levels using a sequence of images. The triggers are then filtered by their frequency containment to find flashing with the correct frequency and suppress other detections.

4.1 Segmentation of high temporal derivative areas

Each camera frame is examined for pixel regions with fast changing light levels. This is done by image segmentation based on each pixels derivative in the time domain. The time derivative of a pixel $p_i(t)$ at location i at time t is the change in intensity divided by the change in time

$$\frac{\mathrm{d}p_i}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{p_i(t + \Delta t) - p_i(t)}{\Delta t}. \tag{46}$$

This can be estimated, for pixel $p_{i,f}$ where i is the image coordinate and f is the frame number, using the backward difference

$$\frac{\mathrm{d}p_{i,f}}{\mathrm{d}f} \approx \frac{p_{i,f} - p_{i,f-1}}{\lambda_{\text{fps}}}, \tag{47}$$

where λ_{fps} is the time between the frames. The segmentation is achieved by thresholding the derivatives magnitude for each pixel in the image and assigning new values to the segmented images pixels $s_{i,f}$ given by

$$s_{i,f} = \begin{cases} |\frac{\mathrm{d}p_{i,f}}{\mathrm{d}f}| > t_{\text{dp}} & : p_{i,f}, \\ \text{else} & : 0. \end{cases} \tag{48}$$

The segments are filtered on morphological parameters such as the segments total area in the frame and the segments length to area ratio. The search and filtering of these regions are further described in the *6 Implementation* chapter of the report. Such a pixel region detected will from now on be called a trigger. A trigger $\boldsymbol{d}$ in a frame is described by its centroids image coordinates, u, v, and the frame number,f, it appeared in

$$\boldsymbol{d} = (u, v, f)^T. \tag{49}$$

Let the set $\boldsymbol{D}_p$ contain each trigger extracted with a position label p, as later described, from all frames

$$\boldsymbol{D}_p = \{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_I\}, \quad \boldsymbol{D}_p \subseteq \boldsymbol{F}_p \subseteq \boldsymbol{F}, \tag{50}$$

where $\boldsymbol{F}_p$ is the set with all possible triggers with this position label and $\boldsymbol{F}$ is the set with all possible triggers anywhere in any frame. If the L_2 norm of the difference in image coordinates of two detections is smaller than a threshold, t_{d}, they are assigned the same position label p, if $\tilde{\boldsymbol{d}}_a$ denotes the two first elements in $\boldsymbol{d}_a$, i.e the image coordinate

$$\left\|\tilde{\boldsymbol{d}}_a - \tilde{\boldsymbol{d}}_b\right\|_2 \leq t_{\text{d}} \Rightarrow p_a = p_b. \tag{51}$$

This labeling of the positions where a certain spread is allowed can be seen as an agglomerative hierarchical clustering using centroid linkage. The exception is that two clusters walking towards each other never can be joined because the clustering is only performed over the new points as they get detected. It could also be seen as k-nearest neighbors with $k = 1$ and the extra condition that a distance above the threshold t_d creates a new cluster.

4.2 Frequency spectrum filtering

The set $\boldsymbol{E}_p$ contains rising edges with position label p. A rising edge e_j is defined as the frame number, a, where a trigger is detected in frame a but not with the same position label in the previous frame, $a - 1$

$$\boldsymbol{E}_p = \{e_1, e_2, ..., e_J\}, \quad \boldsymbol{E}_p \subseteq \hat{\boldsymbol{D}}_p, \tag{52}$$

where $\hat{\boldsymbol{D}}_p$ is the set containing the last elements, f_i, from each vector $\boldsymbol{d}_i$ in $\boldsymbol{D}_p$. The time between two rising edges can now be calculated as

$$t_{b,a} = \frac{e_b - e_a}{f_{\text{fps}}}, \tag{53}$$

where f_{fps} is the number of frames per second. A vector with the time passed between each rising edge in location p can thus be created

$$\boldsymbol{T}_p = \{t_{2,1}, t_{3,2}, ..., t_{J,J-1}\}. \tag{54}$$

When the frequency spectrum of the flashing lights is known a cost function with a minimum in $\lambda_{\text{flash}} = 1/f_{\text{flash}}$ can be defined where f_{flash} is the expected frequency of the light flashing. One such cost function is an upper limited triangular wave described by

$$\begin{aligned} c_\lambda(t) &= \min(a, \max(-kt + m_1, kt + m_2)), \\ k &= \frac{m_1 - m_2}{2\lambda}. \end{aligned} \tag{55}$$

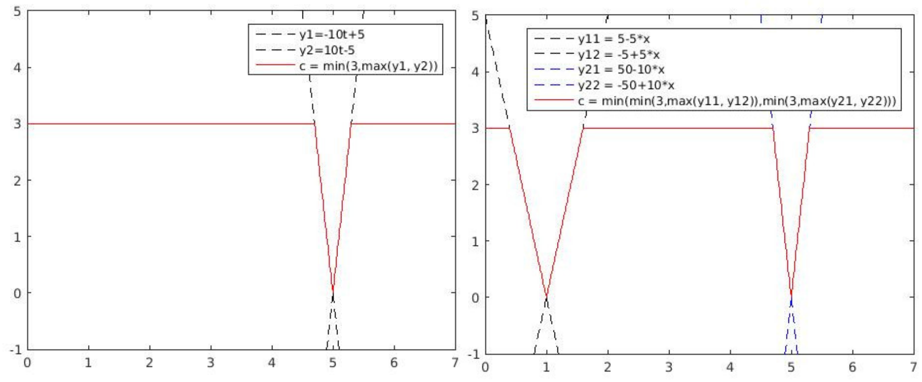


Figure 9: Left diagram represents a cost function with a minimum in $f = 5$Hz, right image is the minimum value of a cost function with a minimum in $f = 5$Hz and one with a minimum in $f = 1$Hz. The minimum at $f = 5$Hz is represented by $k = 10$ and the minimum at $f = 1$Hz by $k = 5$.

The choice of parameters m_1 and m_2 will decide the tolerance of noise in measurements. If the spectrum is known to contain multiple local maximums in the frequency range several cost functions can be combined as

$$c_{\lambda_1,\lambda_2,...,\lambda_i}(t) = \min(c_{\lambda_1}(t), c_{\lambda_2}(t), ..., c_{\lambda_i}(t)). \tag{56}$$

The sum of the cost function applied to each element in the time vector $\boldsymbol{T}_p$ yields a scalar cost value which indicates the flash characteristics similarity to the expected pattern. The triggers can be assumed to indicate a true detection in the location p if the scalar cost value is below a threshold, t_{cost}, and the time vector $\boldsymbol{T}_p$'s cardinality is larger than a value $s_{\min}$

$$\sum_{\boldsymbol{T}_p} c_{\lambda_1,\lambda_2,\ldots,\lambda_i}(t_i) < t_{\text{cost}} \quad \mathit{and} \quad \overline{\overline{\boldsymbol{T}_p}} > s_{\min} \quad \Rightarrow \quad p \in \boldsymbol{D}_{conf}, \tag{57}$$

where the vector $\boldsymbol{D}_{conf}$ includes the location of all confirmed flashes.

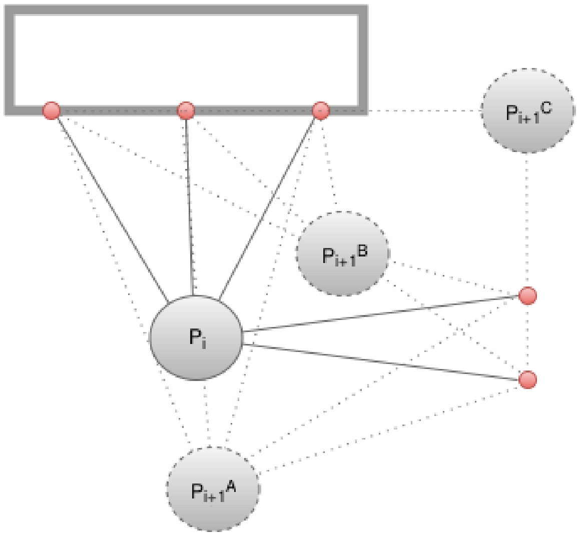
5 Navigation and route planing

If the mapping should be possible without human input the robot must be able to navigate in the stores autonomously. Acting based on the environment is something that all lifeforms do. Simple creatures often do this by stimulus-response mechanism while more advanced creatures plan ahead. For the interested reader the book by *Balkenius, Christian* about natural intelligence in artificial creatures [27] is highly recommended. Although simple stimulus-response models could be used using as an example the robots distance sensors and tactile sensors to move around more or less randomly more advanced algorithms will increase the speed and performance of the objective. By being able to predict the future and plan ahead the robot can optimize its path through the environment.

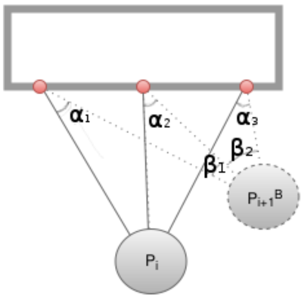
Seen as a planning problem this situation differs from the common case where all information is previously known. The layout of the environment is not fully known at the time of planning and the planning does not need to fully finish before the actions are started. Using these assumptions the planning can be done without overly constrained plans. Research has been performed on planning systems not only with a belief, desire and intention but also the ability to reflect upon its internal representation of these states [46]. During the mapping of the environment more and better information will be available as time moves on. Based on this there is no need to take a decision before it actually needs to be taken. Instead of making guesses of the whole path to take based on inadequate information only the next move and the decisions that it is dependent on may be taken.

5.1 Force field navigation

A nontrivial decision is where to define the boundaries of what decisions are needed to decide the next step. If too little information is included the possibility to get trapped in a dead end is higher. In this project the path is not optimized to be as short as possible but rather to include the best views possible during as few stops as feasible. Two positions yielding measures with low delta angles or in other way containing little information should be avoided. In Figure 10 a) three possible positions are visualized, two of them yielding little new information because of the weak angles. The angles to maximize is the angle between current locations observation and next locations observation of each label and the angle difference of the observations of all labels from the next location as seen in Figure 10 b).



(a) With the current location P_i a number of moves are possible. The viewing angles of the labels will be different. At location P_{i+1}^{C} the robot camera only see two price labels, while the others are collinear.



(b) The angles to be optimized are the viewing angles seen from the robots next position, β, and the angles from each label to the robots current location and the next location, α.

Figure 10: The planning of the robot path can be done by maximizing viewing angles to the digital labels. The price labels are represented by the red dots and the current robot position and possible moves as gray circles.

With the current position of the robot $\boldsymbol{P}_i$, J price labels with positions given by $\boldsymbol{p}_j$ and the intended next position $\boldsymbol{P}_{i+1}$ the internal angles, β, can be calculated as

$$\beta_j = \angle(\boldsymbol{P}_{i+1} - \boldsymbol{p}_{j+1}) - \angle(\boldsymbol{P}_{i+1} - \boldsymbol{p}_j), \quad j \in \{1 ... J-1\}, \tag{58}$$

where $\angle(\boldsymbol{v})$ represent the angle of the vector $\boldsymbol{v}$. The external angles, α, can be decided by

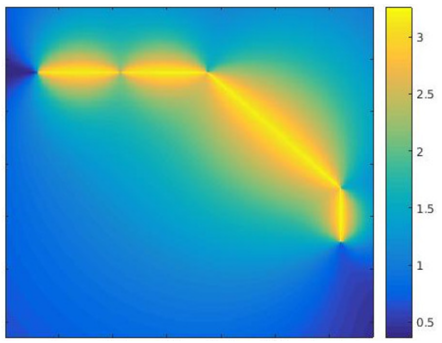
$$\alpha_j = \angle(\boldsymbol{p}_j - \boldsymbol{P}_{i+1}) - \angle(\boldsymbol{p}_j - \boldsymbol{P}_i), \quad j \in \{1 ... J\}. \tag{59}$$

Two models to represent the planning in a map are common, either as a graph search or as a potential field. An example of a force field using the potential field method can be seen in Figure 11 d) where the arrows points in the direction maximizing the angles. Heat-maps where the color represent the positions' root mean square of all angles are represented in Figure 11 a) to c). Every pixels color is calculated as

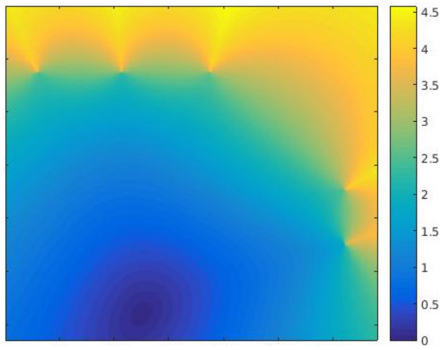
$$c = \sqrt{\sum_{j=1}^{J} \alpha_j^2 + \sum_{j=1}^{J-1} \beta_j^2}. \tag{60}$$

A heat-map representing the goodness of each position is created by calculating the c-value for each and every position. Using the same layout as in Figure 10

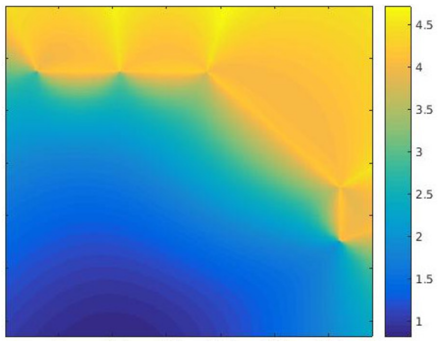
a) the heat-maps for internal angle β, external angle α and the sum of them can be found as visualized in Figure 11 a)-d).



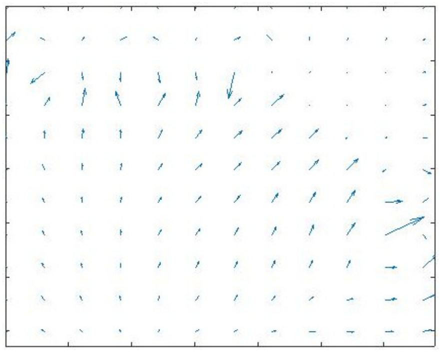
(a) RMS of internal angles, i.e. the robot's difference in angle to the different labels.



(b) RMS of external angles, i.e. the angle between each labels ray to the robot's current position and the next robot position.



(c) RMS of angles from a) and b) combined visualizing the total goodness of each position.



(d) The force field indicating the derivative of the total sum of squares. The robot should follow the arrows as much as possible when planning the route.

Figure 11: Heat-map and force field maximizing the viewing angles to the labels using the positions as in figure 10 a).

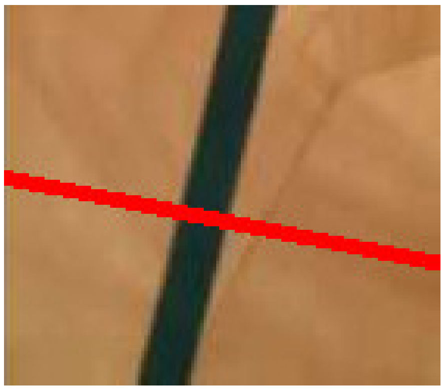
5.2 Prefixed line navigation

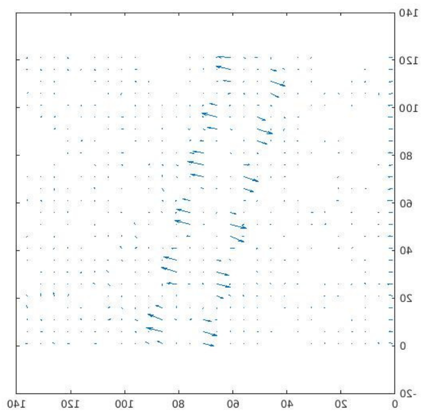
A simpler approach resulting in easier to predict moving patterns is to use a predefined path. To simplify the use and installation of the robotic system the path can be defined by applying a high contrast line on the floor which the robot is supposed to follow. Many line following algorithms have been proposed but many of them are sensitive to noise or textures in the background area. Many competitions are held worldwide in line following where speed and algorithm robustness is tested. Usually the line following is done using a set of reflex detectors resulting in an ultra-low resolution one dimensional image of the line under the robot. During the time of this project a robust technique utilizing the already installed cameras on the robot has been developed. The novel approach developed for line detection using visual detection developed has also been described in a paper submitted for publication [42] during the time of the project. The new algorithm developed is based in the fundamental steps of Stroke Width Transform [41]. The Stroke Width Transform turned out to be a revolution of the OCR text detection in natural images when it was published in 2010.

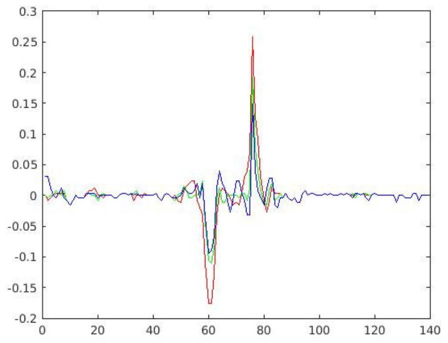
5.2.1 Defining a line

In this project the algorithm has been modified to be considerably less process demanding to allow real time analysis at high frame rate. Two assumptions inspired from the Stroke Width Transform text detection algorithm have been used to detect the line. A line is bordered by a high derivative stroke on each side, the strokes are assumed to be on approximately the same distance from each other, i.e. the width of the line is constant. The gradients of the two borders are assumed to have adverse directions, i.e. the borders are parallel to each other and the background on both sides of the line are either brighter or darker than the line itself.

In Figure 12 a) and c)-d) it can be seen that simply thresholding gradient magnitude in the image can in some cases be enough to find a dark line on a brighter colored background. The same is true for the inverted case where the line has a brighter color than the floor. In many cases other objects and shadows can be hard to differentiate from the line using only this technique. Combining this with filtering of stroke width and edge gradient direction can however make the method robust. The stroke width is the distance between the two opposite peaks of the derivative. This can be assumed to be approximately the same along the length of the line if the perspective transform is taken into account. In Figure 12 b) the gradients magnitudes and direction can be seen. The gradient on opposite sides of the line can be seen to be at approximately 180° angle in relation to each other. The line will from hereon be assumed to have a darker color than the surroundings all tough the opposite case can be analyzed by just moving along the positive direction of the gradient instead of the negative. It is also possible to detect both light and dark lines using this proposed algorithm by running it twice, once in each direction.


(a) Closeup of a black line on a slightly textured floor board. Red line indicating perpendicular axis to the line.


(b) The gradients can clearly be seen to point in approximately adverse directions on the opposite sides of the line.


(c) The derivative along the red axis in image a) for red, green and blue channel.

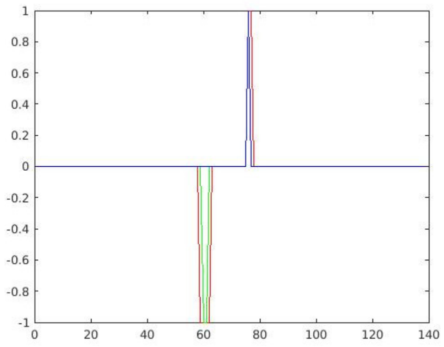

(d) Thresholded derivative clearly indicated line borders.

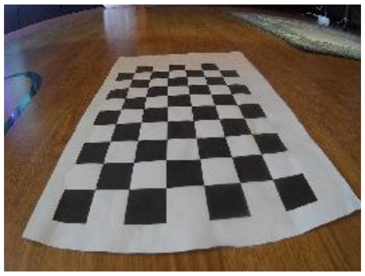
Figure 12: A dark line on a lighter colored floor board can easily be found using the gradients magnitude.

5.2.2 Top view perspective transform

A property of the perspective transform is that parallel lines do not always project to parallel lines. In the case where the camera's image plane is not parallel with the ground plane there will be a perspective transform, seen as the depth in the image, resulting in the edges of a parallel line going towards the horizon being non parallel. Using the assumption that the edges of a line are parallel a top-down view must be used.

By calibrating the position of the camera in relation to the floor plane in the same manner as described in section *2.5.1 Finding homography transforms* the top view can be calculated by pixel remapping using the homography matrix, as seen in Figure 13. The calibration can be done using a chessboard pattern placed in the ground plane.

Performing the calculations in the top view image ensures that the width of the line does not get affected by the perspective transform as the lines goes towards the horizon line.


(a) Raw image from robot camera feed. Chessboard pattern is used to find perspective warp (homography) from ground to image plane.

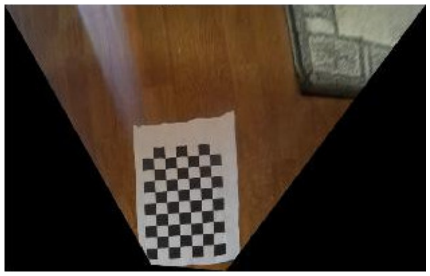

(b) Undistorted and perspective wrapped image from raw image in (a). A top-down view of the calibrated plane is achieved.

Figure 13: Frames from robot camera feed are subjects to a distortion and a perspective warp.

5.2.3 Finding the lines

The lines can be found by first calculating the gradients and their magnitude in a frame. Pixels with a gradient magnitude above a threshold value are in this stage assumed to be pixels on the lines borders. These are split into two groups: start pixels and end pixels. The first group is defined by a negative value in column-wise gradient. For each start pixel an iteration in the image is performed. If the start pixels gradient is given by α then dark lines on brighter background are found by iterating along the direction of $-\alpha$ while brighter lines on darker background are found by iterating in the opposite direction, along α. The iteration is performed until a (non-zero) end pixel is reached. The moved distance and the angle between the start pixels gradient and the end pixels gradient is kept, these are the stroke width l and the stroke border angle difference δ

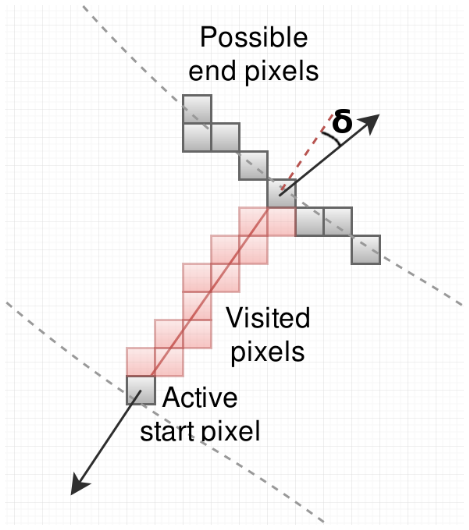


Figure 14: By following the border pixel in the opposite direction of its gradient the opposite bordering pixel is found. The angle δ can be assumed small if the lines edges are approximately parallel.

If the start pixel's gradient direction is α and the stop pixels gradient direction is β then $\delta = \alpha - \pi - \beta$. The angle δ can be assumed to be small if the borders of the line is approximately parallel. If the method is applied to every start pixel in the image an output image is created where the pixels in the line segments are color coded with their stroke width. Some outliers will be present but the true lines are characterized by their approximately constant stroke width and thereby color coding. For each pixel at $(i,j)^T$ in the output image marked with a stroke width value $l_{i,j}$ the number of neighbors with approximately the same value is counted. Only neighbors within a kernel with side length d, satisfying $d \in \{2\mathbb{N}+1\}$, around the pixel are counted and a maximum ratio r between the two pixels values are allowed. Each pixel can thus get a probability score $v_{i,j}$ defined as the normalized number of neighbors fulfilling the ratio test. If $\boldsymbol{P}_\mathrm{t}$ is the set of all neighboring pixels fulfilling the criteria and $(k,l)^T$ are all pixels contained in the kernel around $(i,j)^T$ then

$$\begin{aligned}
&\forall l_{i,j} \in \boldsymbol{P}_\mathrm{t} : \frac{1}{r} \leq \frac{l_{k,l}}{l_{i,j}} \leq r, \\
&k \in \{i + \Delta i : |\Delta i| \leq \frac{d-1}{2}, i + \Delta i \in \mathbb{N}\}, \\
&l \in \{j + \Delta j : |\Delta j| \leq \frac{d-1}{2}, j + \Delta j \in \mathbb{N}\}.
\end{aligned} \tag{61}$$

The probability score for each pixel is then defined as the normalized number of elements in $\boldsymbol{P}_\mathrm{t}$

$$v_{i,j} = \frac{\overline{\overline{\boldsymbol{P}_\mathrm{t}}}}{d^2}. \tag{62}$$

A similarity measure between pixels can be calculated in four dimensions where each pixel is described by the vector $\boldsymbol{p_i} = (i, j, l, \gamma)$ where i, j is the image coordinates in the top view image, l is the stroke width assigned to the pixel and γ is the angle of the stroke covering the pixel. The similarity of two points is calculated as

$$s_{a,b} = \frac{1}{\|\boldsymbol{p}_a - \boldsymbol{p}_b\|_2}. \tag{63}$$

A line can also be assumed to be considerably longer than its stroke width. This can be used to define that the ratio of the distance between the two points furthest apart and the mean stroke width should be above a threshold $t_{\text{l:w}}$. If $\tilde{\boldsymbol{p}}_i$ is the two first elements of $\boldsymbol{p}_i$, i.e. the image coordinates, $\boldsymbol{J}$ is the set of all strokes included in the line segment and $\overline{S}_{\text{width}}$ is the mean stroke width within the current line segment then

$$\frac{\max\limits_{a,b \in \boldsymbol{J}}(\|\tilde{\boldsymbol{p}}_a - \tilde{\boldsymbol{p}}_b\|_2)}{\overline{S}_{\text{width}}} > t_{\text{l:w}} \tag{64}$$

should be fulfilled if the line segment should be considered as a line instead of noise. To improve the calculation speed the distance between two points can be simplified to the approximation

$$d = \sqrt{dx^2 + dy^2} \approx \max(dx, dy). \tag{65}$$

If the image contains multiple line segments that are not connected they need to be clustered using a cluster algorithm, a method for this is described in the next section.

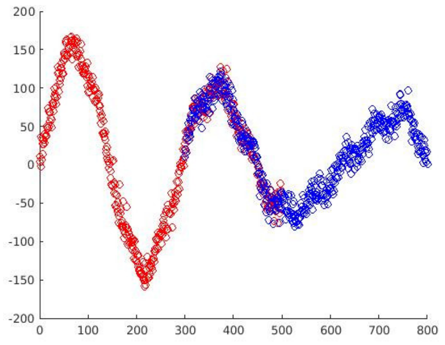
5.2.4 Internal representation and line clustering

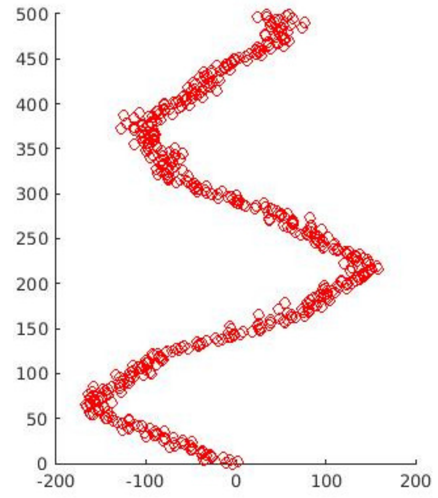
When the top down view of the line is known in the robots local coordinate system several views can be stitched together to a complete map in a fixed coordinate system. If the distance moved by the robot during one image frame is shorter than the distance seen by the camera the views of the lines will be overlapping. Many algorithms exist for registration of point clouds which can be applied in this case. A very common approach is Iterative Closest Point [30] where the transformation of two point clouds is calculated by finding the translation of the first cloud that will minimize the sum of the squared distances between points in the two clouds. Several implementations exist such as the improvement proposed by *Zhang, Zhengyou* [74] to speed up calculations and reduce impact of outliers.

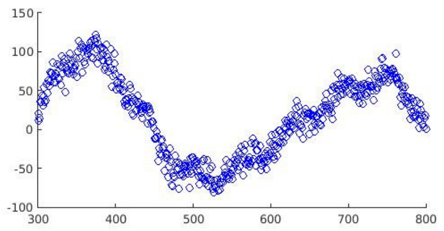
Given the iterative approach a better solution can be calculated faster and with less probability for false minimums if some a priori information is known. Such information can be achieved by using the deduced reckoning from the odometers on the robots wheels. The resulting transform between the two point clouds does not only allow the construction of a complete map of the environment but also yields a good estimation of the robots movement. In Figure 15 the stitching of two point clouds representing a partially overlapping top-down view of a line is visualized.

If several non continuous lines are seen in the image they need to be differentiated. This can be done using clustering. The robot needs to know in every frame which line to follow and this can be done using the transform gotten from the point registration in combination with the a priori knowledge of which line it followed in the last frame. When the robot knows which line it was following in the last frame this transform can be used to find the corresponding line in the current frame.

Using the descriptive vectors defined in Equation 61 the points can be grouped into segments using a clustering algorithm. Each segment represent one view of a continuous line. The lines form long, narrow segments that can partly encircle each other, thus center-based algorithms, such as k-means [70], will not work. A density based algorithm such as DBSCAN [72] is instead proposed. DBSCAN will not cluster all elements but instead filter away low density areas as outliers, a functionality which is appreciated in this context. Many disturbances in the image will be detected and some will pass through the filtering based on the stroke width of the closest neighbors described in *5.2.3 Finding the lines*. The noise left will have inconsistent direction in relation to their neighbors and will thus be removed as outliers.


(a) Two top-down views of pixels detected on two partly overlapping lines.


(b) View of the first line in a local coordinate system


(c) View of the second line in a local coordinate system.

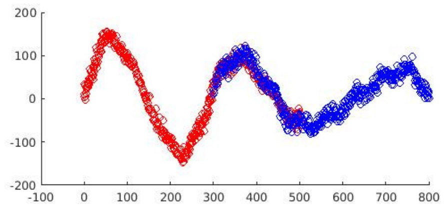

(d) Both lines transformed and stitched together using Iterative Closest Point on the overlapping sector.

Figure 15: By using a registration algorithm such as Iterative Closest Point (ICP) two views of lines can be stitched together to a complete shape. Figure a) shows the true relative positions while d) shows the result of stitching based on the points in Figure b) and c).

6 Implementation

The following section covers all hardware and software designed during the project to allow evaluation of the methods proposed. First the construction of the mobile robotic platform is described. The purpose of the platform is to be able to scan the store environment visually using the omnidirectional assembly of cameras. This needs to be done from several locations which is why the robot needs to be able to move but also have spacial awareness. No focus have been aimed toward constructing a robot suitable for industrial use but instead to allow easy experimentation. In the final implementation designed for flow production the focus should be reduction of cost, dependability and ease of use. The robotic platform designed in this section is made for modularity, easy modification and use of predesigned and available components.

The experimental environment designed and used during the project is also explained. The main focus is to replicate the environment in a retail store and evaluate problem prone areas. The environment is built in a room at Cognimatics office in Lund. The system has also been evaluated in a real store environment. The focus of the real store test was to investigate the impact of larger loops in the navigation and position estimation and possible problems or drawbacks with label detection in crowded environments.

Implementation of custom software is done in several platforms to allow all parts of the network based system to cooperate. Custom software is implemented in the cameras mounted on the mobile platform to detect flashing price labels and stream video, in the mobile platforms computer to allow remote control and telemetry, in the control center server to bind processes together and implement a Human Computer Interface (HCI) using a Grapical User Interface (GUI). Other areas such as the Pricer Electronic Shelf Label system is also combined with custom software. Some software was designed and used as tools to create other software and does not take part in the final use of the system. One such software is the *Data analytics and detector evaluation software* described in the section which was used to collect video sequences from the robot and evaluate the different versions of the label detectors on the exact same data to value repeatability.

6.1 Experimental hardware platform

The robotic platform built in the project is based on an IRobot Roomba 621 robotic vacuum cleaner [8]. The robot has been extended with hardware required for the project including but not limited to

1. Four Modcam [13] wide angle cameras
2. Two LED beacons flashing with different frequency
3. Raspberry Pi 3 [19] compact computer

The robotic vacuum cleaner was chosen as a base since it combines the motors, rotary wheel encoders and proximity sensors with an open serial interface. The serial interface allows full control of the motors and all sensor data is available on request or as a constant stream. The serial protocol was discovered not to be compatible with 4 volt signals and thereby neither the 3.3 volts used by the Raspberry Pi. The Raspberry Pi input pins are not tolerant with 5 volt outputs such as those on the robot's interface. To interface the robot with the Raspberry Pi a FTDI chip [6] was used to add an UART serial port to the USB bus. The data lines of the FTDI chip was interfaced with the robot by using inverting connections with NPN channel transistors according to Figure 16.

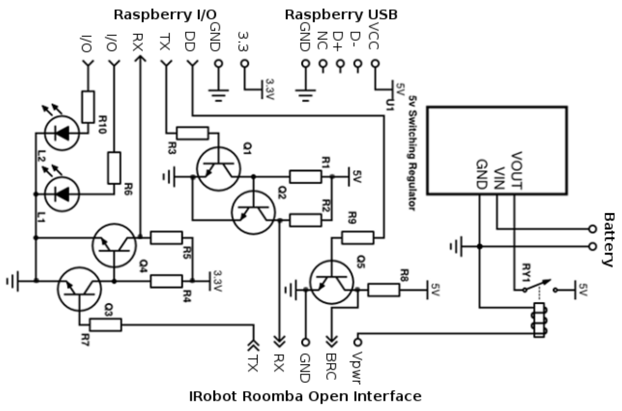


Figure 16: Connection interface between Raspberry Pi and Irobot Roomba Open Interface using double inverting transistor connections to transform signal voltage. Two LEDs used as beacons for robot location estimates are also connected to the Raspberry Pi I/O.

From the data sheet for the BC547 transistors used [3] the characteristics can be found

$$h_{\text{FE,min}} = 110 \quad U_{\text{CE,sat}} = 90\text{mV} \quad U_{\text{BE,sat}} = 700\text{mV}.$$

Setting the values

$$R_2 = R_4 = R_8 = 1\text{k}\Omega$$

yields collector currents given by

$$I_{\text{c}} = \frac{\text{Vdd} - U_{\text{CE,sat}}}{\text{R}} \Rightarrow \qquad \begin{aligned} I_{\text{c,Q}_2} = I_{\text{c,Q}_5} &= \frac{5\text{V} - 90\text{mV}}{1\text{k}\Omega} = 4.91\text{mA}, \\ I_{\text{c,Q}_3} &= \frac{3.3\text{V} - 90\text{mV}}{1\text{k}\Omega} = 3.21\text{mA}. \end{aligned}$$

Using the h-parameter model the needed base current can be calculated as

$$I_{\text{b}} \geq \frac{I_{\text{c}}}{h_{\text{FE,min}}} \Rightarrow \qquad \begin{aligned} I_{\text{b,Q}_2} = I_{\text{b,Q}_5} &\geq \frac{4.91\text{mA}}{110} = 44.6\mu\text{A}, \\ I_{\text{b,Q}_3} &\geq \frac{3.21\text{mA}}{110} = 29.18\mu\text{A}. \end{aligned}$$

the base resistors are then specified by

$$R_\mathrm{b} \leq \frac{\mathrm{Vdd} - U_\mathrm{BE,sat}}{I_\mathrm{b}} \Rightarrow \qquad \begin{aligned} R_{\mathrm{b,Q_2}} = R_{\mathrm{b,Q_5}} &\geq \frac{5\mathrm{V} - 700\mathrm{mV}}{44.6\mu\mathrm{A}} = 96.4\mathrm{k}\Omega, \\ R_{\mathrm{b,Q_3}} &\geq \frac{3.21\mathrm{mA}}{29.18\mu\mathrm{A}} = 147.3\mathrm{k}\Omega. \end{aligned}$$

To allow fast saturation and less sensitivity to noise all base resistors can be chosen to $R_\mathrm{b} = 10\mathrm{k}\Omega$ this will result in base-emitter currents given by

$$I_\mathrm{b} = \frac{\mathrm{Vdd} - U_\mathrm{BE,sat}}{R_\mathrm{b}} \Rightarrow \qquad \begin{aligned} I_{\mathrm{b,Q_2}} = I_{\mathrm{b,Q_5}} &\geq \frac{5\mathrm{V} - 700\mathrm{mV}}{10\mathrm{k}\Omega} = 0.43\mathrm{mA}, \\ I_{\mathrm{b,Q_3}} &\geq \frac{3.3\mathrm{V} - 700\mathrm{mV}}{10\mathrm{k}\Omega} = 0.26\mathrm{mA}. \end{aligned}$$

The LEDs are driven directly from the Raspberry I/O pins. The SoC is able to source a recommended maximum of 16mA per I/O pin and thus the resistors to the LEDs are given by

$$R_\mathrm{LED} = \frac{\mathrm{U_{IO}} - U_\mathrm{LED}}{I_\mathrm{LED}} \quad \Rightarrow \quad R_6 = R_{10} = \frac{3.3\mathrm{V} - 1.8\mathrm{V}}{16\mathrm{mA}} = 93.75\Omega \approx 100\Omega.$$

The serial receive and transmit lines have double inverters to turn the signal back to non inverted state, this is because the UART module does not support inverted signals. Device detect can be inverted in Raspberry Pi software and therefore it is constructed with only a single inverting coupling. The inverting transistor modules could be replaced by opto-isolators which might be both easier to implement and a more robust solution. The beacons used to detect the robot in the roof camera images were build by connecting two LEDs to the Raspberry Pi's I/O pins. The LEDs are mounted on a black non reflective surface to avoid detection of surface reflections. The LEDs are set in software to flash at different frequencies so that the front and rear LED can be identified.

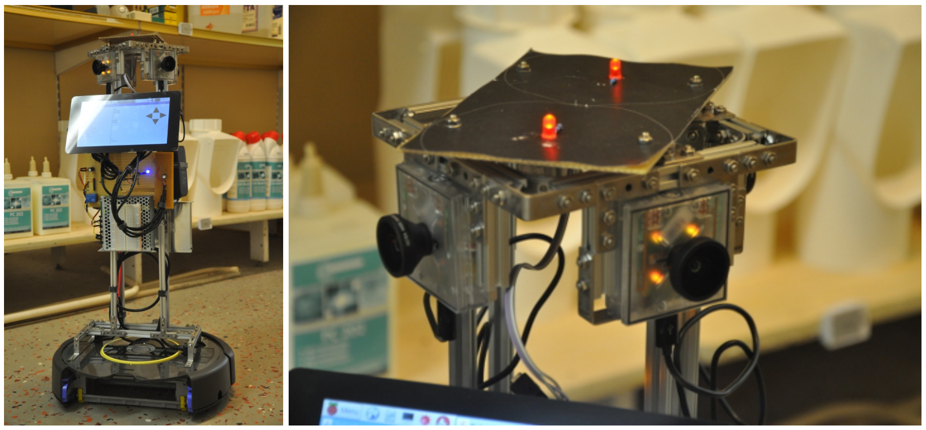


Figure 17: The robotic platform constructed during the project is seen in the pictures. On the top of the robot the two red beacons are seen, flashing at different frequency. Under the beacons are four Modcam wide angle cameras mounted in an aluminum frame. A Raspberry Pi 3 with LCD screen, two battery packs, switching regulator, converter board and robotic vacuum cleaner base can also be seen.

The cameras are mounted on an aluminum profile structure to achieve as close as perpendicular angles to its neighbors and the horizontal plane as possible. The Raspberry Pi has been equipped with a 7 inch LCD screen to allow instant feed back from software. Using a switched 5VDC out DC/DC converter the Raspberry Pi and screen are powered by the robots battery. The DIN-contact on the robot features a voltage output to power external utilities. This output was found to be limited by a poly-fuse to a far lower current than what was necessary to power the Raspberry Pi. This was solved by connecting the power converter directly to the robots battery. However this introduced problems with the charging. The working theory is that the robot measures the current drawn from the battery and from its integral decides if there is a need to charge the battery or not. The result is that drawing current directly from the battery prevents the robot from initializing the charging cycle as it is needed. Because of this the cameras were externally powered from two USB power banks to reduce the current drawn from the robots battery. The result is several hours battery life for the robot and more than a full working days battery life for the cameras. The Modcam cameras also includes an internal battery, albeit the battery life is rather short.

6.2 Environments for evaluation and testing

Two environments have been used for the development, testing and evaluation of the project. The first is an in-office resemble of a store environment and the second used for complementary final testing is a warehouse, former food store.

6.2.1 Experimental environment

In addition to the real store test, described later, the system has been tested thoroughly in a $10m^2$ room decorated to resemble a store environment.

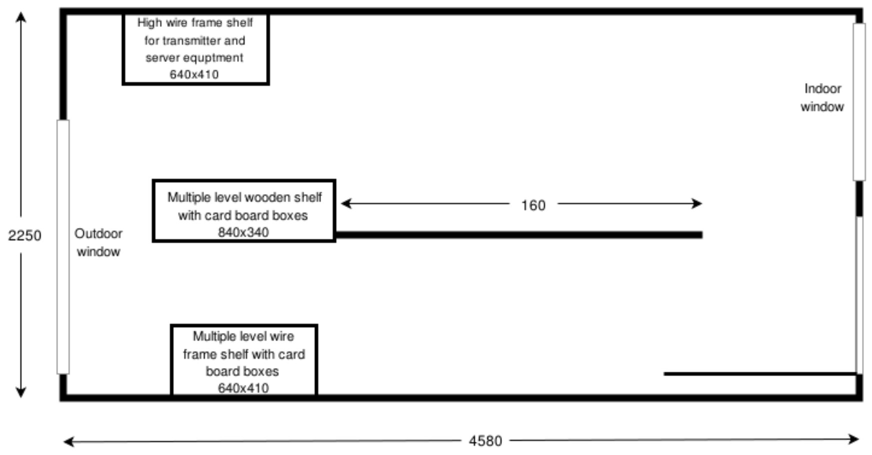


Figure 18: Blueprint of the experimental environment used in the developing and evaluation of the methods.

To match the end environment as close as possible a complete Pricer Electronic Shelf Label system [17] was installed with 29 shelf labels. The Pricer system consist of multiple digital price labels, a base station connected to the network with TCP/IP and a roof mounted transceiver-receiver module connected to the base station. The digital price labels have a screen capable of displaying graphics and a LED which can be set to flash in a factory predefined pattern. The labels have 2-way communication with the rest of the system using infrared light. The labels are spread out in various positions, some on the wall, some on wire frame shelves, some vertical and some horizontal. Some Labels are placed close together in groups to mimic possible placements in a real store.

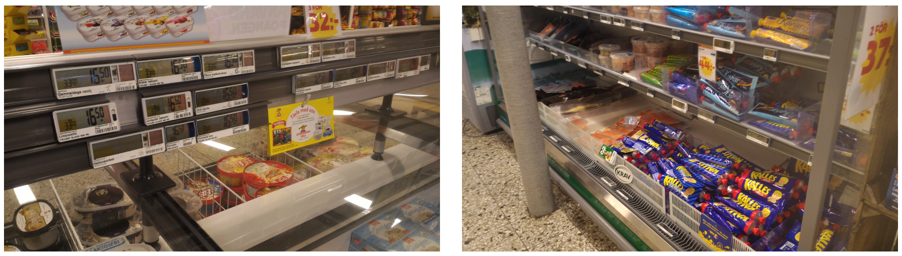


Figure 19: Example of placement of the labels in a real retail store. Some labels are close together while some are further apart. Note the horizontally placed labels in the bottom of the right picture.

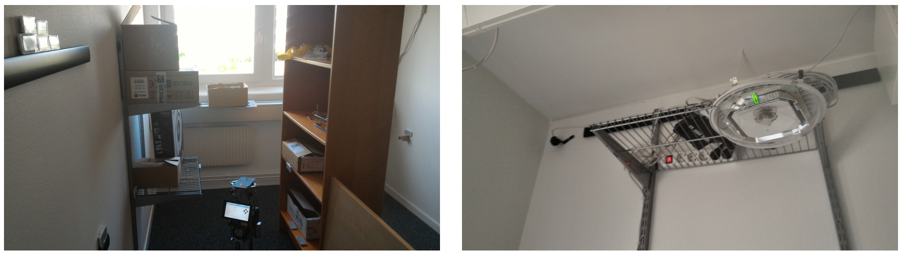


Figure 20: Left image shows example placements in the test room. In the left upper corner a group of five labels is seen and on the bottom shelf on the right some horizontal labels are located. Right image shows the mounting of the Pricer transmitter system. One of the Axis roof cameras can also be seen in the upper left corner.

6.2.2 Warehouse environment

The real store test has been performed in a 9.5 × 4.4 meter section of a store divided into five rooms whereof two rooms were used. Two of the rooms have been equipped with the same Pricer Electronic Shelf Label system as described earlier in the section. 44 price labels were installed on shelf fronts ranging from 20 to 120 centimeters height. The real store environment is a former village food store but is today a warehouse. The blueprint of the two rooms used for testing and the position of the labels are seen in Figure 21.

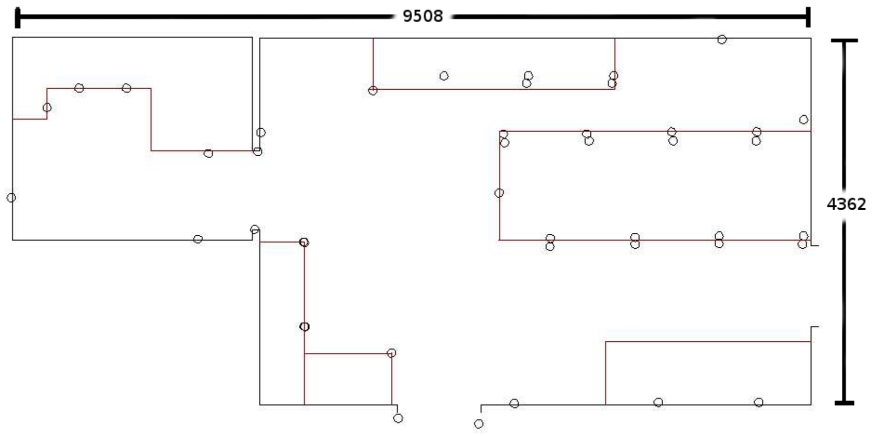


Figure 21: Blueprint of the part of the warehouse used for testing and evaluation of the methods. Pricer digital shelf labels are marked as circles.

6.3 Software

The software is network based and spread out over several nodes. The control center and graphical user interface are running on a stationary PC using Debian operating system [5] and communicates with all other nodes. The Raspberry Pi compact computer on the robotic platform is running Raspian operating system [20] and, inter alia, collects the sensor readings from the robotic vacuum cleaning platform to make them available over the network. The Pricer Electronic Shelf Label server is utilizing Windows operating system [12] and thus runs on a separate server PC. The Modcam cameras are running a distribution of the Android operating system [1] and the detection of flashing labels are done in each cameras software. Most of the software is written in Java [9] with the complement of native code using Java Native Interface.

6.3.1 Modcam camera unit

The software in the Modcam cameras is written in Java language for the Android operating system. At start-up the camera is automatically connecting to the control center server and goes into a passive state waiting for commands. The automatic start-up of the program is achieved by initializing a broadcast receiver listening to the boot completed action and thereafter start the app as a new intent. The program sets up a WiFi connection to connect to the local network and initializes a connection to the control center server through a socket client. It implements three main modes which can be started, paused and changed by commands sent from the control center: live image feed, record to memory card and LED detection with live network streaming. It also allows uploading of video saved on the memory card and upload of log files. Live telemetry of information such as battery level and charge current is available using Android Debug Bridge over WiFi or USB connection.

Images are fetched from native-code methods using Android NDK. Modcam SDK is implemented in C language and allows a feed of images through a frame client. The frame client is implemented to allow multiple apps to communicate with the camera module simultaneously without taking control over it. The images are fetched in 640x480 8-bit gray scale format.

In live image mode the frames are fetched and sent to the control center server without modification. Images are non compressed, neither intraframe nor interframe, resulting in low maximum frame rate. The frame rate is limited by the transfer rate in the network connection between camera and control center server.

In record mode the frames are written to the cameras memory card for later use. A frame is written as a raw data dump, without compression, to a binary file. Each frame is written to its own data file marked with the date and time stamp of the frame. When a image upload command is received all images within the folder are read and sent to the server in chronological order. This process is rather slow but allows non real time recordings to have significantly higher frame rate than live video.

The live LED detection mode finds flashing LEDs in the images and sends data live to the server. The stream can be set up to include both image data and detection data or only the latter. Because of the uncompressed images the frame rate is limited when image data is sent. The image analysis is however done at fixed frequency in the cameras and the current frame is only sent if the buffer is not full. During the mapping in this project the robot is not moving during the cycles. That is why the system can send only the first frame from each camera and then overlay with the detection data which is constantly live streamed, until the cycle for one location is finished. The method behind the detection algorithm is described in the *6.3.1 Detecting flashing LEDs* section. The implementation is based on the OpenCV open source computer vision library [15] using the OpenCV manager installation for Android operating system.

The detection of flashing LEDs starts with an image segmentation to find pixel groups with high derivatives in the time space. These areas are found by first using a Gaussian convolution to blur sharp edges. The kernel is 11x11 pixels with a 5 unit sigma. The current, blurred, frame is subtracted from the last, blurred, frame and the absolute difference is kept for each pixel. A binary matrix is created by thresholding the absolute change of the pixels. A pixels value at image location given by coordinates $(u, v)^T$ at the time i can thus be described as

$$p_i^{\text{segmentation}}(u, v) = \begin{cases} \mathrm{d}(u, v) < t : & 0, \\ \mathrm{d}(u, v) \geq t : & 1, \end{cases} \tag{66}$$

where

$$\mathrm{d}(u, v) = \mathrm{abs}\big(p_i^{\text{blurred}}(u, v) - p_{i-1}^{\text{blurred}}(u, v)\big), \tag{67}$$

and p is the pixel value and t the threshold. The position of the contours in the binary image are found using the method proposed by *Suzuki, Satoshi et. al.* [69]. The area of each detected pixel group is calculated using Green's theorem [48]. The groups are filtered with upper and lower limit on the area as well as the maximum length and maximum length to area ratio where the maximum length is the largest L_2 distance between two points on the boundaries. The centroid of the groups are found using Green's theorem and the frequency matching score is calculated using the cost functions presented in *6.3.1 Detecting flashing LEDs.*

Images are fetched at 10 frames per second during the analysis. With 10 Hz sampling the Nyquist frequency [49] becomes 5Hz which is the maximum frequency that the discrete system can sample without introduction of aliasing, also known as folding. The frequency spectrum of the Pricer digital labels LED flash consist of a 5Hz and a 1Hz peak. Thus the signal is right at the edge for what is possible to measure correctly. The reason for the low frame rate was initially lack of processing power which prevented real time analysis at higher frequency. Over time the implementation was optimized and the time consumption running in the cameras CPU is low enough to allow the full 30 fps frame rate. The frequency has however not been increased due to already good results in the detection of labels. The main reason for false positives have been seen to be reflections in walls and windows. A LED reflected in a surface has the same flash characteristics and thus passes through the detector. A method to prevent this would be to detect simultaneous flashing and only keep the brightest source. This would however need to be distributed between the software running at each camera and would require synchronization of time between the cameras with considerable precision. Another method would be to use robust techniques in the the bundle adjustment to find outliers by their high reprojection error.

Calibration of the cameras was done using 3 radial coefficients and two tangential. It was done using 20 images of a chessboard pattern using both Matlab camera calibrator app and an algorithm manually implemented in Java using OpenCV library. The calibration with the lowest reprojection error for each camera was used. When mixing calibrations using Matlab and OpenCV it is worth noting that

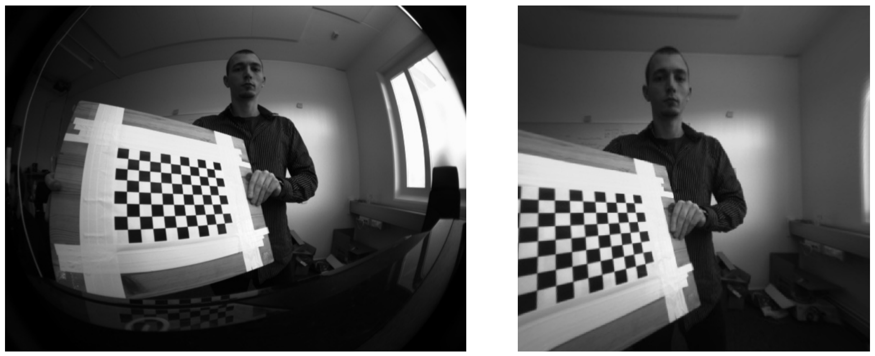
$$\boldsymbol{K}_{\text{Matlab}} = \boldsymbol{K}_{\text{OpenCV}}^{T}, \tag{68}$$

where $\boldsymbol{K}$ is the camera matrix and distortion parameters, $\boldsymbol{D}$, in Matlab is returned as

$$\begin{aligned} \boldsymbol{D}_{\text{Matlab, RadialDistortion}} &= (k_1, k_2, k_3), \\ \boldsymbol{D}_{\text{Matlab, TangentialDistortion}} &= (\tau_1, \tau_2), \end{aligned} \tag{69}$$

while OpenCV methods expect them as

$$\boldsymbol{D}_{\text{OpenCV}} = (k_1, k_2, \tau_1, \tau_2, k_3). \tag{70}$$

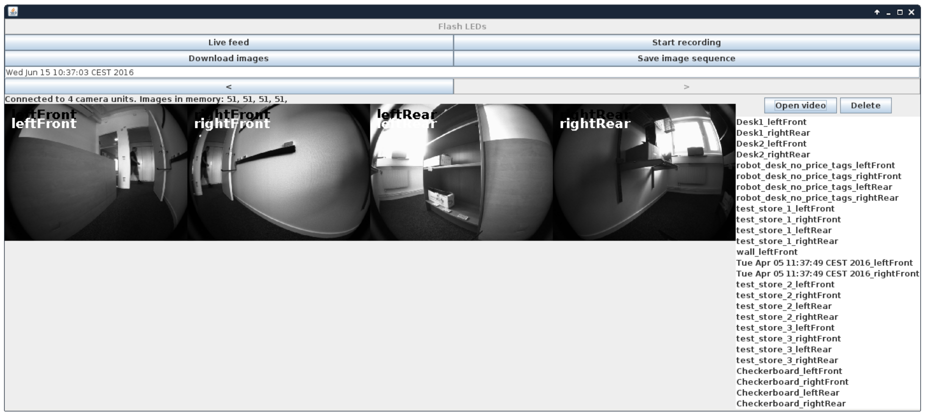


(a) Uncalibrated image (b) Part of calibrated image

Figure 22: Calibration of wide angle Modcam cameras was done using images of chessboard patterns.

6.3.2 Data analytics and detector evaluation

In the design process of the detector used to localize the flashing LEDs it is of high importance that different detectors can be tested and evaluated on the same data set. To allow this a software used to record video sequences from the robot and store them on the local computer was designed. The program allows record-and-download or live image feed, the latter one at more limited frame rate. The frames in the saved videos can be played in real time or stepped through frame-by-frame. Each video stream can be opened in a detector testing window where the detector, also implemented in each camera's software, can be tested locally on the PC. By manually tagging the true locations of digital price labels the software can automatically evaluate the precision and fault rate of the detector. Characteristics about the true locations' flashes can also be extracted and used to tune the parameters of the detector. The detector can be run in real time or in a frame-by-frame mode to evaluate every reaction to the changes between two frames. Detections are presented together with information about the cost function output used to analyze frequency content. Images can also be exported as jpeg used among others to calibrate the cameras.



Wed Jun 15 10:37:03 CEST 2016

Connected to 4 camera units. Images in memory: 51, 51, 51, 51,

Desk1_leftFront
Desk1_rightRear
Desk2_leftFront
Desk2_rightRear
robot_desk_no_price_tags_leftFront
robot_desk_no_price_tags_rightFront
robot_desk_no_price_tags_leftRear
robot_desk_no_price_tags_rightRear
test_store_1_leftFront
test_store_1_rightFront
test_store_1_leftRear
test_store_1_rightRear
wall_leftFront
Tue Apr 05 11:37:49 CEST 2016_leftFront
Tue Apr 05 11:37:49 CEST 2016_rightFront
test_store_2_leftFront
test_store_2_rightFront
test_store_2_leftRear
test_store_2_rightRear
test_store_3_leftFront
test_store_3_rightFront
test_store_3_leftRear
test_store_3_rightRear
Checkerboard_leftFront
Checkerboard_rightFront
Checkerboard_leftRear
Checkerboard_rightRear

Figure 23: Graphical user interface designed to record and download videos from cameras and manage locally saved video sequences.

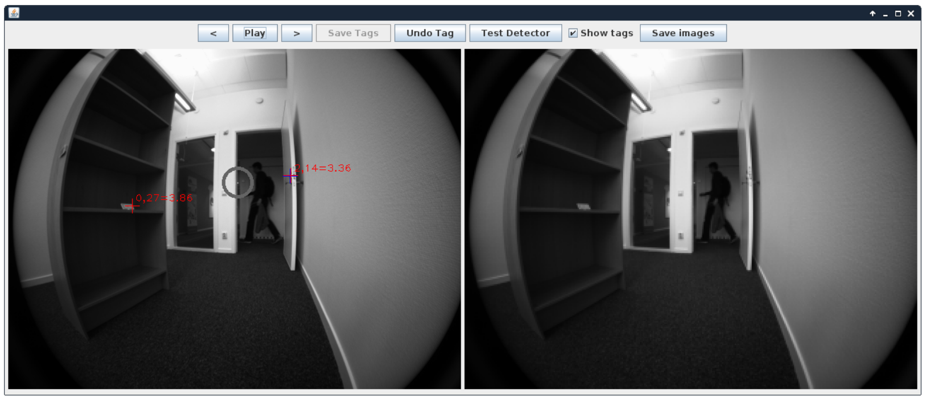
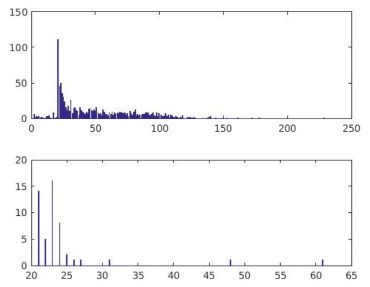
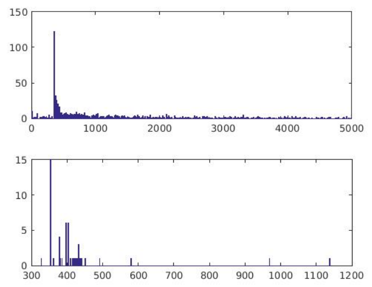


Figure 24: Graphical interface used to test detection algorithms on repeatable data and extract information about flash characteristics. Blue markers are manually placed true locations of digital labels, red markers are automatically detected displayed together with cost function value. Gray circle marker indicates typically hard to classify areas, in this case a ceiling light partly hidden behind the door post. Right image is unmodified frame. Manual stepping frame-by-frame is possible to evaluate the detectors reaction to a single frame change.



(a) Length of the longest side of the pixel region. Upper is all detections, lower is detections at true locations.



(b) Area of the pixel regions. Upper is all detections, lower is detections at true locations.

Figure 25: The tagging of true positions in the GUI allows extraction of data comparison between all pixel regions passing through the time derivative threshold and the regions at true label locations only.

6.3.3 Robotic platform server

The Raspberry Pi 3 compact computer has been configured to start a Java application running the network server. The application communicates with the robot through IRobots Open Interface [7] implementing the RS232 protocol

[67]. Communication over the UART was done using the RXTX library [21]. According to the specifications in the open interface a sensor data stream was set up. The stream can be specified to include just the sensors of interest and contains a checksum, on top of the checksum specified in RS232 protocol, to assure no corrupt data. The stream is sent at 66Hz and thus allows fast update of sensor information. Requesting sensor values individually is slow compared to the stream and can in worst case give a one second delay before the response is sent. Because of this an Irobot Roomba model which support streams is highly recommended for similar projects. The manual for the Open Interface implemented by the robot is hard to find and only a manual for the 500-series was found at a third party web page [7]. The robot used in the project is from the 600 series and this might explain some of the contradictions to the protocol explained in the following text. Due to what seems like a bug in the protocol the distance and angle from the deduced reckoning are always zeros when sent in the stream. This was solved by complementing the stream with manual requests for those two variables. As soon as an response was received a new request was sent. This resulted in a bit complex code when the response and stream might be received in a mixed order. This was solved by comparing the answer to expected models and thereby being able to classify it as answer on request or data stream. According to the manual the distance should be in millimeters and the angle in degrees which does not seem to be the case on the robot used in the project. The calibrations of rotation and distance resulted in 7.0 units per millimeter traveled and 0.58 units per degree turned. Furthermore the manual states that the checksum of the stream is calculated as the 8-bit complement of all bytes except the header. The 600-series, or at least the robot used in the project, is including the header in the checksum and thus

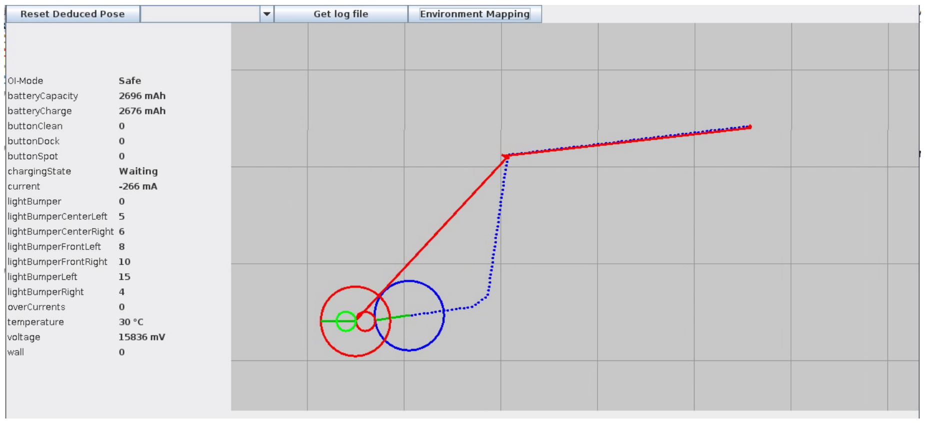
$$D_{\text{header}} + N_{\text{bytes}} + \boldsymbol{D}_{\text{id,data,i}} + D_{\text{checksum}} \mod 255 = 0, \tag{71}$$

where N_{bytes} is number of bytes and $\boldsymbol{D}_{\text{id,data,i}}$ is the id and data bytes for all i. The inclusion of the header in the calculation of the checksum seems like an important feature because of the fact that a broken data line resulting in all zeros otherwise would yield a correct checksum ($0+0+...+0 \mod 255 = 0$). This can be avoided by using the expected value of the header instead of the received.

The I/O-pins on the Raspberry Pi are used to interface with the device-detect line to the Irobot Roomba and to control the flashing of the LED beacons used to find the robots position in the roof camera images. The I/O pins are controlled with the use of Pi4J library [16].

6.3.4 Control center and planning

The control of the mapping process is done in the PC running Debian operating system. A program written in Java combines a robot controller and telemetry system with a label detecting and positioning system. The robot controller implements a graphical user interface displaying live sensor information from the robot such as battery voltage, charge level and current, obstacle sensor data and mode states. A dynamic map is also implemented showing the real time pose of the robot and its move trace based on the deduced reckoning and the roof camera detections. The map is visualized with a 500mm grid seen as a top-down view of the room with the origin marked as a corner in the room. The control center GUI also allows manual control of the robot which makes it possible to manually drive the robot using a WLAN connection.

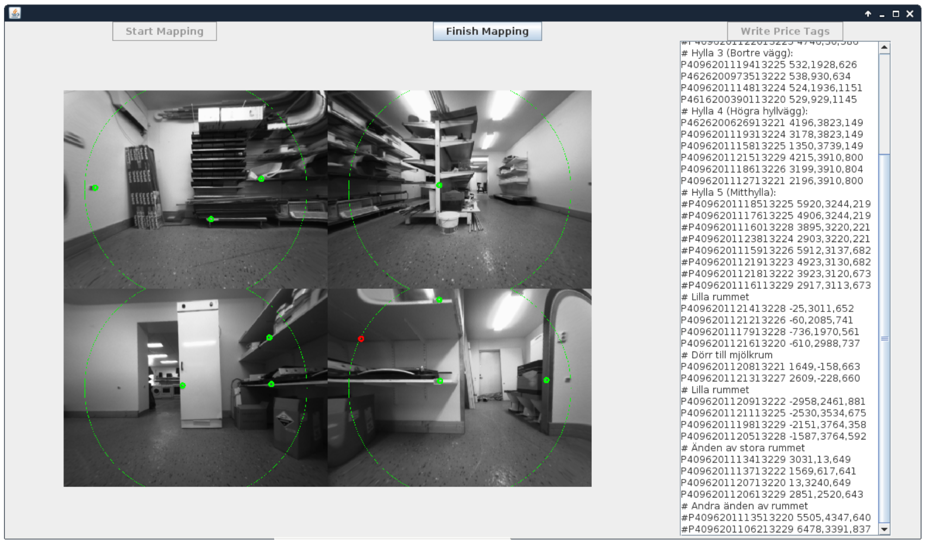


OI-Mode	Safe
batteryCapacity	2696 mAh
batteryCharge	2676 mAh
buttonClean	0
buttonDock	0
buttonSpot	0
chargingState	Waiting
current	-266 mA
lightBumper	0
lightBumperCenterLeft	5
lightBumperCenterRight	6
lightBumperFrontLeft	8
lightBumperFrontRight	10
lightBumperLeft	15
lightBumperRight	4
overCurrents	0
temperature	30 °C
voltage	15836 mV
wall	0

Figure 26: The graphical user interface created to allow 2-way live interfacing with the robot. Robots position is displayed for each roof mounted camera and deduced reckoning. Sensor data from the robot is displayed to the left.

The deduced robot pose estimation is calculated in the software, designed for the robots computer, based on a model of the robot's mechanics. The pose estimation is sent to the control center at 50Hz where a trail of the positions is created. The trail is stored in non volatile memory which allows the mapping to continue even after an all system shutdown. This makes it possible to map the premises split into several days or to complement an earlier mapping tour with more data. The telemetry data and the map can be seen in Figure 26 where the deduced position and its trail are shown together with trail and pose estimated from one roof mounted camera.

The label detecting and positioning system implements a GUI allowing control over the label mapping sequence. A text field allows the user to specify the product ID of all labels and optionally the real position of it to use as a reference. Images taken from the four cameras mounted on the robot are displayed with an overlay of colored circles for all detected labels' LEDs. The detections can be marked false by clicking on them with the mouse and thus allowing manual filtering of outliers. The GUI can be seen during a mapping operation in Figure 27. Green thin bordered circles indicates the area of the images where the lens distortion calibration error is assumed to be less than the acceptable threshold.



```
# Hylla 3 (Bortre vägg):
P409620111941 3225 532,1928,626
P462620097351 3222 538,930,634
P409620111481 3224 524,1936,1151
P461620039011 3220 529,929,1145
# Hylla 4 (Högra hyllvägg):
P462620062691 3221 4196,3823,149
P409620111931 3224 3178,3823,149
P409620111581 3225 1350,3739,149
P409620112151 3229 4215,3910,800
P409620111861 3226 3199,3910,804
P409620111271 3221 2196,3910,800
# Hylla 5 (Mitthylla):
#P409620111851 3225 5920,3244,219
#P409620111761 3225 4906,3244,219
#P409620111601 3228 3895,3220,221
#P409620112381 3224 2903,3220,221
#P409620111591 3226 5912,3137,682
#P409620112191 3223 4923,3130,682
#P409620112181 3222 3923,3120,673
#P409620111611 3229 2917,3113,673
# Lilla rummet
P409620112141 3228 -25,3011,652
P409620112121 3226 -60,2085,741
P409620111791 3228 -736,1970,561
P409620112161 3220 -610,2988,737
# Dörr till mjölkrum
P409620112081 3221 1649,-158,663
P409620112131 3227 2609,-228,660
# Lilla rummet
P409620112091 3222 -2958,2461,881
P409620112111 3225 -2530,3534,675
P409620111981 3229 -2151,3764,358
P409620112051 3228 -1587,3764,592
# Änden av stora rummet
P409620111341 3229 3031,13,649
P409620111371 3222 1569,617,641
P409620112071 3220 13,3240,649
P409620112061 3229 2851,2520,643
# Andra änden av rummet
#P409620111351 3220 5505,4347,640
#P409620110621 3229 6478,3391,837
```

Figure 27: A graphical user interface displays the detected labels and allows manual filtering by marking detections true or false with a mouse click. Product ID and optional 3D location of the labels are displayed and modified in the right side text field.

Detection of the beacons on top of the robot is done in the control center program. Two roof mounted cameras are used to cover the full room from two directions. The cameras are supplied by Axis Communications [2] and consist of a F34 main unit with a F1004 and a F1005 lens. The detection of the beacons are done mostly similar to the detectors described in the *6.3.1 Modcam camera unit* section with difference that it is only carried out in the red channel and with different expected target frequency for the front and rear beacon. Images are fetched with an average 12.9 fps over TCP/IP from the roof mounted cameras base unit as intraframe compressed M-JPEG imaged sent as a stream of HTTP replies. Maximum time to process one frame was during 109 seconds measured to 180 milliseconds while the minimum was 63 milliseconds. The front and rear beacons are flashing respectively at 6.25Hz and 3.125Hz which is mostly less than the Nyquist frequency, i.e. half of the sampling frequency. While the frame rate in the Modcam camera units is stable the frame rate of images acquired in the control center from the roof cameras varies much. The variance in frame rate is introduced by the non real time operating system running several threads on each core. To solve this problem the time between two rising edges in Equation 53 is not calculated with the expected value of the frame rate but instead the actual measured time of arrival. Using axis F-series cameras it is possible to request a time stamp indicating the exact time of capture together with the stream. While the capture time stamp requires model specific design to create and decode the HTTP response the time of arrival was instead used. e_a and e_b are in this case stored as time since boot up instead of frame number which yields the delta time as

$$t_{b,a} = e_b - e_a. \tag{72}$$

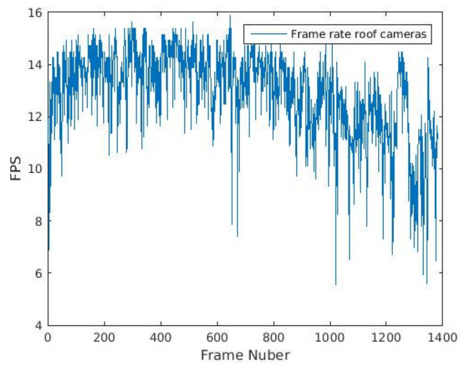


Figure 28: The frame rate of the roof mounted cameras during 109 seconds. The frame rate is unstable due to the PC's scheduling of threads.

The detected beacons are displayed as small circles in red for rear beacon and green for front beacon in the GUI top view map by applying the homographic transform to the image coordinates. The position and heading of the robot is then calculated as the normalized vector from rear beacon to front beacon respectively the centroid of the two coordinates in the world coordinate frame. In Figure 29 the detection of front and rear beacon in two roof mounted cameras can be seen. The green circles indicates that three out of four views of the beacon LEDs were detected in this particular frame despite a bypassing person. By filtering the detections over a couple of frames accurate and stable readings can be achieved.

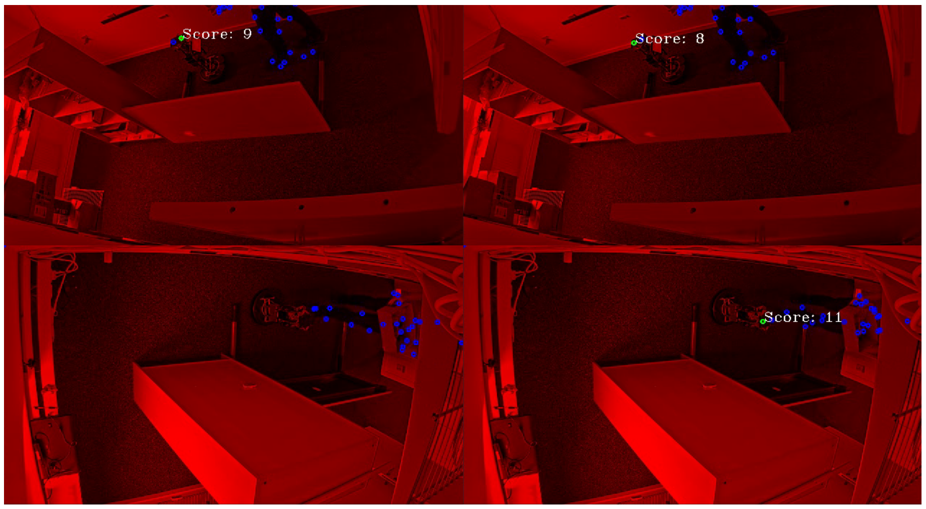


Figure 29: The LED beacon detection program using two roof mounted cameras. Left images shows detection of rear beacon, right images shows detection of front beacon. At those particular frames the front beacon was located in both cameras while the rear beacon only was located in the upper camera. Blue circles mark areas of high time derivative, here mostly consisting of the movement of a bypassing person, and green circles together with a confidence score indicate areas with expected frequency content.

Communication with the robot's computer is done over WLAN and is based on a 2-way object stream using a socket server in the PC and a socket client in the robot's computer. Communication is done to update all information at 50Hz.

Communication with the Pricer Server is done through a Java socket client implemented locally on the Pricer Server system. A request to flash the LED of a label is sent to the client together with the labels product ID, flash strength and requested duration, for further explanation see *6.3.6 Pricer Electronic Shelf Label system integration.*

6.3.5 SLAM/SfM implementation

Once all information about detection of labels, beacons in roof mounted cameras and deduced tracking is acquired it is automatically exported to Matlab [10]. The communication between Java and Matlab is done with the Matlabcontrol API [11] which is a wrapper for the undocumented Java Matlab Interface used internally in Matlab. Matlabcontrol allows adding, modifying and removing environment variables in Matlab as well as evaluating instruction or calling functions. All code used to model cameras, estimating positions, using perspective N-point algorithms and bundle adjustments has been implemented in Matlab during the project due to the easy use of linear algebra. One Matlab function can be called after collecting information to calibrate the pose of all cameras on the robot and the location of the beacons. Another function can be called to estimate the world, i.e estimate position of all unknown labels and the location and heading at all of the robots positions. All iterative minimization has been solved with the *lsqnonlin* method using a subspace trust-region method based on interior-reflective Newton method [38]. This method allows minimization of bounded or non bounded multiple variable, multiple cost function problems. To solve the minimization problems faster and more accurately all cost functions have been split up into vectors of as many cost functions as possible. In the case of a reprojection of one point in one camera this would mean one cost function for error in y-direction and one for error in x-direction. On a problem with F different robot locations, C cameras on the robot, P known or unknown points and R roof mounted cameras looking for B beacons on the robot this would result in $2PCF + 2RB$ cost functions and the minimization algorithm searching for

$$\underset{\boldsymbol{O}}{\operatorname{argmin}} \|\boldsymbol{F}\|_2 , \quad \boldsymbol{F} = \begin{bmatrix} F_1(\boldsymbol{O}) \\ F_2(\boldsymbol{O}) \\ \vdots \\ F_{2PCF+2RB}(\boldsymbol{O}) \end{bmatrix} . \tag{73}$$

6.3.6 Pricer Electronic Shelf Label system integration

The Pricer server software has a text file based API. A Java program has been developed to run in the server system and communicate with the other nodes through a TCP/IP port. Once a request is received from the robot control center the instruction is translated into the Pricer system script language and written to a directory which has been set up for scanning by the server software. The Pricer text based API is non real time and is scanned approximately every third second. This combined with the non real time transmission between the Pricer base station and the digital price labels introduces a considerable delay of unknown magnitude. No feedback is available in the API to know when the labels actually receives the flash LED command. To be sure that two digital price labels are not flashing at the same time, resulting in a probability of mixed up positions, a delay of many seconds between a flash end and a new flash start is necessary. The server system also implements other APIs but the problem is consistent through all of them.

6.3.7 Line detection

The line detection algorithm was first implemented using Matlab to evaluate and extract images of the different steps. The Matlab implementation was very slow and far from applicable in real time line following. A real time implementation was made using the interpreted Python programming language [18]. Much of the calculations are done using the efficient algorithms already implemented in OpenCV and the Numpy [14] package for scientific computing. The iteration over the pixels on each stroke and the counting of neighbors with same stroke width requires heavy looping over a total of millions of pixels in every frame. To speed up this looping those parts of the code were implemented using Cython [4] which is an optimizing static compiler for the Python and extended Cython programming language. By using partly statically typed Python code built to c-code and partly pure c-code called from the Cython methods to integrate them with the rest of the Python program real time performance was achieved.

The gradients are calculated on the gray scale image. The RGB images from the camera are recalculated to gray scale using

$$Y = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B. \tag{74}$$

The Sobel operator is used to approximate the derivative of the gray scale images. The kernels used for the convolutions to get the row- and column-wise derivative are given by

$$\boldsymbol{C}_{row} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad \boldsymbol{C}_{col} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{75}$$

Only the relations between the magnitudes are of interest, not the absolute values. The magnitude of the gradient $\boldsymbol{G} = (G_{row}, G_{col})^T$ is simplified to speed up calculations using the approximation

$$G_{\text{mag}} \approx |G_{\text{row}}| + |G_{\text{col}}|. \tag{76}$$

Start and stop points are defined as the pixels where the gradients magnitude is above a threshold t. Start pixels are defined as the ones where the gradient along the column axis is positive, the rest are defined as end pixels. If $\boldsymbol{p}_{\text{start}}$ is the set of all pixels in a start-map, $\boldsymbol{p}_{\text{end}}$ is the set of all pixels in an end-map and $i \in \boldsymbol{I}$ where $\boldsymbol{I}$ is all indexes in the maps

$$p_{\text{start},i} = \begin{cases} G_{\text{mag},i} > t \quad \text{and} \quad G_{\text{col},i} > 0 : \quad G_{\text{mag},i}, \\ \text{else:} \quad 0, \end{cases} \tag{77}$$

and

$$p_{\text{stop},i} = \begin{cases} G_{\text{mag},i} > t \quad \text{and} \quad G_{\text{col},i} \leq 0 : \quad G_{\text{mag},i}, \\ \text{else:} \quad 0. \end{cases} \tag{78}$$

The stroke length is found by starting on each nonzero pixel in the start-map and stepping pixel by pixel in the negative direction of the gradient until the pixels value in the end-map is nonzero. The method's order of complexity is linear with the number of nonzero pixels in the start-map and also linear with the stroke width. Looping in Python is very slow mainly because of the large overhead for both functions and variables. By using Cython compiler and extended language the type of python variables can be statically typed and the access to multi-dimensional arrays can be done using typed memoryviews. Instead of using math and standard functions as defined in Python they are called from native c libraries. Using this approach all loops can be compiled to pure c-code resulting in several magnitudes of speed up.

7 Experimental results

The experimental results are presented below and are the foundation of the propositions about what methods and sensors to use. The result is split up into four main parts, detection of digital price labels using vision systems, localization of robot based on different sensor data, including vision, as well as the accuracy of mapping the 3D position of all price labels in the room and the path planing using the line detection approach.

7.1 Detection of labels

The detection of the LEDs on the price labels show good detection rate in the performance test. An evaluation was performed in 15 recorded video sequences with a total of 31 flashing LEDs. The LEDs were sometimes flashing alone and sometimes several together. The videos were recorded in different angles, some with the sunlight from behind the cameras and some against the sun, some with persons walking by in the background and at different distances from the labels. The detector found 30 out of the total 31 labels with a total false positive count of 1 resulting in both a recall and precision of 96.8 percent. The statistics for each video sequence recorded can be seen in Table 1. During the final test in the store-like environment built in the office, as can be seen in Figure 33, the total number of true positives was 78 with 6 false positives. This yields a slightly lower precision of 92.9 percent. The recall was not measured in this case but is believed to be slightly lower. The largest problem encountered is LEDs reflected in the walls or windows. The reflections shows the same frequency content and thus can't be filtered away by the bandpass filter. The intensity of a reflection can be higher than the intensity of the direct view of a LED because of different exposure times between cameras. These false detections will have to be sorted away by outlier filters or robust projection methods.

Table 2: Max view angle for label detection

Light condition	Light direction	Distance	Max view angle	
Very bright	Behind camera	525	86.2	
Very bright	Behind camera	2520	-	[1]
Very bright	Opposite camera	1300	76.3	
Very bright	Sideways of camera	1570	80.3	
Fair	Passive	3180	18.2	[2]

Distance in millimeters, angles in degrees

[1] Over exposed in area surrounding label (Figure 30)

[2] Close to maximum detection distance

Table 1: Precision and recall of LED label detector

# Labels	# Detected	# False detections	Precision [%]	Recall [%]	
0	0	1	0	-	
1	1	0	100	100	
0	0	0	-	-	
0	0	0	-	-	
5	4	0	100	80	
2	2	0	100	100	
1	1	0	100	100	[1]
0	0	0	100	100	
2	2	0	100	100	[1]
3	3	0	100	100	
6	6	0	100	100	
4	4	0	100	100	
2	2	0	100	100	
2	2	0	100	100	
3	3	0	100	100	
31	30	1	96.8	96.8	

[1] One or more bystanders walking by in camera view

LEDs in some positions of the image can be detected from large distances and wide view angles while some other parts might be saturated such that the light of the LED can't be detected, see Figure 30. The maximum view angles of some labels were measured and are presented in Table 2. The viewing distance can easily be increased by lowering the threshold of the time derivative of the pixels light level. Doing this however will increase the number of false detections.

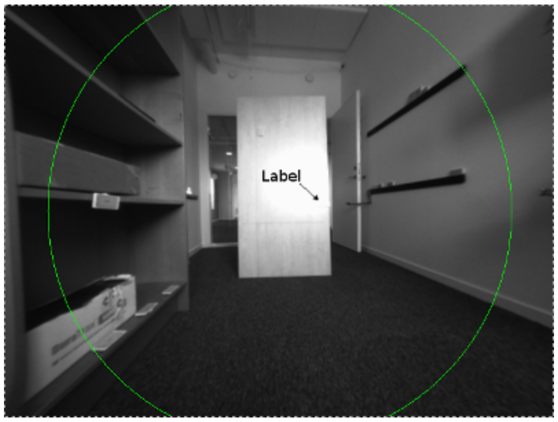


Figure 30: Detection of label is not possible due to the over exposed and saturated area covering the label.

7.2 Experimental environment results

Below follows the presentation of results achieved when mapping the in-office $10m^2$ room decorated to resemble a real store.

7.2.1 Manual measurement of true label position

The true positions of the digital price labels, used as a reference in the *7.2.3 Mapping accuracy* section, were measured manually using a laser range finder. The manual measurements are prone to error. To minimize the error every distance between each of the points was measured and multidimensional scaling was used to find the most probable location of each label. The mean error was calculated to 13.3 millimeters and the maximum to 26.5 millimeters.

7.2.2 Robot positioning

The positioning of the robot has been compared using three different methods, deduced reckoning using the odometers on the robots wheels, Perspective N-Points (PnP) using a single camera image with a number of known points to position the robot and the roof mounted cameras' pose estimation of the robot. All these results have been combined and used in the complete SLAM problem.

7.2.2.1 Perspective N-Points

A camera with a view of five digital price labels with known position on a coplanar wall was used to find the cameras position using Perspective N-Points algorithm. The error $\boldsymbol{d}_e = (X, Y, Z)^T$ in world coordinates of a single test was manually measured to

$$\boldsymbol{d}_e = (19.4, 29.0, 70.1)^T \text{millimeters}.$$

This resulted in a reprojection error $\boldsymbol{R}_e$ in the image specified for each labels L_2 norm distance error measured in the image to

$$\boldsymbol{R}_e = (1.35, 0.78, 0.50, 0.94, 0.25)^T \text{pixels}.$$

The distances from the camera to each of the labels are given by $\boldsymbol{d}_{\text{label}}$ as

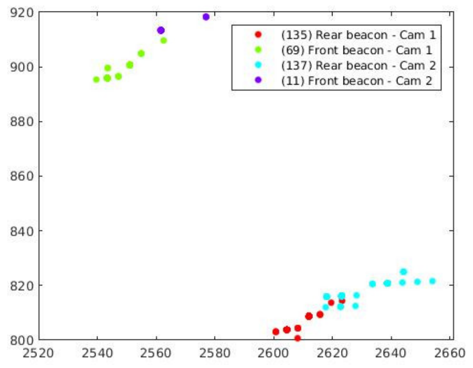
$$\boldsymbol{d}_{\text{label}} = (1138, 1064, 1191, 1137, 1232)^T \text{millimeters}.$$

7.2.2.2 Roof camera pose estimation

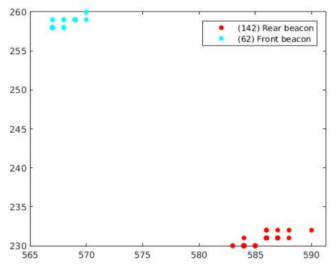
The positioning of the robot using the roof cameras was achieved with reasonable precision. The standard deviation of detections projected to world coordinates and detections direct from the camera sensor can be seen in Table 3. The deviation was measured by having the robot statically placed in the center of the room and recording a total of 352 detections during 20 seconds. The precision in the measurements is dependent on the robots position in relation to the roof camera. The perspective projection has a big influence on the errors when the robot is far away from the camera. The standard deviation is in most cases approximately above 1 pixel and it would probably be possible to get better estimations by improving the LED flash finding detector. In Figure 31 the detections of the beacons can be seen. Note how the perspective warp introduces a correlation in the error of the world coordinate detections. This is however solved in the optimization by minimizing the reprojection error in the cameras image planes instead of the error in world coordinates. Using this techniques the solver will automatically trust a camera less if the robot is far away than if it is just under the camera.

Table 3: Standard deviation in roof cameras' beacon detection algorithm

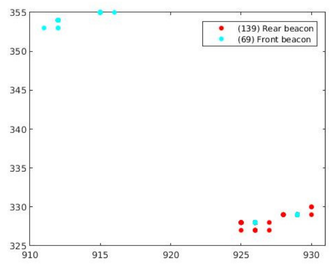
Camera	Beacon	STD	
		Pixels	Millimeter
1	Rear	1.26	4.90
1	Front	1.15	5.39
2	Rear	1.72	9.08
2	Front	12.95	7.52



(a) The estimated position of the robots front and rear beacon (green, purple and red, cyan) using their homography transform to real world coordinates (mm). The robot was placed stationary in the center of the room.



(b) Position in image coordinates of front and rear beacon detected in camera 1. A total of 142 detections of the rear beacon have been mapped to only 11 pixels in the camera image.



(c) Position in image coordinates of front and rear beacon detected in camera 2. A total of 139 detections of the rear beacon have been mapped to only 8 pixels. Two miss-detections can be seen where the front beacon has been mixed up with the rear.

Figure 31: Detections of beacons during 20 seconds from two roof cameras. Two things can be noted, the spread is mostly along the same axis. This correlation exists because of the projective transform. The numbers of detections of front respective rear beacon for camera 1 are 135 and 39 and for camera 2 137 and 11. Most of the points are placed exactly on top of each other, this is because of the limited resolution (1280 × 720) in the cameras.

7.2.2.3 Deduced reckoning

Deduced reckoning is always subject to drift due to the integration of the errors. Using differential drive the robot is especially receivable to a rotational wrap created by small angular differences resulting in large distance errors. The robot has been tested on a textile surface using the roof mounted cameras as position reference. The result can be seen in Figure 32. Although the results are bad on their own they give enough extra information to the SLAM algorithms to induce good final results.

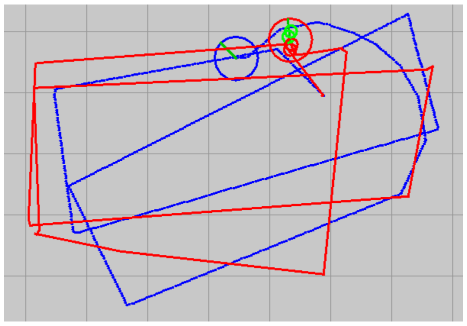


Figure 32: Comparison of the robots coordinates using the roof cameras (red trail) and the deduced reckoning (blue trail). It can be seen that the deduced reckoning suffer from a large drift creating the typical spiral pattern. Grid size is 500mm.

7.2.3 Mapping accuracy

The result visualized in this chapter is based on a loop around the test environment. The images taken during the mapping and all detected digital labels are seen in Figure 33. A top view schematic map with the obstacles can be seen in Figure 18.

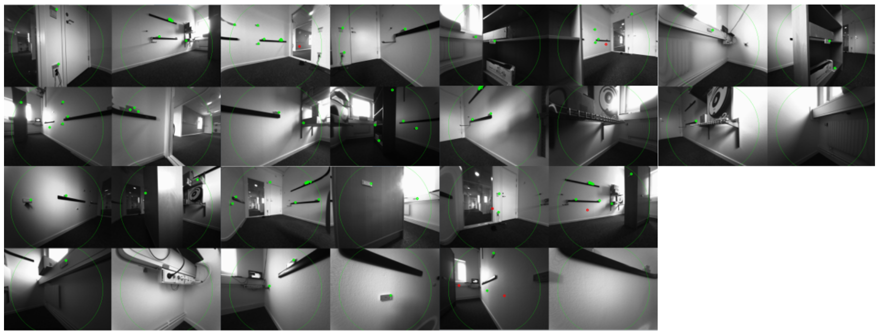


Figure 33: The images extracted by the four cameras during one loop around the $10m^2$ room. Green circles are confirmed detections of price labels while red are filtered outliers

Once the SLAM algorithm is completed the map of the room is presented as a 3D-graph. Orange points represent the estimated location of all digital price labels. If the true locations were submitted they are shown as red or black dots. The red dots are connected to their corresponding estimation by a black line. If there are too few views of a price label to position it the true location is marked black and no estimation is done. Each position of the robot during the frames taken are visualized by a set of three arrows. These locations are connected by dotted paths. One path represents the initial guess, taken from deduced reckoning or roof mounted cameras tracking capabilities, and the other track represent the estimated movement given by the SLAM algorithm. In Figure 34 a mapping of the test room and 29 digital price labels can be seen. In Table 4 the mean and median distance error of the estimated positions for the 29 labels can be seen. It's worth noting that most methods show similar results but is about ten times more accurate than the linear method that is commonly used. The estimation errors are calculated as the Euclidean distance from the manually measured point locations. The measurement of the points was explained in section *7.2.1 Manual measurement of true label position.* The results presented in Table 4 are from a single mapping route and it is worth noting that many of the errors are within the measurement error of the true locations and the estimations might thus be better than what is shown in the results.

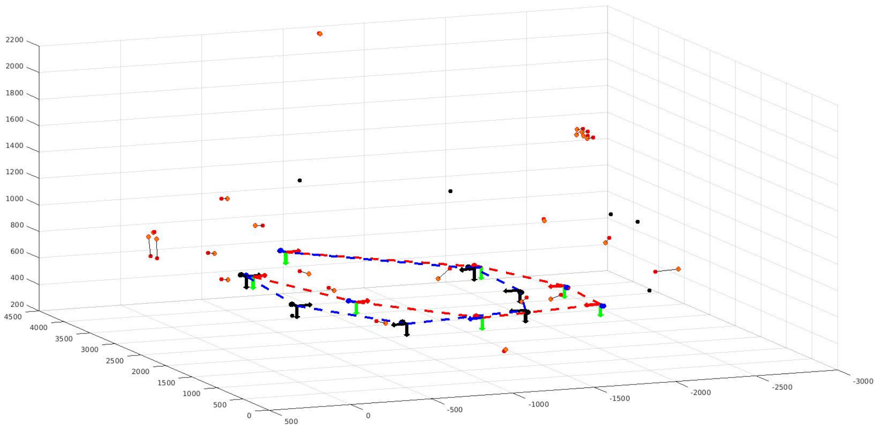


Figure 34: Mapping of a room with 29 digital price labels. Each labels estimated position is shown as an orange dot while the true, manually measured, position is shown with a red dot. A black line connects each true position with its estimation. Black dots are non positionable due to too few views of it. Mapping was done from seven locations and the pose estimated from deduced reckoning is shown as a blue line while the estimated path using bundle adjustment is shown in red. Axis units are in millimeters.

Table 4: Estimation error of digital price labels position

Known points	Localization method	Estimation method	Mean error	Median error
0	Roof mounted camera	Bundle adjustment	66.4	33.7
0	Roof mounted camera	Linear (SVD)	939	548
0	Deduced reckoning	Bundle adjustment	70.4	46.5
0	Deduced reckoning	Linear (SVD)	755.5	462.5
4	Roof mounted camera	Bundle adjustment	55.0	34.0
4	Deduced reckoning	Bundle adjustment	70.2	43.1

Estimation error given in millimeters

7.2.3.1 Known point references vs. all unknown

If some of the digital price labels in the room are placed at known locations these can be used to find the pose of the robot. This can either complement or fully replace the localization done by deduced reckoning or roof mounted camera tracking. If only deduced reckoning is used no absolute world coordinates is available and the resulting map will be in a local coordinate system with the same scale. By complementing with two known points the absolute position can be calculated for all other price labels. Figure 35 shows a comparison between all unknown or four known points using the roof mounted cameras as position feed back and Figure 36 shows the same comparison using only deduced reckoning. No big difference in the results can be seen. From Table 4 the mean error using four known points can be found to be 17 percent lower when using roof mounted camera feedback and 0.3 percent using deduced reckoning.

7.2.3.2 Roof camera positioning vs. deduced reckoning

The position feedback from the roof mounted cameras resulted in a better result albeit not significantly. The result using bundle adjustment with and without the roof mounted cameras can be seen in Figure 35 vs. Figure 36. The estimation achieved using the roof mounted cameras showed a 5.7 percent lower mean error. When using the linear method for estimation of points, not using the feed back from roof mounted cameras resulted in a 20 percent better estimation. The result of using the linear method is very unstable and it would require more tests to evaluate if the contradiction means something or is a coincident.

7.2.3.3 Linear vs. bundle adjustment

The bundle adjustment shows an extreme improvement in accuracy compared to the linear method. One of the explanations for this has been investigated earlier in the report covering the geometric importance of the minimized functions. The estimation using only deduced reckoning was improved by 93.0 percent when changing from linear triangulation using Singular Value Decomposition to bundle adjustment. The corresponding improvement when using feed back from

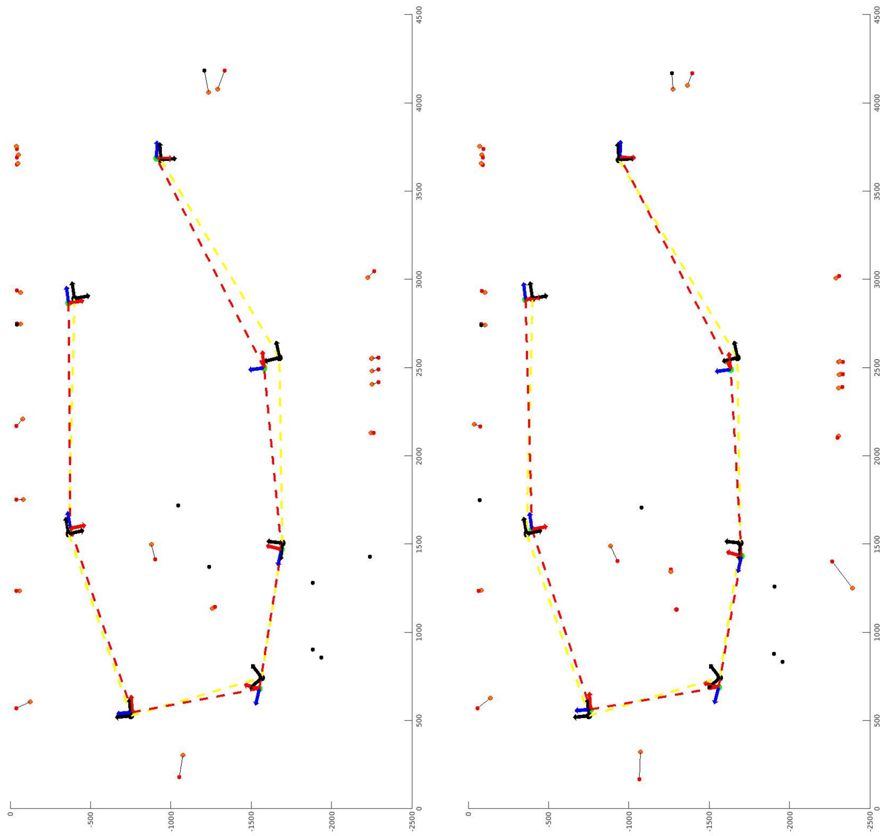


Figure 35: Images show mapping of price labels using position feed back from roof mounted cameras. Left image use 4 pre-known labels to improve positioning while right image has no view of known points. Red points are true locations, orange points are estimated locations. The yellow stroke is the estimated path of the robot using the roof mounted cameras and the red stroke is the estimation after bundle adjustment.

roof mounted cameras is 90.7 percent. A weak side of the bundle adjustment is its iterative approach. A linear triangulation demands very little processing time. Many efficient algorithms have been developed for bundle adjustment and the estimation based on the data used in this project should with no problems be able to run in real time.

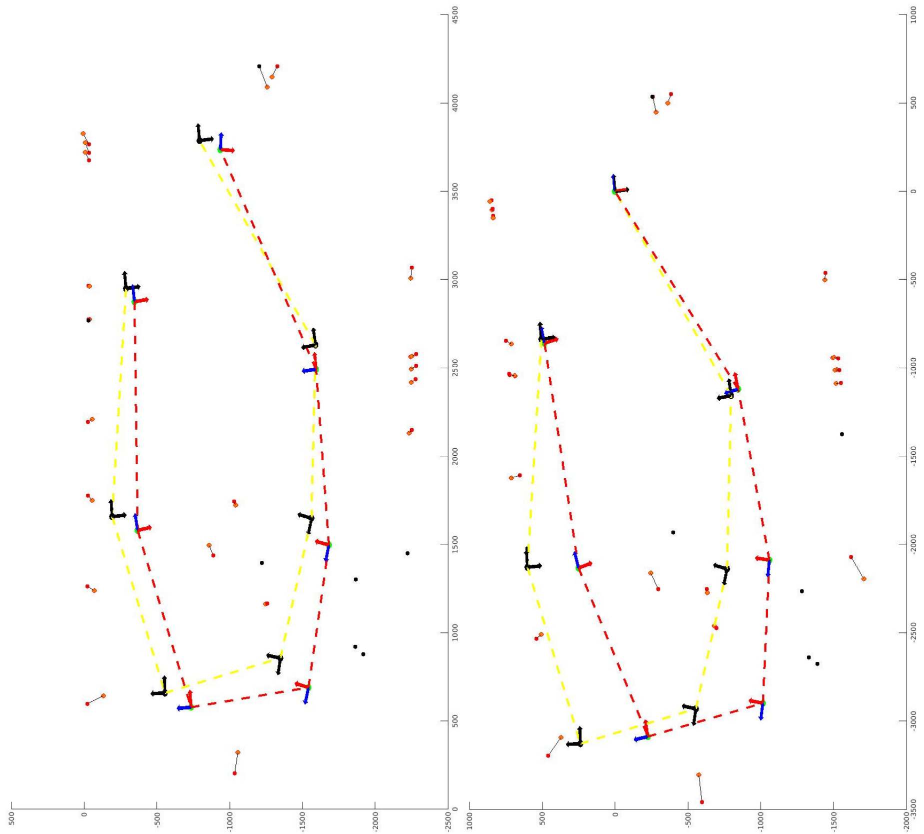
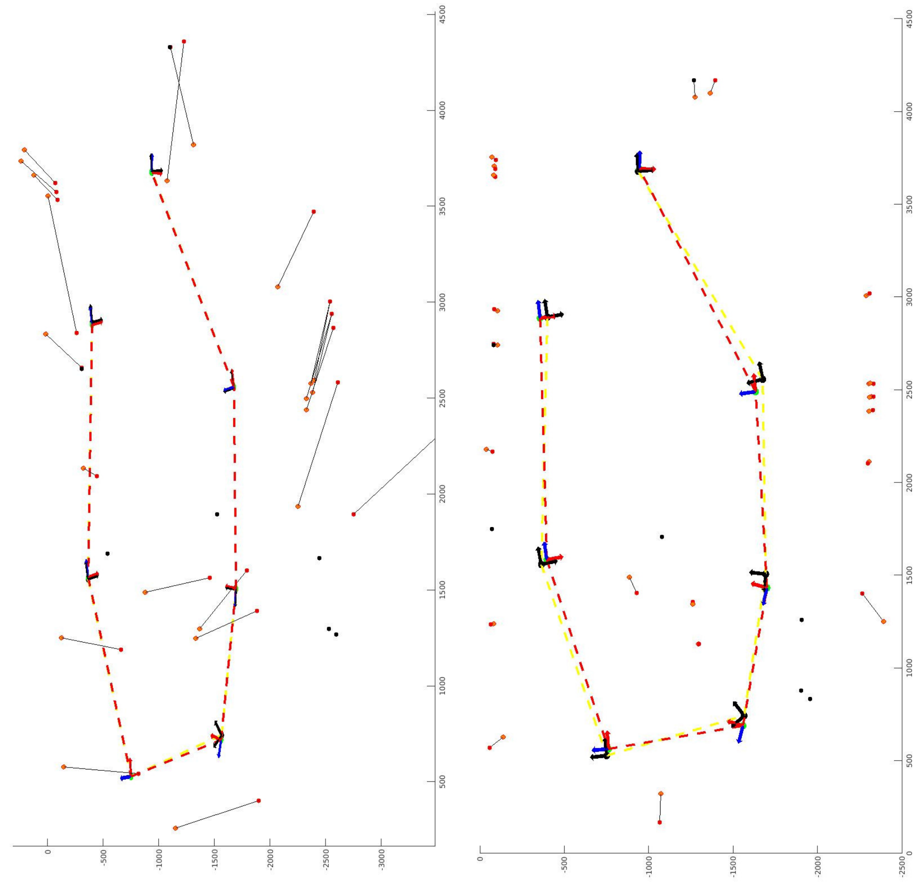


Figure 36: Images show mapping of price labels using deduced reckoning. Left image uses 4 labels with known position to improve positioning. Right image is mapping with no known reference points. Note that absolute coordinates in right image are arbitrary, with no known absolute reference the first location of the robot was defined as origin. However the scale is kept constant thanks to calibration of wheel rotary encoders. The visualization of the true points has been shifted to a representation in this new coordinate system. Red points is true location, orange points are estimated location. Yellow line is estimated path of the robot using deduced reckoning from wheel rotary encoders while red line is the estimation after bundle adjustment.

7.2.3.4 Sensitivity to initial guesses

Using the SLAM algorithm, in the optimal case the solution would be exactly the same no matter the initial guess. However when using iterative minimization that is not the case because of local minima. A good system should though be insensitive to small variations in the initial guess. The sensitivity has been tested by running the SLAM algorithm 50 times adding normal distributed noise to



(a) Linear triangulation using single value decomposition

(b) Linear triangulation combined with bundle adjustment

Figure 37: Comparison of linear point triangulation using singular value decomposition (left) and bundle adjustment (right). Mapping was done using robot pose estimation from roof camera and iterative minimization of reprojection error to find most probable robot pose before labels were estimated. Red points are true locations, orange points are estimated locations. Yellow stroke is estimated path of the robot using the roof mounted cameras and red stroke is the estimation after bundle adjustment.

the pose guess achieved from deduced reckoning. The noise is given by

$$\text{Heading: } \sigma = \frac{\pi}{20} \text{ rad}, \quad \mu = 0,$$
$$\text{Position: } \sigma = 200 \text{ mm}, \quad \mu = 0.$$

29 points with known locations were used as references. As can be seen in Figure 38 the estimated path (red line) is just barely affected by the large variance in the initial guesses (yellow lines). This is a very good result indicating a stable method.

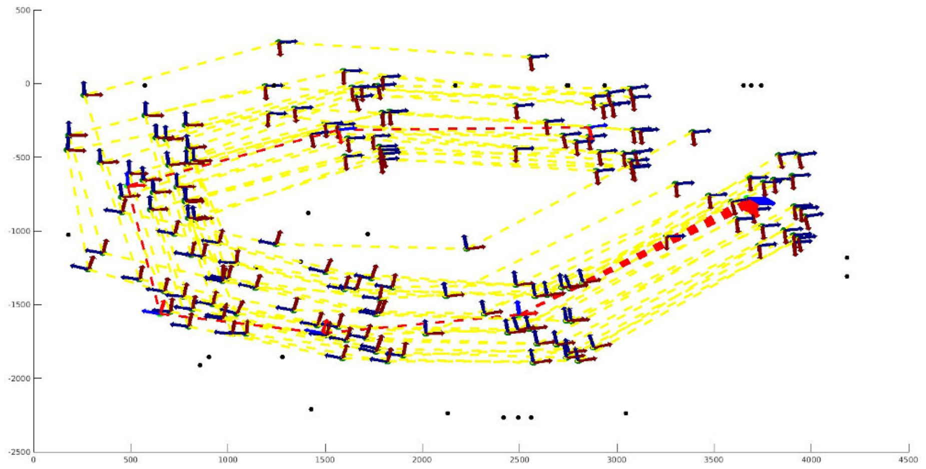


Figure 38: The path measured using deduced reckoning (yellow) was modified with normal distributed noise. As can be seen the algorithm finds the same maximum likelihood path despite the added noise. In the optimal case all red paths would be the same, now only a small spread can be seen at the beginning of the path.

7.3 Warehouse test

The following results were achieved during a field-test in a real warehouse environment as seen in the blueprint given in Figure 21.

7.3.1 Manual measurement of true label position

The manual measurement of label positions in the in-house test described in *7.2.1 Manual measurement of true label position* was done using multidimensional scaling. Because of the vast store size and the large number of labels this was not practically achievable in the warehouse test. Each labels was therefore measured using just three measurements corresponding to x, y and z coordinates. This increases the uncertainty considerably and it does not yield estimation values for the accuracy and precision.

7.3.2 Deduced reckoning

Results of the deduced reckoning is highly dependent of the surface and friction between the robots wheels and the ground. In difference from the soft textile carpet in the test environment the floor in the warehouse is an epoxy coated concrete floor. The result of a test drive in the store is visualized in Figure 39 where the deduced reckoning is plotted together with manual measurements. As in earlier tests it can be seen that the main source of error is the drift created by non exact angle measurements.

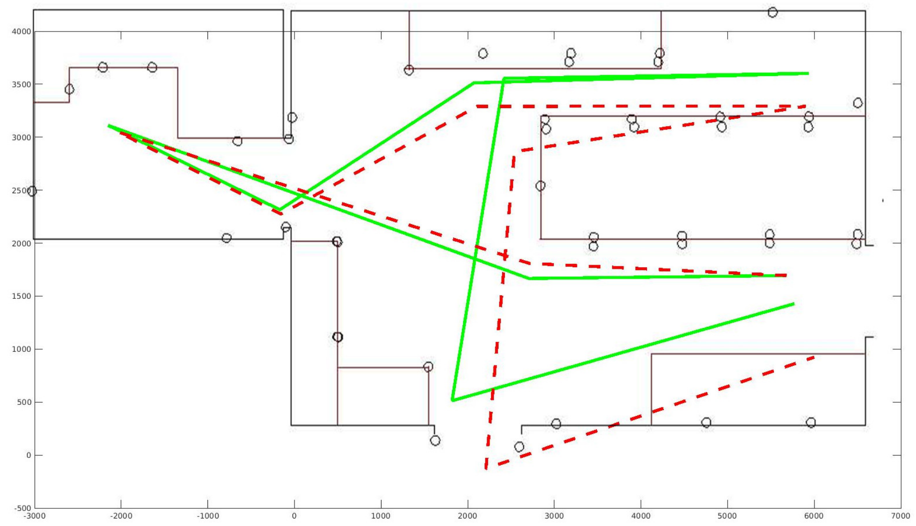


Figure 39: The drift using deduced reckoning (red) is visualized together with real path (green). Deduced reckoning creates only relative measurements and the path has thus been rotated using an iterative minimization algorithm with weights decaying as $w = (N - n)^2$ where the first measurement point is at $n = 0$ and the last at $n = N$.

7.3.3 Mapping accuracy

The results in the following section are based on a path through two of the rooms in the warehouse with 12 stops for mapping. As follows the result in the warehouse lack the accuracy of the inhouse tests. The reasons for this can be derived to the linear guessing method using singular value decomposition as a primer for the bundle adjustment. These problems, and their suspected causes, are further discussed in section *9 Discussion of results.*

7.3.3.1 Linear vs. bundle adjustment

As can be seen in Figure 40 the linear estimation shows extremely poor results. Due to this the initial guess for the bundle adjustment is far from the true optimum. In Figure 41 it can be seen that the bundle adjustment can't correct for an error this big. In Figure 40 many price labels can be seen to be on the opposite side of the robot than in true life, this yield peaks in the iterative optimization function that is hard to cross and it can thus not be expected that the bundle adjustment would solve a situation like this.

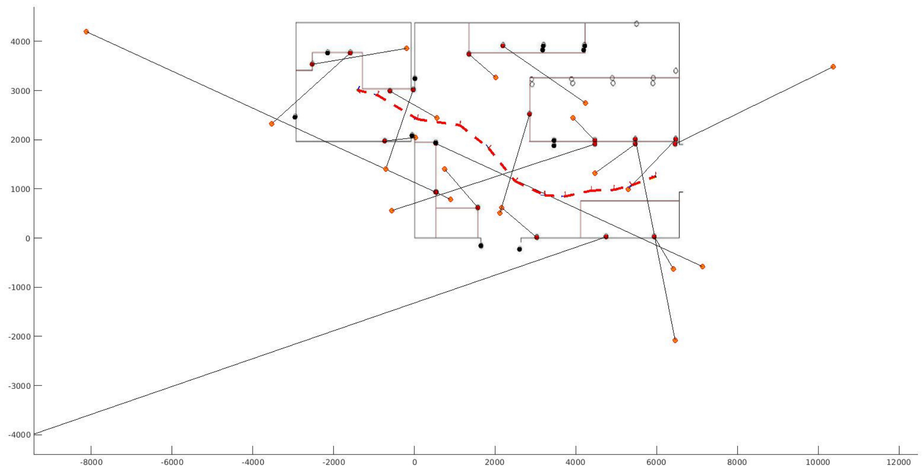


Figure 40: Using singular value decomposition as linear estimation method to estimate label positions in the warehouse turns out to be insufficient in this experiment.

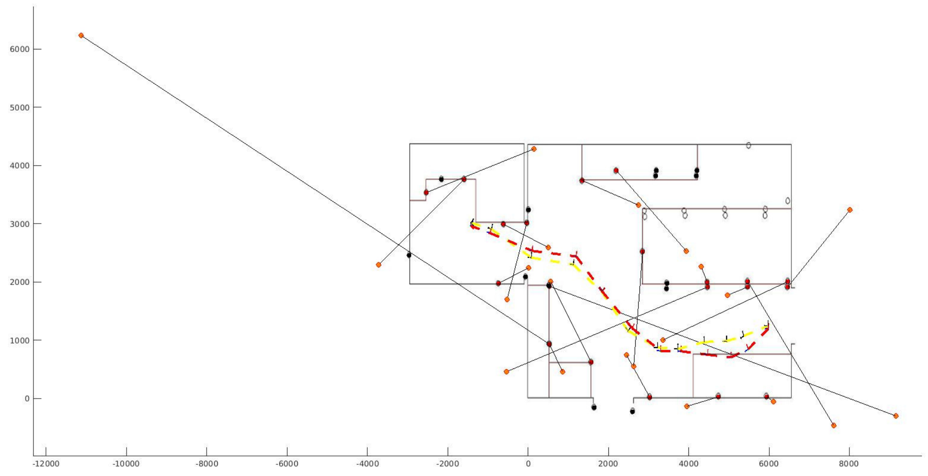


Figure 41: The linear estimation used as guess for the bundle adjustment is too far from the truth to result in a true estimate using the iterative bundle adjustment. The resulting error is however improved to approximately $\frac{2}{3}$ of the error using linear method.

By inspecting the reprojection error of each estimated point an accuracy measure can be achieved. This measure can be used to ignore points that are badly placed by the linear estimation before the bundle adjustment is used. This will improve the results considerably but just a fraction of the unknown point's positions will be estimated. See Figure 42.

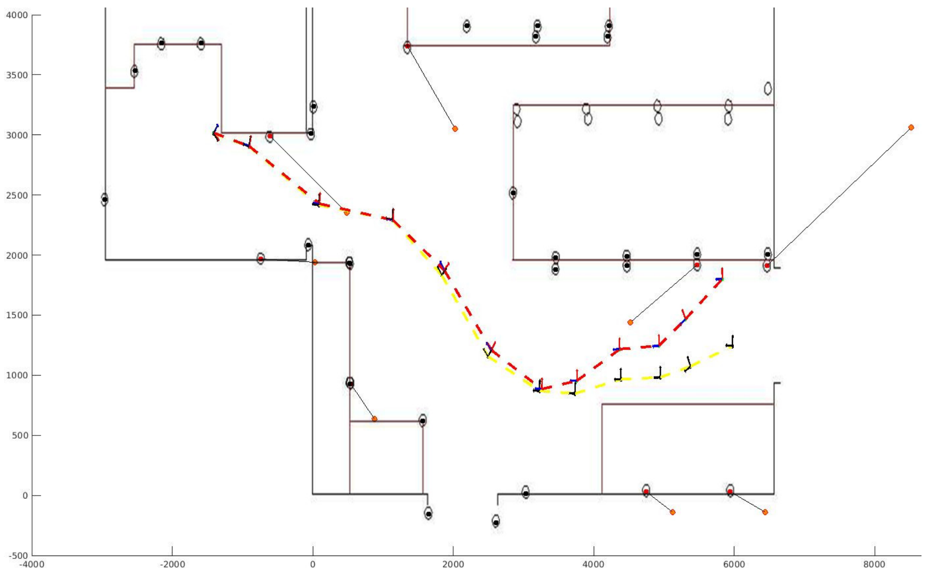


Figure 42: When ignoring uncertain points in the bundle adjustment a better estimation of both path and label positions is achieved, however many of the labels are retained without estimated position.

7.3.3.2 Sensitivity to initial guesses

The result is clearly a bad estimation given by a non optimal local minimum. This is also clear when looking at the response to disturbance in the initial guess. The noise is, as in the in-house test environment, given by the normal distribution with

$$\text{Heading: } \sigma = \frac{\pi}{20} \text{ rad}, \quad \mu = 0,$$
$$\text{Position: } \sigma = 200 \text{ mm}, \quad \mu = 0.$$

As can be seen in Figure 43 the estimated path takes big differences when the guess changes just a bit. This is a clear sign that the result is not trustworthy.

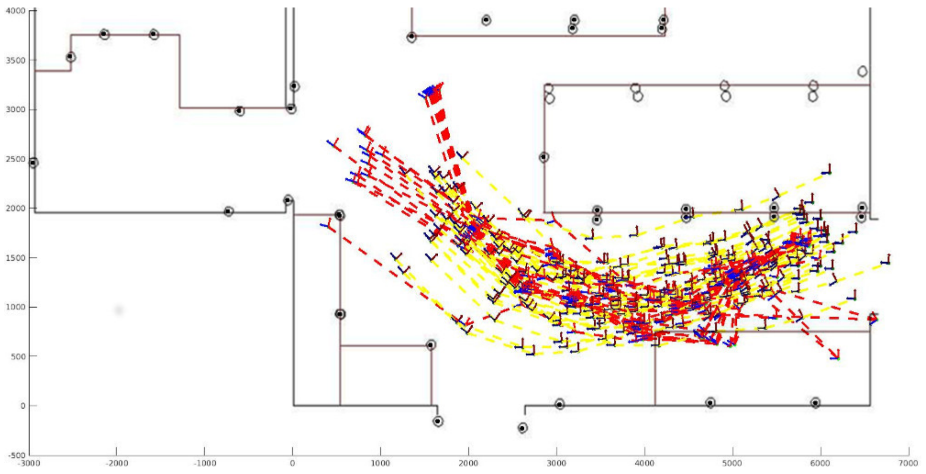
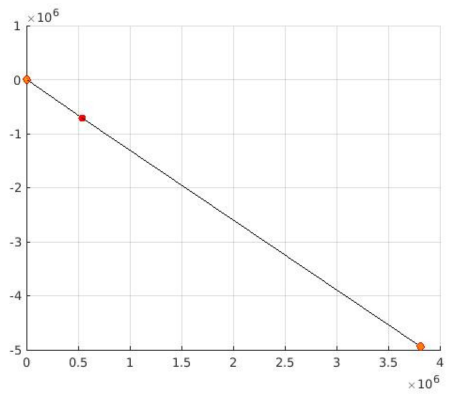


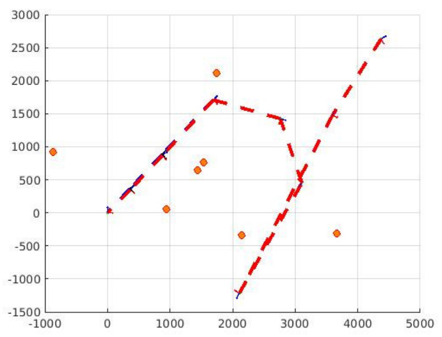
Figure 43: In a case of a poor estimation the estimated path of travel (red) changes much when the initial guess (yellow) is changed.

7.3.3.3 Direct or piece-wise estimation

With a dense data point set and good position estimates from deduced reckoning the estimation of all poses in a single bundle adjustment generates good results. However when the deduced reckoning offers a large drift or uncertainty or when the set of detected data points is sparse the results are improved when using piece-wise estimation of the robots pose as an increasing subset iteratively analyzed with the bundle adjustment. In Figure 44 a case with extremely sparse set of data point labels is presented. In a) all positions are at once estimated using the complete formulation of the bundle adjustment, in b) the first two positions were estimated using bundle adjustment and thereafter one new location was added in each iteration until the complete problem was solved. As can be seen the set-up is an extreme case where the relation of seen labels yields low information and the deduced reckoning must be highly trusted. Without piece-wise estimation the bundle adjustment find an absurd solution, in opposite when using increasing subsets the algorithm find a more plausible solution.



(a) Solution of the whole problem stated as a single bundle adjustment problem. The solution is found in an absurd minimum.



(b) Solution using 8 on each other following bundle adjustments using increasing subsets of mapping locations.

Figure 44: Bundle adjustment of a sparse data problem where the iterative approach of prediction using increasing subsets show large value. Graph unit in millimeters.

7.4 Line detection

The line detector developed during the project shows far better results than any of the other published methods found. In the following section a broken down analysis of the fundamental steps is presented. The stroke width algorithm and the clustering are first presented on an image of black tracks on a gray slightly textured background, seen in Figure 45. The line filtering is then presented with an image of a highly textured floor. The complete algorithm is then presented on a number of hard to find lines in a variety of background surfaces. Finally the algorithm is compared to the basic methods often used as foundation in other algorithms.

7.4.1 The stroke width algorithm

In Figure 46 the steps of the algorithm can be seen. The first step visualized in a) is the detection of start and stop pixels on the border of the lines. These are found by thresholding the gradient and it can be seen that most of the non-bordering pixels are not marked although there are several outliers especially around the lighter colored crack in the floor. Figure b) shows the stroke width beginning and ending on the bordering pixels. Black strokes are filtered away due to not finding a nonzero end pixel before the maximum stroke width was reached. Red lines are filtered away due to non opposite gradients in the start and end pixel (a deviation of 10° was allowed). Figure c) and d) visualizes the stroke angle and stroke length as a color for each pixel that is a centroid of a stroke. These sparse images are not necessary to create, most pixels are zero and to save memory space only the nonzero values can be stored in a dictionary together with their image coordinates.

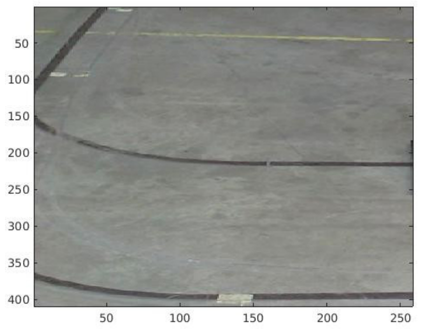


Figure 45: A view of some dark lines on a brighter and slightly textured floor.

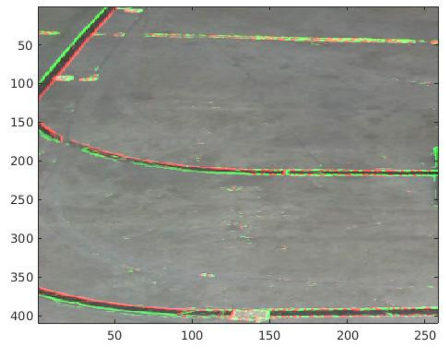
If the method is set to look for dark lines a line brighter than the background will result in strokes going in the opposite direction, out from the line, and a corresponding border pixel will thus not be found. The only difference when looking for a dark or bright line is if the algorithm is looking along the gradient (bright line) or in the opposite direction (dark line). The strokes from the algorithm looking for dark lines can be seen in Figure 47.

7.4.2 Line clustering

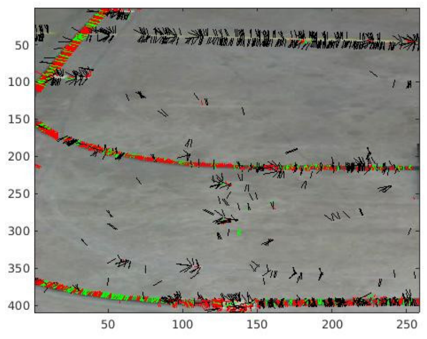
The lines are clustered using DBSCAN density based clustering algorithm to find the different line segments. This is useful for the navigation. If the robot can see several segments of lines it will need to reason about which line it is currently following. This can be done if the lines are segmented using a clustering algorithm. The result of a DBSCAN clustering can be seen in Figure 48 where three different line segments are found. The clustering is performed in four dimensions, the image coordinates, the stroke length and the stroke direction. This allows a smooth transitioning curve with a constant width to be clustered together even if there are some gaps in the detections along the lines. A sharp 90° angle will be clustered together if the detections along it are dense.

7.4.3 Line shape filtering

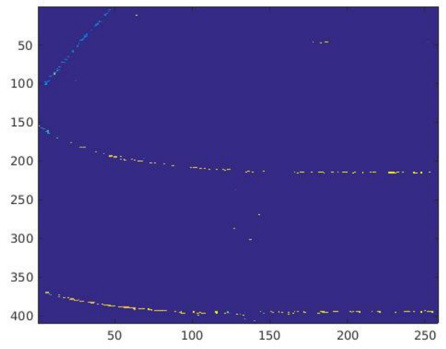
A line has been defined to be considerably longer than wide. If square or circle markers exist on the ground and their side length or radius is within the stroke width searched for they will pass through the stroke width filter. This is due to the fact that they also have gradients of the opposite directions on the opposite sides. The result of this without a length to width ratio filter is displayed in Figure 49. These are filtered away by defining a minimum ratio between the two points within the same segment that are furthest apart and the average of the



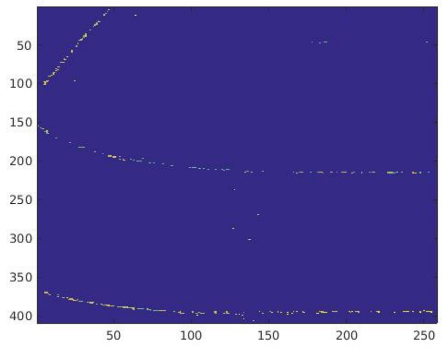
(a) Start and end points of strokes indicated in green and red. Points are given by thresholding the magnitude of the gradient.



(b) The resulting strokes. Black strokes did not pass the length filter and red did not pass the maximum angle difference.



(c) Color indicating the stroke angle marked at the center of each stroke that passed through the filter.



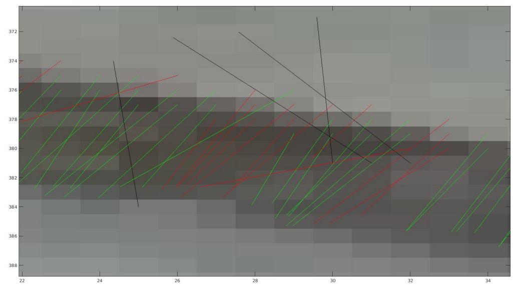
(d) Color indicating the stroke length marked at the center of each stroke that passed through the filter.

Figure 46: Detection of lines on a slightly textured background. Image c) and d) shows the four parameters of each stroke that passed through the stroke length and delta angle filter: image coordinates, stroke length and stroke angle.

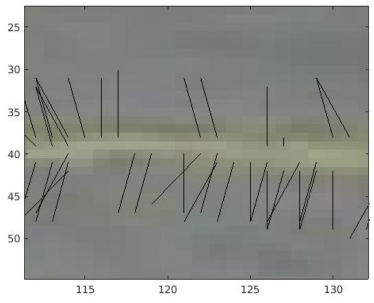
stroke width contained in the cluster. Using a 1:10 filter the results are given to the left in Figure 54.

7.4.4 Detection of lines

The detection of lines has been tested thoroughly on several surfaces ranging from asphalt and concrete to wooden floors. Results can be seen in Figure 50-52. For each tested surface six images are displayed. Top left shows the raw image from the onboard camera. Top right is the top-down view achieved by remapping the pixels using the homography transform. Middle left shows the detected start pixels as green markers and stop pixels as red markers. Middle right shows a



(a) Closeup of the strokes on a segment of the line. The red strokes with to big difference in angle between the gradients on the two sides can be seen to be unreliable in direction and is thus filtered away.



(b) A closeup of the strokes from a brighter colored line using a detector looking for dark lines. The strokes follow the negative gradient and thus can't find a corresponding border pixel, thus the line is filtered away.

Figure 47: The strokes on a detected line versus a line with the wrong color.

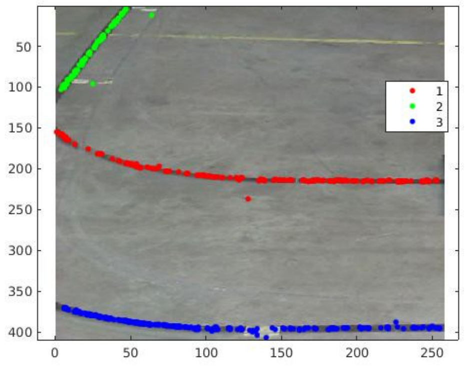
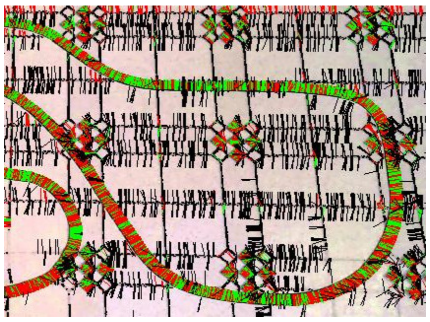
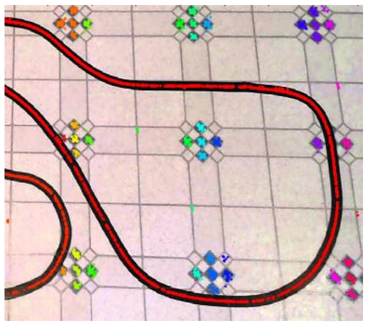


Figure 48: Clustering of the strokes using DBSCAN based on the four parameters, image coordinates, stroke length and stroke angle.

color coded map representing the stroke width for each pixel. Left bottom represent the probability score in a color coded map. Right bottom marks all detected pixels in a line segment using color coding for different segments. The numbering of clusters in the right bottom image in Figure 51 and 52 indicates that smaller clusters not fulfilling the length:width ratio have been dropped in the morphological filter stage.



(a) Dark square markers on the floor with parallel edges and a width within the specified line width will pass through the stroke width filter.



(b) After the clustering square markers will be marked as their own line segments. These can be filtered away by looking at the relation of the line length and mean stroke width.

Figure 49: The effect of not using a line segment length to stroke width ratio filter is displayed above.

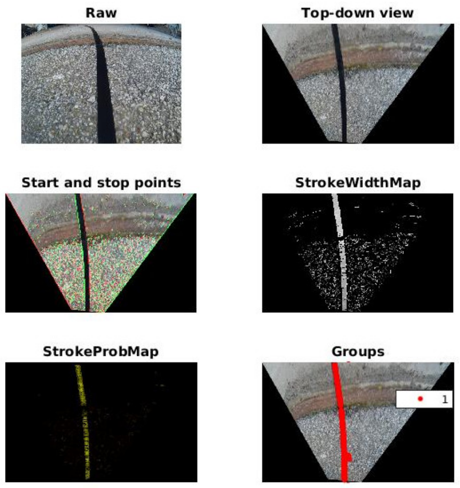


Figure 50: *Detection of a dark line on a mixed concrete-asphalt ground*

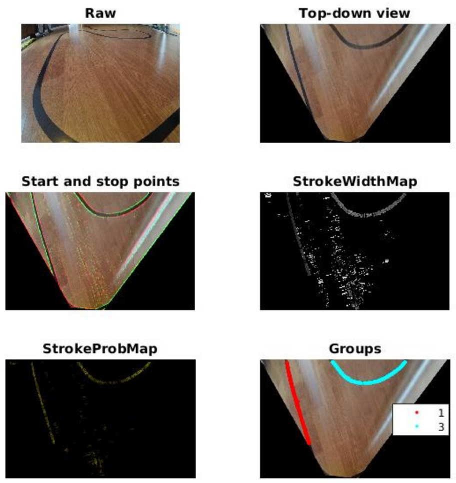


Figure 51: *Detection of two dark line segments on a wooden floor*

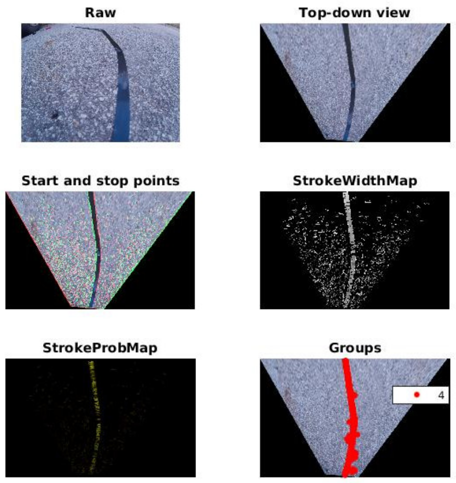


Figure 52: *Detection of a dark line on asphalt ground*

7.4.5 Comparison to commonly used algorithms

A comparison to three other common methods can be seen in Figure 53 to 55. The other methods used to compare are pure fixed value thresholding, Otsu thresholding and Hough line transform. The other methods are not complete line detection algorithms but shows the foundation used in many algorithm. The method developed during the project shows near perfect results on the three test cases while the other methods will require lots of tuning and filtering to give some useful information.

The fixed value thresholding has been manually tuned to fit all test cases as good as possible. Otsu's method automatically calculated the optimal threshold value for every image by maximizing the inter-class variance and minimizing the intra-class variance of the pixel values. The Hough line transform is carried out on an image preprocessed with a Canny edge detector. The Hough line transform tries to find straight lines that will fit to as many pixels as possible during a continuous or mostly continuous path.

As can be seen in Figure 53 to 55 the method developed during the project shows state of the art results in the line following detection at the same time as it is possible to run in real time on limited hardware. As expected it can also be seen that the fixed value thresholding is not generalizing enough to deal with images in different light conditions and colors. The Otsu thresholding manages to extract a bit more information of where the line is but barely more than making it usable as prepossessing or part in a bigger chain of operations. The Hough line transform can in some cases find the straight segments of the line but will also find a lot of other things. Especially in Figure 54 where the background is covered with a mesh the Hough line transform hardly finds anything else than background elements.

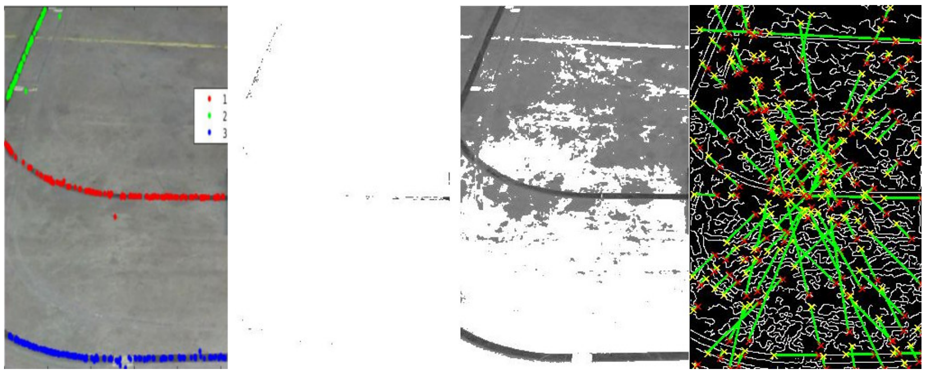


Figure 53: A comparison of the line detection algorithm developed during this project (most left image) and three other commonly used algorithms. From left: In project developed algorithm, fixed value thresholding, Otsu thresholding and Hough line transform.

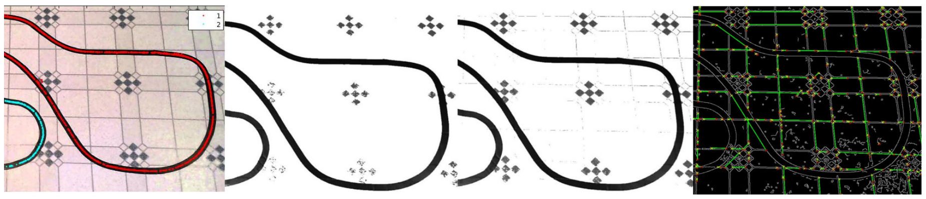


Figure 54: A comparison of the line detection algorithm developed during this project (most left image) and three other commonly used algorithms. From left: In project developed algorithm, fixed value thresholding, Otsu thresholding and Hough line transform.

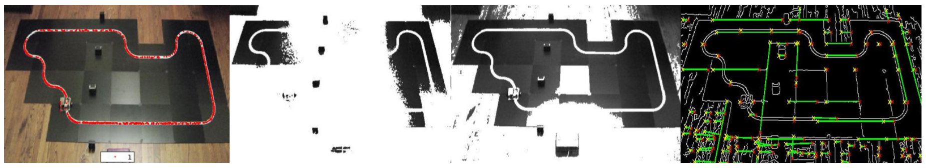


Figure 55: A comparison of the line detection algorithm developed during this project (most left image) and three other commonly used algorithms. From left: In project developed algorithm, fixed value thresholding, Otsu thresholding and Hough line transform.

7.4.6 Real time performance

For line following applications it is important that the algorithm runs in real time and can react fast on environmental conditions. Since the algorithm needs to be able to execute onboard a mobile robot the method was optimized using the Cython extended programming language as a complement to the non calculation heavy parts implemented in Python. Standard algorithms were used as implemented in the OpenCV library. The OpenCV library contains several assembler level optimizations to avoid delays such as data hazards and pipeline stalls and fully make use of single instruction multiple data architectures. It was thus decided without further investigation that no more time would be spent to optimize this. Instead the optimization was done only on the heavy custom methods implemented in this project. The speed up is on the scale of several orders of magnitude which is a very good result. Even more speed up can be expected by further optimizing the code in C or partly implementing in assembler.

8 Future work

After finishing this project the recommendation is that further work focuses on two main areas except the adaption for large scale production: minimizing the time consumed for the mapping and improving the stability. The small scale tests in the $10m^2$ room prove that the mapping using the current setup and software can yield more than satisfying results although the algorithms are far from finally tuned. The failed experiments in the warehouse however need to be addressed to find the cause and possibly a solution to improve stability. By looking at what improvements can be done in the methods and in the implementation of each step a far more robust system could be achieved.

To simplify further work and encourage comparisons the complete data set from both the in-house test and the warehouse including system calibrations has been made publicly available online. Instructions how to get the data set are attached in Appendix A.

8.1 Analyze large scale results

The warehouse test was performed in the end of the master's thesis and no time was available to investigate what the limiting factors were. The bad linear estimation used as a guess to initialize the bundle adjustment can be assumed to be a problem considering its large deviation from ground truth. This can be confirmed by using the manually measured positions of the labels as an input guess to the bundle adjustment. The accuracy of the manually measured positions of the labels is, as described in section *7.3.1 Manual measurement of true label position*, considerably lower than in the in-house test. This might affect the stability of the test regarding noise in initial test presented in section *7.3.3.2 Sensitivity to initial guesses.* This might be tested by adding noise to the manual measurements in the in-house test and evaluating the results of the same test repeated.

8.2 Improving mapping speed

The mapping today is a very slow process. The biggest issue is the time consumed to detect one digital label's LED flash at a time. The time spent to detect a single flash is set to 16 seconds to minimize the possibility of false detections and improve the recall. This time spent for one detection should be possible to reduce without drastic changes. The Pricer system allows only fixed steps of flashing time which may pose a problem when it comes to faster detection. Also the fact that the Pricers API does not allow any feedback of the state of the LEDs introduces a need to wait for a considerable amount of time between LED flashes to be sure that only one LED is flashing at a time. These problems could be avoided by integrating the system directly with Pricer's system without using the API designed for third party developers.

Some other recommendations for improvement of the speed is discussed bellow and includes the need for more efficient bundle adjustment techniques, optimizing the robot's route through the environment and removing unnecessary flashing and detection of LEDs that does not introduce significant new information.

8.2.1 Efficient bundle adjustment

The bundle adjustment algorithm is today implemented using a nonlinear least-squares solver using a vector-valued function, *lsqnonlin*, implemented in Matlab. These generic minimization algorithms are much slower than methods implemented purely for the use in bundle adjustment. Faster methods have been frequently evaluated in research papers such as [55],[71].

8.2.2 Minimize label flashes

As a complement to just optimizing the route to yield the best views of the labels a reasoning of whether the location gives an improvement of the result or not can be conducted for each label. If the view of a label is close to the same angle as an earlier view of it this label can be skipped in the detection process and thus saving time. Another prioritization between speed and accuracy is if a label already seen from enough views to locate it should be looked for or skipped from the current position.

8.2.3 Optimal route

The choice of the route reflects directly on the number of frames needed and the precision of the result. The route used in the experiment has been manually decided by human input. In the section *5 Navigation and route planning* two different approaches were introduced, including the force field navigation. This method is however not finalized and needs a lot more effort put into it to give better results than the line following approach. By optimizing the locations the frame are captured at the number of needed views of a label can be minimized.

8.2.4 Line following improvements

The line following shows close to perfect results in most conditions tested. Some outliers can be found but will not pose a big problem if a robust method is used to fit the line based on the detected points. Most improvements would consider the time consumption. Although the algorithm does execute in real time today further speed up could make it possible to run on less capable hardware or even embedded microcontrollers.

The similarity measure between points are measured using L_2 norm. The covariance of stroke length and angle is known and thus another similarity measure like Mahalanobis distance could be used to account for the correlation. The Mahalanobis distance is a unitless multi-dimensional distance based on the standard deviation. The distance of a set of vectors $\boldsymbol{X}$ with the mean values given by $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{S}$ can be calculated as

$$d_M(\boldsymbol{X}) = \sqrt{(\boldsymbol{X} - \boldsymbol{\mu})^T \boldsymbol{S}^{-1} (\boldsymbol{X} - \boldsymbol{\mu})}.$$

The registration of point clouds to create a full map and the control of the robot based on the layout of the line has only been briefly touched in this report and are big topics for further studies.

8.3 Improving accuracy

The changes to improve accuracy is in many cases a contradiction to the proposals in the *8.2 Improving mapping speed* section. As an example reducing the number of views will increase the speed of the mapping but reduce the accuracy. In these cases a prioritization need to be done to assess what is the most important feature.

8.3.1 Denser viewpoints of cloud

If the detections of LEDs can be sped up considerably the number of views could be increased resulting in better estimations and less impact of noise. In a real retail store the point cloud representing the digital labels would be extremely denser than what is used in the experiments during the project. This can be expected to increase the accuracy of the mapping. With sparse point clouds every point is important for the localization while with more redundant data the detection of outliers is simplified and zero mean based noise will be more averaged out.

8.3.2 Improving LED detection

The standard deviation of the LED detections presented in the results are typically above 1 pixel and it would be possible to get better estimations by improving the LED flash finding detector. Especially the smoothing step in the detector can result in detected centroids moving away from actual projected centroids. This could be improved by after finding a confirmed detection moving back to original unaltered full-resolution image and performing a sub pixel optimization to find its centroid. Another problem is if the image contain a saturated blob bigger than one pixel around the LED. In that case it is not possible through software to find the true location. This could be solved by taking better control over the exposure time to avoid saturation. Using that technique it is however possible to lose track of LEDs which might be weaker because of projection or view angle. By taking several images at different exposure time both these problems could be avoided but the detection time would be increased.

8.4 Limiting factors

Several improvements can be done to reach a more practical time consumed for the mapping and even better accuracy.

8.4.1 Sequential detection of labels

In this project the price labels have been detected one at a time. Improvement can be done by several different techniques. By flashing several labels at the same time the detection could be assigned to the closest of the from another pose previously detected labels. Using a more traditional SLAM approach all LEDs could flash at the same time and by using matching the stream of incoming points could be merged to a point cloud. An example of a method commonly used to find the corresponding points is Iterative Closest Point (ICP) [30]. Using the possibility to uniquely identify every point makes many methods implementable that otherwise would be impossible or inefficient. A binary method could be used where each label is assigned a minimal unique binary code and each label with a true value at an increasing binary position is set to flash. This method would be highly reliant on definitive detection, if a flashing label is not detected every time it would be identified as another label. A more efficient way would be to implement a unique flash pattern in every label. All signs could then be detected simultaneously resulting in considerable time savings.

8.4.2 Stop of movement for mapping

If the robot should be able to map on-the-go without stopping the cameras need to be synchronized and the algorithm for flash detection modified. With the cameras running independently the maximum time difference between the two closest frames in two cameras can be described by

$$\Delta t = \frac{1}{f_{\text{fps}}} = 100\text{mS}.$$

The distance traveled by the robot during this time is described by

$$\Delta d = S_{\text{speed}} \times \Delta t \approx 50\text{mm},$$

and is typically small. Even though the distance traveled by the robot during this time is small the angle difference can introduce a substantial error into the triangulation. When the robot is turning the difference in angle is described by the rotational speed, ω, and the passed by time

$$\Delta \alpha = \omega \times \Delta t \approx 0.1\pi.$$

A label at a distance, $R = 4000\text{mm}$, away can thus get a worse error of

$$d_{\text{error}} = \Delta \alpha \times R \approx 1256\text{mm}.$$

The SDK distributed by Modcam is not designed to make synchronization of cameras possible although future releases of the SDK might make it possible. During the project the pose of the robot has been held fixed during the collection of frames.

8.4.3 Limited accuracy

The restriction of the results precision seems to be a combination of camera resolution, centroid detection accuracy and camera calibration. The detection of the LEDs centroid is done on a slightly blurred image to reduce impact of pixel flicker due to noise around sharp edges. This estimation of the centroid could be improved by going back to the original image and performing a subpixel interpolation. The detection of centroids is also prone to errors if several pixels around the LEDs are saturated. The camera lens calibration is done using a sixth degree polynomial equation. This could probably be improved by using an eighth degree polynomial which might be necessary due to the cameras' wide view angles.

8.4.4 Sparse data for localization

The relative pose estimation could be improved by combining the detection of labels' LEDs with the detection of other fixed points in the images to create denser point clouds. Such points could be SIFT-points or other as mentioned earlier automatically detected points in the images or physically placed points such as QR-codes.

8.4.5 Limited hardware for deduced reckoning

The deduced reckoning might be improved by combining with an inertial-measurement unit (IMU) to measure acceleration around six axes, rotation and translation. Especially a compass module, or hardware with similar use, to avoid drifting headings might improve the results notable.

8.4.6 Bugs in implementation

Communication between cameras and control center server is done using the Java *ServerSocket* and *Socket* objects. When the communication was performed using the in Modcam built in WiFi-module the transmission of a data package, using Javas object stream, could sometimes freeze for several seconds. This delay causes the implemented image buffer in the cameras to fill up which for a yet unknown reason causes the software in the cameras to crash. Although the reason has not been found it is suspected that a deadlock is introduced in the thread safe monitor causing further serious problems. Due to the already slow mapping combined with these crashes the test in the real warehouse could not be performed as dense as planned.

9 Discussion of results

In the project it has been proved that an accurate mapping of a room based on a small number of price labels is possible. However it has also emphasized the importance of a robust system and it can be established that the construction of such is hard. The test in the warehouse resulted in bad mapping results and the reason is still not ascertained. This shows the need for a stable system that is not receivable to disturbance and that is easy to use and configure. However, the area of SLAM/SfM has been much researched and improving the methods from a proof of concept to a production ready system should not be impossible.

9.1 Reprojection errors and false minima

From the mapping results presented for the test environment and the field-test in the warehouse it is clear that a false local minimum has been found in the latter. The reason for this is hard to find and needs to be addressed as a continuation of the project. The linear estimation is clearly a source of the problem but considering the instability of the algorithm seen even when all labels have known position it is not the only problem. One theory is that the relative poses of the cameras might have been accidentally altered while the robot was moved to the warehouse. Another reason might be the more limited view of price labels. To isolate the cause of the problem the mapping would need to be performed in the original test environment once again. This is however not easily achieved while the equipment was removed and relocated to the warehouse.

To evaluate the quality of a solution the reprojection errors in the cameras are of great value. Looking at the final errors in the test environment gives the following reprojection errors given in pixels

$$\text{Max: } 64.69,\ \text{Mean: } 7.36,\ \text{Median: } 5.16,\ \text{Min: } 0.02$$

The same errors for the test in the warehouse are given by

$$\text{Max: } 99999.00,\ \text{Mean: } 57050.02,\ \text{Median: } 99999.00,\ \text{Min: } 0.58$$

The maximum errors which have the same value as the median indicate a limitation in the calculation accuracy.

9.2 Usefulness

Although the mapping is what has been of greatest focus in this report the other parts are also by them self interesting for the evolution of technology and robotics in retail and other areas for robotic use.

The detection of digital price labels using vision system can be used in many areas other than mapping. The vision system does not need to be placed at a mobile robot, it could be the already existing surveillance cameras or it could be a hand held video camera. Other than positioning of the labels similar detectors could be used to transfer small amount of data using cheap and already installed hardware. By using communication based on only a flashing LED and the surveillance cameras information could be sent using very cheap and small hardware with low energy consumption.

Positioning of the camera using a small amount of known or unknown landmarks can be used as a low resource demanding localization of robots or cameras compared to the more common algorithms using e.g. Scale-invariant feature transform (SIFT) [34], Speeded Up Robust Features (SURF) [29], or Binary Robust Invariant Scalable Keypoints (BRISK) [61].

Different means of navigation during the mapping introduces different pro et contra. The commonly used random patterns are easy to implement but are very ineffective and not predictable. These methods are recommended to avoid due to the large variances in the results they would introduce and the enormous time the robot would have to consume to avoid ambiguities in all places of the environment. The force field navigation which has been briefly touched is a powerful mean of navigation capable of maximizing the results while minimizing the consumed time. Implementation is not straight forward though and much more research would need to be done to propose recommendations of exactly formed methods. Line following is a robust and predictable method that is easy to implement both in the robotic hardware and in the store environment. By choosing a good strategy when placing the line a low time consumption and good results can easily be achieved. If the aesthetic aspect of having a line on the floor is a problem other methods exist, developed for the industry, which are based on among others magnetic fields. These methods however demands more physical changes in the environment and greatly increases the installation cost.

The line following method developed can be used in many cases where high performance line following is needed, either for good accuracy or for high speed navigation such as line following contests. The method is efficient enough to run a high speed navigation on a mobile robot equipped with a Raspberry Pi 3 compact computer. Line following is also used in many industrial applications where automated transports are used.

10 Conclusion

During the project high precision and accuracy mapping has been achieved and as a proof of concept the project is considered a success. As a result the need for a robust system has also been emphasized during the warehouse test. Recommendations for design and further work have been formed. The main drawback with the current implementation is the slow mapping process and the lack of robustness. Further work should focus on fully automated and faster mapping with higher robustness.

Different methods and location systems were compared and it can be seen that the minimum hardware case where no fixed external hardware beyond the robot and Pricer Electronic Shelf Labels system, that is already installed in more than 13'000 retail stores world wide, shows more than adequate results. With a good designed system without complementary hardware almost as good results are achievable resulting in installation savings of time and money. Even though the fixed cameras on the room walls increased the precision of the mapping they can be considered not worth the cost. The results are satisfying for a retail environment without them and by not using them the installation is simplified.

It can also be seen that the iterative minimization using bundle adjustment proposed in this report generates far better result than the linear methods that are not seldom used.

The navigation planning was briefly touched and a novel method for line following was developed as a navigation aid for the project. The new method developed shows state of the art results compared to the current methods commonly used today. The algorithm detects and clusters line segments with high precision and recall while being efficient enough to execute in real time in close to any PC. Several improvements over the commonly used algorithms were introduced. A common line following control method after image segmentation is to calculate the center of gravity of the pixel segments and drive towards it. This method has proved to work reasonably well but the method described here allows several improvements of the robot control. Using center of gravity the robot will cut corners short without any possibility to impact how much. If two lines gets visible in the field of view the robot will drive in the center between the lines. By vectorizing the lines, as showcased in this project, and cluster them it is possible to specify exactly how much or little the robot is allowed to cut corners. For fast navigation the path can be optimized to be as short as possible and for exact navigation the path can be exactly followed.

References

[1] Android open source mobile operating system. http://www.android.com. Accessed: 2016-05-31.

[2] Axis Communications network cameras and ip-solutions. www.axis.com. Accessed: 2016-06-07.

[3] BC547 datasheet. http://www.fairchildsemi.com/datasheets/BC/BC547.pdf. Accessed: 2016-06-08.

[4] Cython optimizing static compiler for python and cython. http://cython.org/. Accessed: 2016-06-27.

[5] Debian open source operating system. http://www.debian.org. Accessed: 2016-05-31.

[6] FTDI-chip usb interface modules. http://www.ftdichip.com/. Accessed: 2016-06-07.

[7] IRobot Open Interface specification. http://cfpm.org/ Accessed: 2016-06-07.

[8] IRobot Roomba robotic vacuum cleaner. http://www.irobot.com/. Accessed: 2016-05-31.

[9] Java programming language. http://www.java.com. Accessed: 2016-05-31.

[10] Matlab the language of technical computing. http://mathworks.com/products/matlab/. Accessed: 2016-06-09.

[11] Matlabcontrol a java api to interact with matlab. http://code.google.com/archive/p/matlabcontrol/. Accessed: 2016-06-09.

[12] Microsoft Windows operating system. http://www.microsoft.com/windows. Accessed: 2016-05-31.

[13] Modcam connected compact sensor. http://www.modcam.com/. Accessed: 2016-05-31.

[14] Numpy package for scientific computing with python. http://www.numpy.org/. Accessed: 2016-06-27.

[15] OpenCV open source computer vision. http://opencv.org/. Accessed: 2016-06-14.

[16] Pi4J library. http://pi4j.com/. Accessed: 2016-06-07.

[17] Pricer electronic shelf label system. http://www.pricer.com/. Accessed: 2016-05-31.

[18] Python programming language. http://www.python.org/. Accessed: 2016-06-27.

[19] Raspberry Pi open source compact computer. http://www.raspberrypi.org/. Accessed: 2016-05-31.

[20] Raspian open source operating system. http://www.raspbian.org/. Accessed: 2016-06-07.

[21] RXTX library. http://rxtx.qbang.org. Accessed: 2016-06-07.

[22] The Central Limit Theorem random, formerly virtual laboratories in probability and statistics. http://www.math.uah.edu/stat/sample/CLT.html. Accessed: 2016-06-03.

[23] Syed Riaz Ahmed. Applications of data mining in retail business. In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, volume 2, pages 455–459. IEEE, 2004.

[24] Josep Aulinas, Yvan R Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. In *CCIA*, pages 363–371. Citeseer, 2008.

[25] Andrew Reed Bacha. Line detection and lane following for an autonomous mobile robot. 2005.

[26] Patrick Baker, Cornelia Fermüller, Yiannis Aloimonos, and Robert Pless. A spherical eye from multiple cameras (makes better models of the world). In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–576. IEEE, 2001.

[27] Christian Balkenius. *Natural intelligence in artificial creatures*, volume 37. Lund University, 1995.

[28] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. Math*, 3(1):133–181, 1922.

[29] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[30] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.

[31] Michael J Black and Anand Rangarajan. The outlier process: Unifying line processes and robust statistics. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 15–22. IEEE, 1994.

[32] Robert C Bolles, H Harlyn Baker, and David H Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1(1):7–55, 1987.

[33] Tom Brijs, Gilbert Swinnen, Koen Vanhoof, and Geert Wets. Using association rules for product assortment decisions: A case study. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 254–260. ACM, 1999.

[34] Wilhelm Burger and Mark J Burge. Scale-invariant feature transform (sift). In *Digital Image Processing*, pages 609–664. Springer, 2016.

[35] Martin Byröd, Klas Josephson, and Kalle Åström. Fast optimal three view triangulation. In *Computer Vision–ACCV 2007*, pages 549–559. Springer, 2007.

[36] Michael Chau and Margrit Betke. Real time eye tracking and blink detection with usb cameras. *Boston University Computer Science*, 2215(2005-2012):1–10, 2005.

[37] Chu-Song Chen and Wen-Yan Chang. On pose recovery for generalized visual sensors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(7):848–861, 2004.

[38] Thomas F Coleman and Yuying Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on optimization*, 6(2):418–445, 1996.

[39] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[40] Austin Eliazar and Ronald Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *IJCAI*, volume 3, pages 1135–1142, 2003.

[41] Boris Epshtein, Eyal Ofek, and Yonatan Wexler. Detecting text in natural scenes with stroke width transform. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2963–2970. IEEE, 2010.

[42] Daniel Falk. Cognitive vision for line following using stroke width vectorization, 2016. Manuscript submitted for publication.

[43] Olivier D Faugeras and Francis Lustman. Motion and structure from motion in a piecewise planar environment. *International Journal of Pattern Recognition and Artificial Intelligence*, 2(03):485–508, 1988.

[44] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003.

[45] Andrea Garulli, Antonio Giannitrapani, Andrea Rossi, and Antonio Vicino. Mobile robot slam for line-based environment representation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2041–2046. IEEE, 2005.

[46] Michael P Georgeff and Amy L Lansky. Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682, 1987.

[47] Kristen Grauman, Margrit Betke, James Gips, and Gary R Bradski. Communication via eye blinks-detection and duration analysis in real time. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–1010. IEEE, 2001.

[48] George Green. *An essay on the application of mathematical analysis to the theories of electricity and magnetism.* author, 1828.

[49] U Grenander. The nyquist frequency is that frequency whose period is two sampling intervals. *Probability and Statistics: The Harald Cramér Volume*, page 434, 1959.

[50] Daniel Hahnel, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 1, pages 206–211. IEEE, 2003.

[51] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[52] Amir M Hormozi and Stacy Giles. Data mining: A competitive weapon for banking and retail industries. *Information systems management*, 21(2):62–71, 2004.

[53] John Illingworth and Josef Kittler. A survey of the hough transform. *Computer vision, graphics, and image processing*, 44(1):87–116, 1988.

[54] Tony S Jebara and Alex Pentland. Parametrized structure from motion for 3d adaptive feedback tracking of faces. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 144–150. IEEE, 1997.

[55] Yekeun Jeong, David Nister, Drew Steedly, Richard Szeliski, and In-So Kweon. Pushing the envelope of modern methods for bundle adjustment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(8):1605–1617, 2012.

[56] Michael Kaess and Frank Dellaert. Visual slam with a multi-camera rig. 2006.

[57] Nahum Kiryati, Yuval Eldar, and Alfred M Bruckstein. A probabilistic hough transform. *Pattern recognition*, 24(4):303–316, 1991.

[58] Zuzana Kukelova, Tomas Pajdla, and Martin Bujnak. Fast and stable algebraic solution to l2 three-view triangulation. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 326–333. IEEE, 2013.

[59] Charles L Lawson and Richard J Hanson. *Solving least squares problems*, volume 161. SIAM, 1974.

[60] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[61] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[62] Richard S Palais. A simple proof of the banach contraction principle. *Journal of Fixed Point Theory and Applications*, 2(2):221–223, 2007.

[63] Robert Pless. Using many cameras as one. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–587. IEEE, 2003.

[64] Aveek Purohit, Zheng Sun, Shijia Pan, and Pei Zhang. Sugartrail: Indoor navigation in retail environments without surveys and maps. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 300–308. IEEE, 2013.

[65] Xiaojun Qi and Yinbing Ge. A real-time vision-based outdoor navigation system for the wheelchair robot.

[66] Henrik Stewénius, David Nistér, Magnus Oskarsson, and Kalle Åström. Solutions to minimal generalized relative pose problems. In *Workshop on omnidirectional vision*, volume 1, page 3, 2005.

[67] Christopher E Strangio. The rs232 standard. *Online Document*, 10, 2006.

[68] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010.

[69] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[70] Pang-Ning Tan et al. *Introduction to data mining.* Pearson Education India, 2006.

[71] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustmenta modern synthesis. In *Vision algorithms: theory and practice*, pages 298–372. Springer, 1999.

[72] Manish Verma, Mauly Srivastava, Neha Chack, Atul Kumar Diswar, and Nidhi Gupta. A comparative study of various clustering algorithms in data mining. *International Journal of Engineering Research and Applications (IJERA)*, 2(3):1379–1384, 2012.

[73] Teddy N Yap Jr and Christian R Shelton. Slam in large indoor environments with low-cost, noisy, and sparse sonars. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1395–1401. IEEE, 2009.

[74] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[75] Thomas Guthrie Zimmerman. System and method for performing inventory using a mobile inventory robot, April 6 2010. US Patent 7,693,757.

Appendices

(A) SLAM data set

(B) Deduced reckoning or dead reckoning

(C) Paper A

Appendix A

SLAM data set

Data set for SLAM in the in-house room as well as in the warehouse is made available online for future comparison and further work. Data set includes detections in calibrated images, position and heading from deduced reckoning and true manually measured label positions. For further info read instructions on the download page and the included read-me file.

http://da-robotteknik.se/learningrep/slam_masters.php

Appendix B

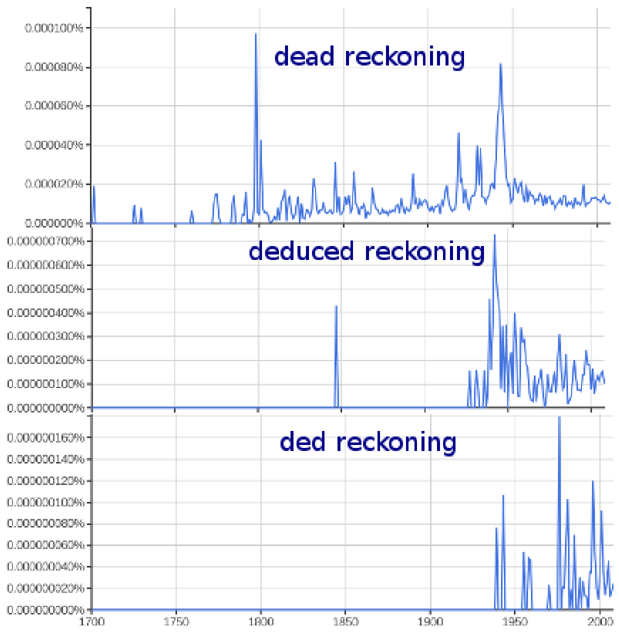
Deduced reckoning or dead reckoning

It is clear that in a historical view the correct term to use has been dead reckoning. The term has been used for hundreds of years to describe the navigation technique where the current position is estimated by using the vessel's speed, course and deserted time. In the 20th century the use of the term deduced reckoning became popular. The source of it is unclear even though it holds a more intuitive explanation of the process than dead reckoning. In the newspaper *Oakland Tribune* you could in 1947 read

> *A friend of mine who prides himself on being a precisionist, went to see "Dead Reckoning" the other night and I asked him how he liked it. "Oh, the picture was fine," he said, "but the title ..." "What's wrong with the title?" I asked. He looked down his nose at me. "Theres no such thing as 'dead reckoning', "he replied. "It's 'Ded' Reckoning, which is short for 'Deduced Reckoning'. Ask any navigator."*

In US patent *US6046565 A* the same mistake is done

> *Ded-reckoning, often called dead-reckoning in error, is a shortening of the term deduced reckoning.*



Using Google Books Ngram viewer the following usage statistics from 1700 to 2008 was found. Although it can be seen that dead reckoning still today is the most common term the trends shows that deduced reckoning is up and coming. This latter term is also what is commonly used in professional situations today and is why the use in this report was decided in favor for that.

Appendix C

Paper A

Daniel Falk. *Cognitive vision for line following using stroke width vectorization*, 2016. Manuscript submitted for publication.

Cognitive vision for line following using stroke width vectorization

Falk, Daniel[1]

[1] DA-Robotteknik, Lund, Sweden
Email: daniel@da-robotteknik.se

Abstract
Line following in mobile robotics is both popular in industrial environment and recreational competitions. This paper aim to propose a method for visual line tracking and segmentation which allows real time analysis at high frame rate. Possible curvilinear line segments are found by searching the image for constant width strokes which are defined by adverse bordering gradients. The method is novel using an approach based in text detection. Results are state-of-the-art with the ability to detect either dark or bright lines in heavily textured background surfaces with high precision rate. By clustering the found line segments, multiple lines can be in the view without affecting the tracking. Tests in diverse environments prove that the method is superior to the commonly used algorithms.

Keywords
Line following, navigation, computer vision, cognitive science, stroke width transform

1. Introduction

Line following is a popular mean of navigation for mobile robotics in the industry mainly because of its easily predictable character. Many of the biggest international competitions in robotics include line following where the speed and robustness of the control are tested. Here a new algorithm for detecting curvilinear lines is presented. It is developed based on the fundamental steps of Stroke Width Transform [1]. The Stroke Width Transform turned out to be a revolution of the OCR text detection in natural images when it was published in 2010. This paper aim to propose a method considerably less process demanding which allows real time analysis at high frame rate. Possible line segments are found by searching the image for constant width strokes which are defined by adverse bordering gradients. By clustering the found line segments multiple lines can be in view without negatively affecting the tracking.

2. Defining a line

Lets make some assumptions to define a line; a line is bordered by a high derivative stroke on each side, the strokes are assumed to be on approximately the same distance from each other, i.e. the with of the line is approximately constant. The gradients of the two borders have adverse directions, i.e. the borders are parallel to each other and the background on both sides of the line are either brighter or darker than the line itself. A visualization of this can be seen in Figure 1 which shows the gradients in a photography of a line. The gradient on opposite sides of the line can be seen to be at approximately π radians angle in relation to each other. In Figure 2 it can be seen that simply thresholding the gradient magnitude in an image can be enough to find a dark line on a brighter colored background. The same is true for the inverted case where the line has a brighter color than the background. In many cases other objects and shadows can be hard to differentiate from the line using only this technique. Combining this with filtering of stroke width and edge gradient direction can however make the method robust. The stroke width is the distance between the two peaks in the derivative's magnitude. This can be assumed to be approximately the same along the length of the line if the perspective transform is taken into account.

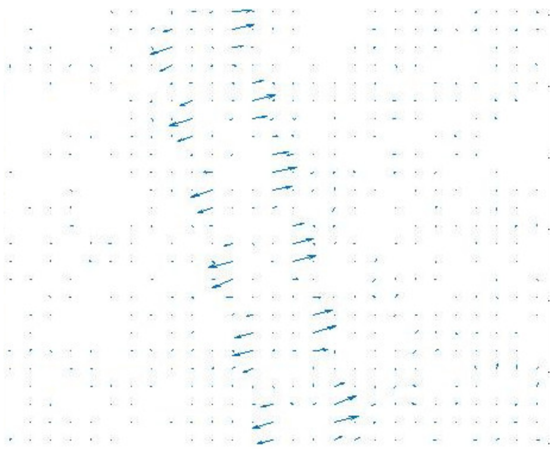


Figure 1. *Visualization of the gradients in a photo of a dark line, spanning over a slightly textured background, clearly shows the constant line width and adverse directions of the bordering gradients*

3. Top view perspective transform

Given two arbitrary planes in $\mathbb{R}^3$ it is possible to calculate a transformation matrix from one plane to the other if enough correspondence is known. The projective transformation consist of a similarity transform, an affinity and a projectivity. The nature of the perspective transform will result in parallel lines mapping to non-parallel lines. To keep the assumption of parallel borders of the line valid it is thus important to conduct a perspective transform to get a top-down view of the lines to track. A transformation from $\mathbb{R}^2$ to $\mathbb{R}^2$ is done by multiplication between the homogeneous coordinate and a 3×3 matrix. The correspondence between the image plane and the ground plane can be found by a

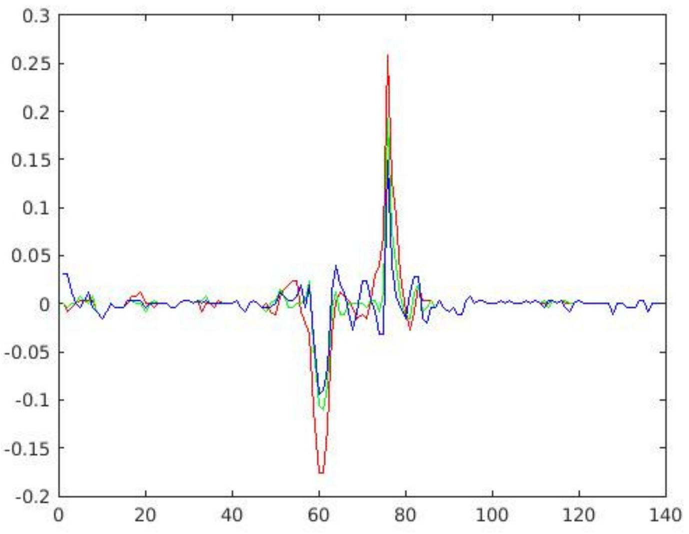


Figure 2. *The derivative of each color channel along an axis perpendicular to a black line in a RGB image*

calibration using a chessboard pattern placed on the ground. The transformation matrix can be found by either a linear approximation using e.g. direct linear transform (DLT) combined with singular value decomposition (SVD) [2] or an iterative approach such as bundle adjustment. The images used need to have an Euclidean coordinate system. Lens distortion can be non linear and thus the coordinate system of an uncalibrated camera is not Euclidean. Especially when using wide angle cameras the image will be distorted in what is sometimes called a barrel shape. The lens distortion has been modeled using three radial and two tangential parameters. Let u' and v' describe the estimated coordinates in the distorted image based on the coordinates u and v in an Euclidean image. Parameters k_1 to k_3 are used to model radial distortion while τ_1 and τ_2 model the tangential distortion. A common model is then described by

$$\begin{aligned} u' &= u*(1+k_1*r^2+k_2*r^4+k_3*r^6)+2\tau_1 uv+\tau_2(r^2+2u^2),\\ v' &= v*(1+k_1*r^2+k_2*r^4+k_3*r^6)+\tau_1(r^2+2v^2)+2\tau_2 uv,\\ r^2 &= u^2+v^2. \end{aligned} \tag{1}$$

To undistort an image the inverse of the lens distortion is needed. It is not suitable to find an algebraic inverse to Equation (1) however making use of Banach fixed-point theorem, [3],[4], an iterative solution can be implemented as following

Algorithm 1 Inverse distortion for an image point $x = (u', v')^T$

```
$u := u'$
$v := v'$
**while** change < converge criteria **do**
    $r_2 := u^2 + v^2$
    $c_{\text{inv}} := 1/(1+((k_3 r_2 + k_2) r_2 + k_1) r_2)$
    $l_{\text{x}} := 2\tau_1 uv + \tau_2(r_2 - 2u^2)$
    $l_{\text{y}} := \tau_1(r_2 + 2v^2) + 2\tau_2 uv$
    $u := c_{\text{inv}} * (u' - l_{\text{x}})$
    $v := c_{\text{inv}} * (v' - l_{\text{y}})$
**return** $u, v$
```

Given a homogeneous coordinate in the camera's Euclidean image plane, $\rho = (u, v, 1)^T$, and a coordinate $\chi = (x, y, 1)^T$ in the ground plane, the relation using a 3×3 homography transformation matrix H can be written

$$\chi \propto H\rho, \tag{2}$$

or by introducing a scalar λ which represent the arbitrary scale factor

$$\lambda\chi = H\rho. \tag{3}$$

Exploiting Equation 3 for every point correspondence in the planes and applying direct linear transform yields an equation of the form $Ah = 0$ where

$$A = \begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & x_1u_1 & x_1v_1 & x_1 \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & y_1u_1 & y_1v_1 & y_1 \\ -u_2 & -v_2 & -1 & 0 & 0 & 0 & x_2u_2 & x_2v_2 & x_2 \\ 0 & 0 & 0 & -u_2 & -v_2 & -1 & y_2u_2 & y_2v_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -u_N & -v_N & -1 & 0 & 0 & 0 & x_Nu_N & x_Nv_N & x_N \\ 0 & 0 & 0 & -u_N & -v_N & -1 & y_Nu_N & y_Nv_N & y_N \end{bmatrix} \tag{4}$$

and

$$h = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix}. \tag{5}$$

By using singular value decomposition the approximative nullspace is given [5] by the eigenvector of A, found in the columns of V, corresponding to the smallest eigenvalue, found in the diagonal elements of Σ

$$A = U\Sigma V^*, \tag{6}$$

where V^* is the conjugate transpose which is equivalent to V^T when A consist of non-complex values. The matrix H is found by reshaping h to a 3×3 format. Given that homogeneous coordinates are up to scale and that H is a 3×3 matrix, H has only eight degrees of freedom. Each point correspondence results in three new equations but also a new unknown scale factor, thus the number of points required to find the homography matrix, with the scale invariant considered, is given by

$$3N - N \geq 9 - 1 \Leftrightarrow N \geq 4. \tag{7}$$

4. Finding the lines

The gradients are calculated in a gray scale image. The RGB images from the camera are recalculated to gray scale using

$$Y = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B. \tag{8}$$

The Sobel operator is used to approximate the derivatives of the gray scale images. The kernels used for the convolutions to get the row- and column-wise derivatives are given by

$$C_{row} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad C_{col} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}. \tag{9}$$

The magnitude of the gradient, $G = (G_{row}, G_{col})^T$, is approximated, to speed up the calculations, using the simplification given by the L_1 norm as

$$G_{\text{mag}} \approx |G_{\text{row}}| + |G_{\text{col}}|. \tag{10}$$

Pixels with a gradient magnitude above a threshold value are in this stage assumed to be pixels on the lines' borders. These are split into two groups: start pixels and end pixels. The pixels in the first group are defined by a negative value in column-wise gradient. For each start pixel an iteration in the image is performed. If the start pixels gradient is given by α then dark lines on brighter background is found by iterating along the direction of $-\alpha$ while brighter lines on darker background are found by iterating, in the opposite direction, along α. The iteration is performed until a (non-zero) end pixel is reached. The moved distance and the angle between the start pixel's gradient and the end pixel's gradient are saved, these are the stroke width l and the stroke border angle difference δ. If the stroke width l is between an upper and a lower threshold and δ is below a maximum angle then all visited pixels defined by the stroke are marked with the minimum of the values corresponding to the stroke length, l, or the value already assigned to the pixel. That is, if I is the set containing all saved line strokes

$$\forall i \in I : t_{l,\text{lower}} < l_i < t_{l,\text{upper}} \quad \text{and} \quad \delta_i < t_{\delta\max}, \tag{11}$$

and L_i is the set of all pixels covered by the stroke i then the new pixel value p_j^{new} is at the assignment of each stroke set to

$$\forall j \in L_i : p_j^{\text{new}} = \min(p_j^{\text{old}}, l_i). \tag{12}$$

If the method is applied to every start pixel in the image an output map is formed where the pixels in the line segments are intensity encoded with their stroke width. Some outliers will be present but the true lines are characterized by their approximately constant stroke width and thereby intensity. For each pixel at $(i,j)^T$ in the output map marked with a stroke width $l_{i,j}$ the number of neighbors with approximately the same value is counted. Only neighbors within a kernel with a specified side length d, satisfying $d \in \{2\mathbb{N}+1\}$, around the pixel are counted and a maximum ratio r between two pixels' values is allowed. Each pixel can thus get a probability score $v_{i,j}$ defined as the normalized number of neighbors fulfilling the ratio test. If P_t is the set of all pixels

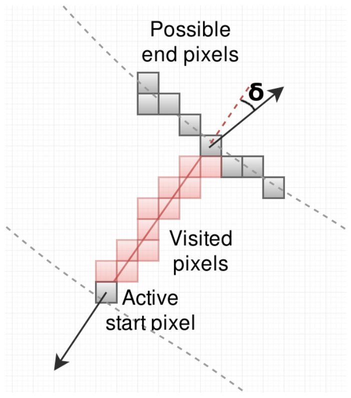


Figure 3. *By following the pixel at a dark line's border in the opposite direction of its gradient the opposite bordering pixel is found, the angle δ can be assumed small if the line's edges are approximately parallel*

fulfilling the ratio criteria and $(k,l)^T$ is all neighboring pixels contained in the kernel around $(i,j)^T$ then

$$\begin{aligned} &\forall l_{i,j} \in P_t : \frac{1}{r} \le \frac{l_{k,l}}{l_{i,j}} \le r, \\ &k \in \{i + \Delta i : |\Delta i| \le \frac{d-1}{2}, i + \Delta i \in \mathbb{N}\}, \\ &l \in \{j + \Delta j : |\Delta j| \le \frac{d-1}{2}, j + \Delta j \in \mathbb{N}\}. \end{aligned} \tag{13}$$

The probability score for each pixel is then defined as the normalized number of elements in P_t

$$v_{i,j} = \frac{\overline{\overline{P_t}}}{d^2}, \tag{14}$$

where the overlines represents the cardinality.

5. Internal representation and clustering

With a top-down view of the detected and vectorized lines in the robot's local coordinate system several views can be stitched together to a complete map in a fixed frame coordinate system. If the distance moved by the robot during one image frame is shorter than the distance covered by the camera, the views of the lines will be overlapping. Many algorithms exist for registration of overlapping point clouds which can be applied. A common approach is Iterative Closest Point [6], [7] where the transformation of two point clouds is calculated by minimizing the sum of the squared distances between points in the two clouds. When several, non connected, lines are seen in the image they need to be differentiated through clustering. In every frame knowledge of which line to follow is required. This is done using a priori knowledge of which line that was followed in the last frame in combination with either the transform achieved from the point registration or from deduced reckoing. A similarity measure between pixels can be calculated in four dimensions where each pixel is described by the vector $q_i = (i, j, l, \gamma)^T$ where i, j is the image coordinates in the top-down view image, l is the stroke width assigned to the pixel and γ is the angle of the stroke covering the pixel. The similarity of two points is defined as the inverse of the Euclidean distance

$$s_{a,b} = \frac{1}{\|(p_a - p_b) \odot (c_1, c_1, c_2, c_3)^T\|_2}, \tag{15}$$

where $\odot$ represents the hadamard product allowing different weight factors compensating for the various units. The lines form long, narrow segments that may partly encircle each other, thus center-based algorithms such as k-means [8] are not suitable. A density based algorithm, such as DBSCAN [9], is instead proposed. DBSCAN will not cluster all elements but will instead filter low density areas as outliers, a functionality which is appreciated in the line segment detection context. With the line segments divided into clusters morphological filters can be applied. In this paper only one such filter was implemented, a length to width ratio. The distance between the two points furthest away from each other within the cluster was compared to the mean stroke width of the same cluster.

6. Trajectory planning

The stroke width detection algorithm offers several advantages when the trajectory planning is to be implemented. In every point of the detected line the line's normal is known and can be used to optimize the trajectory. Using the differences between the normals at two or more points the curvature of the path is obtained. This curvature is an important feature when the speed of travel is decided.

7. Experimental results

The algorithm has been implemented using a Python and Cython combination for evaluation of performance and execution speed. The results show good performance and are presented in Section 7.1 and 7.2.

7.1 Detection of lines

The detection of lines has been tested thoroughly on several surfaces ranging from asphalt and concrete grounds to wooden floors. Results are seen in Figures 4-6. For each tested surface six images are presented. Top left images show the raw frames from the onboard camera. Top right are the top-down views achieved by remapping the pixels using the homography transform. Middle left show the detected start pixels as green markers and stop pixels as red markers. Middle right show an intensity encoded map representing the stroke width of each pixel. Left bottom represent the probability score in a color coded map. Right bottom images mark all detected pixels in a line segment using color encoding for different segments. The numbering of clusters in the right bottom images in Figure 5 and 6 indicate that smaller clusters, not fulfilling the length:width ratio, have been dropped in the morphological filter stage.

7.2 Real time performance

Real time performance is crutial in high speed line following. The number of strokes has been vastly reduced by defining start and

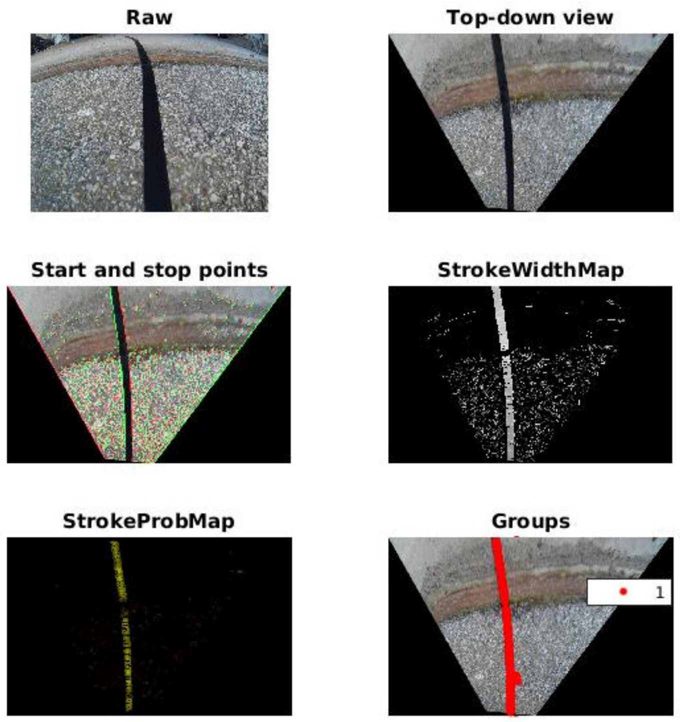


Figure 4. *Detection of a dark line on a mixed concrete-asphalt ground*

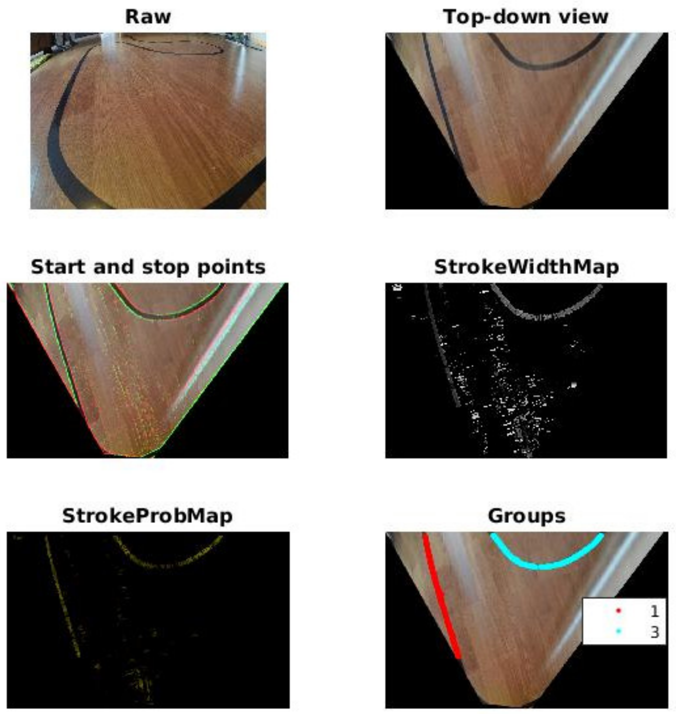


Figure 5. *Detection of two dark line segments on a wooden floor*

end pixels as a highly limited subset of all the images' pixels, as seen in middle left images in Figure 4-6. Stroke detection and creation of probability map is capable of running on limited hardware (Intel Core i5 M460 @2.53GHz using a single core)

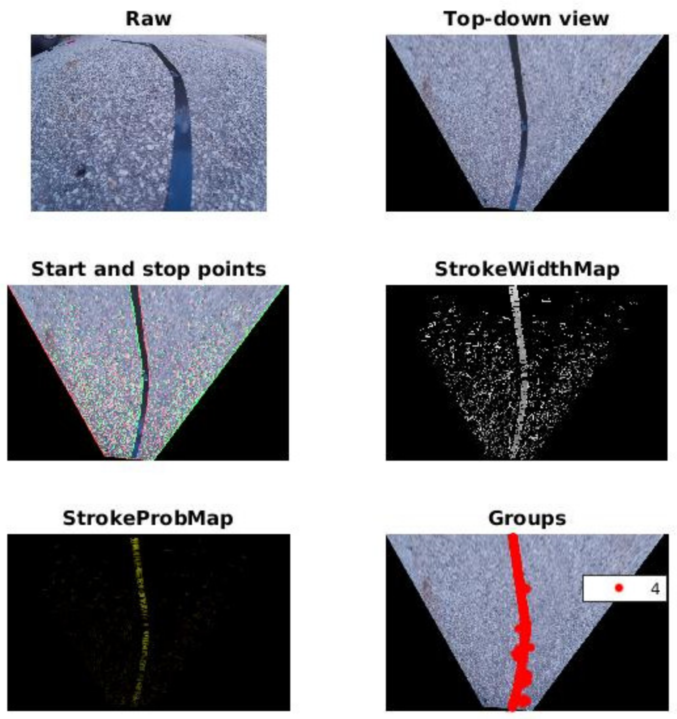


Figure 6. *Detection of a dark line on asphalt ground*

at full camera frame rate, 30fps, in high resolution mode up to 700 by 500 pixels. Clustering using DBSCAN has however introduced stricter limitations of frame rate or resolution.

8. Conclusion

The algorithm developed shows good performance with high accuracy and potential for high speed navigation. Diverse background surfaces for the line following have been tested with good results. Fast navigation that demands full camera frame rate must be limited to a lower resolution than the 700 by 500 pixels used in the experiment or a more time efficient clustering algorithm must be utilized. The clustering method's impact on the frame rate is dependent on the total number of pixels marked as line segments in the image. Lower image resolutions do not pose a problem for line following although the total information is reduced.

References

[1] EPSHTEIN, Boris; OFEK, Eyal; WEXLER, Yonatan. Detecting text in natural scenes with stroke width transform. *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE*, 2010, p. 2963-2970.

[2] HARTLEY, Richard; ZISSERMAN, Andrew. Multiple view geometry in computer vision. *Cambridge university press*, 2003.

[3] PALAIS, Richard S. A simple proof of the Banach contraction principle. *Journal of Fixed Point Theory and Applications*, 2007, 2.2: 221-223.

[4] BANACH, Stefan. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. Math*, 1922, 3.1: 133-181.

[5] LAWSON, Charles L.; HANSON, Richard J. Solving least squares problems. *Siam*, 1995.

[6] BESL, Paul J.; MCKAY, Neil D. Method for registration of 3-D shapes. *Robotics-DL tentative. International Society for Optics and Photonics*, 1992. p. 586-606.

[7] ZHANG, Zhengyou. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 1994, 13.2: 119-152.

[8] PANG-NING, Tan, et al. Introduction to data mining. *Library of congress*. 2006.

[9] VERMA, Manish, et al. A comparative study of various clustering algorithms in data mining. *International Journal of Engineering Research and Applications (IJERA)*, 2012, 2.3: 1379-1384.