NTNU – Trondheim
Norwegian University of
Science and Technology

Visual Sensing in Mobile Robots

Vegard S. Lindrup

Submission date: December 2015
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Title: Visual Sensing in Mobile Robots

Student: Vegard S. Lindrup

Problem description

Introduction

This project is a continuation of previous projects in developing a concept for robotic maintenance performed by a mobile autonomous robot. Over the course of previous projects, the system has been equipped with a robot manipulator arm, several sensors, a wireless router, on-board power supply based on batteries and a central PC. This equipment is mounted on a steel wagon. The wagon stands on four omni-wheels, each with their own electric motor drivers.

Project Goals

To increase the robot‘s degree of autonomy, it is desired to explore possibilities for reliable and safe movement of the robot without involving a person. The purpose of such a movement may be to reach a docking station or a location where a maintenance or an inspection task will be performed while avoiding obstructions and hazardous situations. As a step towards achieving these goals, the following points shall be carried out:

1. Explore potential methods for autonomous navigation that may fulfill the goals above, where computer vision is the primary navigational aid.
2. Implement one, several or a combination of the methods found in point one. This includes selection and installation of new equipment, e.g. cameras, if necessary.
3. Performance and suitability assessment of the selected implementation with respect to autonomous navigation.
4. Assess how well the system handles errors and potentially hazardous states and situations.
5. Propose changes to the implementation and suggest further work in order to improve the safety and reliability of the system and its ability to navigate autonomously.

Supervisor: Tor Engebret Onshus, ITK

Abstract

Preface

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

List of Figures

List of Tables

Contents

Chapter 1
Introduction

1.1 Mobile Autonomous Robotics and Computer Vision

The field of computer vision has seen an enormous growth over the last few decades - not only in scale, but in accessibility and capability as well. As a consequence of this recent growth, tapping into the field of computer vision is bound to reveal applications that are useful for a mobile autonomous maintenance robot. Recent discoveries within computer vision includes robust feature recognition and object detection, face detection and video processing. The latest great additions to the field are Big Data and Artificial Intelligence.

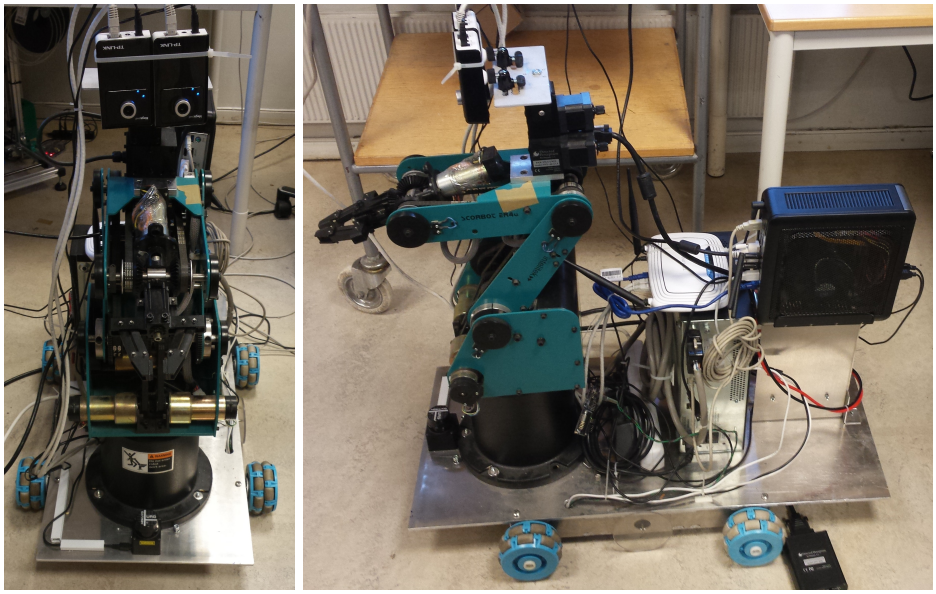
1.2 System Overview

The mobile robot being worked on in this project is shown in figure 1.1a. The manipulator arm has been used in previous projects on robotic maintenance, and it was placed on the mobile platform during the master thesis of Petter Aspunvik. This section provides a short description of the systems used in this project and other surrounding equipment. If a more detailed description of the robot and it's equipment is required, consult the thesis of Aspunvik [Asp13].

1.2.1 The Robot

The robot in its current state is the result of several preceeding projects, where the master thesis of Petter Aspunvik [Asp13] and Mikael Berg [Ber13] are the most recent contributions.

Propulsion The steel chassis of the robot stands upon four omni-wheels. The wheel pairs are placed in parallel, making the vehicle uncontrollable along the lateral axis. Each wheel is powered by an electrical motor and motor driver. The motor drivers are controlled with pulse width modulation by an evaluation board from Atmel (Xmega A3BU).



(a) Front view. (b) Side view.

Figure 1.1: The robot used in the project.

Sensors The robot has been outfitted with several sensors over the course of previous projects. These are:

- Two odometer wheels with encoders. One on each side.
- Two infrared distance sensors.
- A LIDAR (Light Detection and Ranging).
- Two IP-cameras.

Only the cameras were used in this projects.

Robot Arm A robot arm, SCORBOT-ER 4u from Intelitek, is monted on the wagon. It is intended for educational use in the context of automation and work cells. The robot has five rotation joints plus a servo gripper at the outer joint. Control of the robot in this project is done from the on-board computer on the wagon, which is connected to the robot arm through a USB cable.

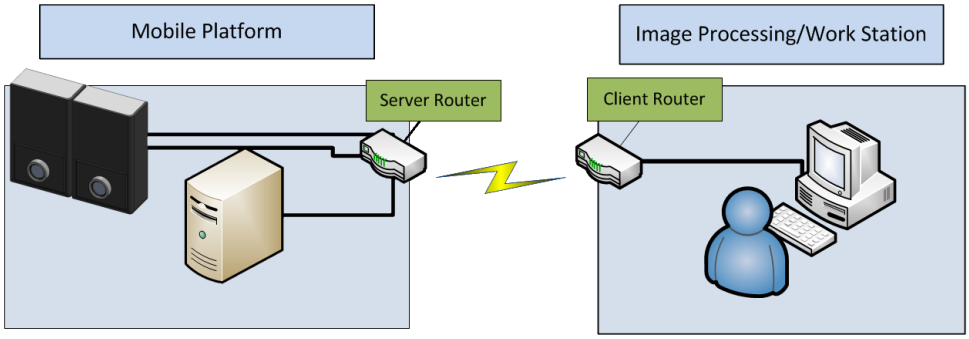


Figure 1.2: Stereo vision set-up.

1.2.2 Project Set-up

Vanishing points and stereo vision are the main topics in this report. The stereo vision portion consist of the two IP cameras, a pair of wireless routers and a remote computer. The remote computer can be any computer which is able to run Open Source Computer Vision Library (OpenCV) and receive video feed from the camera pair. The vanishing point detector is simply a desktop application which accesses a web camera. Both the stereo vision and vanishing point detector applications have been developed with Qt and run on Windows 7.

1.3 Report Structure

How the report is structured, and a very brief description of the contents in each section.

Chapter 2
Background Theory

2.1 Introduction

2.1.1 Chapter Scope

This chapter contains the background theory which is necessary to understand the implementations in chapter 4 and how they are intended to work.

2.1.2 Brief Introduction to Computer Vision

Computer Vision is the field of giving computers the ability to duplicate the way we perceive and understand our surroundings. This is not an easy task for several reasons. Much information is lost in the process of representing a 3d world on a 2d surface. As image processing really is digital signal processing in 2d, it is also subject information loss and noise through quantization. In short, the field of computer vision can be summed up to be the science of drawing conclusions about the real world based on pixel values in an image matrix, or a series of such matrices.

2.1.3 Introduction to OpenCV

OpenCV is an open source library with a vast number of advanced computer vision and machine learning algorithms. The library supports Windows, Linux, iOS and Android, and has interfaces to C, C++, Python, Java and MATLAB. All OpenCV applications in this project uses OpenCV 3.0.0 for Windows. OpenCV for Windows can be downloaded from sourceforge.net. This download contains source files, sample programs, sample data and a pre-built library for MSVC 2010 and 2013. The pre-build library can quickly be plugged into an IDE such as Qt Creator or Visual Studio 2013, thus giving the programmer access to all basic OpenCV features. A steb-by-step guide for using both the pre-built and a custom-built library can be found in Appendix A.

cv::Mat - The Image Container

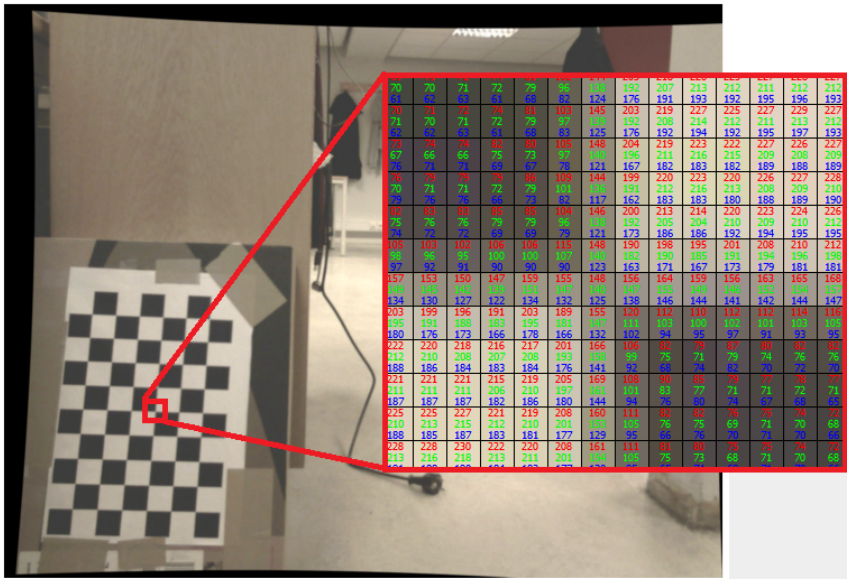


Figure 2.1: Elements in a cv::Mat representing an image. This is a 3-channel image with 8 bit RGB colors.

2.2 A Brief Introduction to Vision Based Autonomous Navigation

2.2.1 Simultanious Localization and Mapping (SLAM)

Consider a robot that is placed in an unknown environment with no a priori information of the surroundings. If this robot is capable of solving the Simultanious Localization and Mapping (SLAM) problem, it is implied that it can build a map of the surroundings while it determines, simultaniously, where it is located on the same map. This section fill, very briefly, present some solutions to the SLAM problem. From [DWB06] a preliminary description of the SLAM problem is based on the following parameters at time k is:

$\boldsymbol{x_k}$: "State vection describing position and orientation of the vehicle"[DWB06].

$\boldsymbol{u_k}$: Control input at time $k-1$. Used to drive the vehicle to x_k.

$\boldsymbol{m_i}$: The position of the *ith* landmark. The true position is assumed to be time invariant.

$\boldsymbol{z_{ik}}$: Observed position of the *ith* landmark relative to the vehicle at time k.

$\boldsymbol{X_{0:k}} = \{\boldsymbol{x_0}, \boldsymbol{x_1}, ..., \boldsymbol{x_k}\} = \{\boldsymbol{X_{0:k-1}}, \boldsymbol{x_l}\}$ A set of all previous vehicle locations.

$\boldsymbol{X_{0:k} = \{x_0, x_1, ..., x_k\} = \{X_{0:k-1}, x_l\}}$ All previous control inputs.

$\boldsymbol{X_{0:k} = \{x_0, x_1, ..., x_k\} = \{X_{0:k-1}, x_l\}}$ All previous landmark observations.

Probabilistic SLAM Probabilistic SLAM computes the probability distrobution $P(x_k, m|Z_{0:k}, U_{0_k}, x_0)$ at all time steps k[DWB06], i.e. the joint probability that the robot is in state $\boldsymbol{x_k}$ and that the landmarks can be described by a map,$\boldsymbol{m}$, given historic inputs, landmark observations and vehicle states.

EKF-SLAM

MonoSLAM MonoSLAM is the first successful vision based real-time SLAM algorithm[mon07]. The algorithm works by maintaining a probabilistic 3d map of a sparse set of landmarks. The map is continously updated by an Extended Kalman Filter (EKF). Landmarks are chosen based on som prominent features in the surroundings. In the algorithm, they are represented as flat surfaces with a certain orientation in 3d space. This solution will approximate the fact that the appearence of a feature will change based on the position from where they are observed.

The Kinect Sensor and SLAM The Kinect sensor has been a remarkable contribution to the field of SLAM and computer vision. It is a cheap sensor capable of generating high quality, but noisy, depth maps at $\boldsymbol{30Hz}$. By applying SLAM to the real-time depth data, it is possible to generate a very accurate 3d reconstruction of the surroundings.

2.3 The Pinhole Camera Model

2.3.1 Model Description

Figure 2.2 illustrates the geometry of a pinhole camera. In a pinhole camera, light reflected from some object in a scene will be projected onto an image plane inside the camera house. The light is projected through the "pinhole", that is the camera projection point $\boldsymbol{O}$, which is located at the origin. Imagine that a single ray enters the camera house

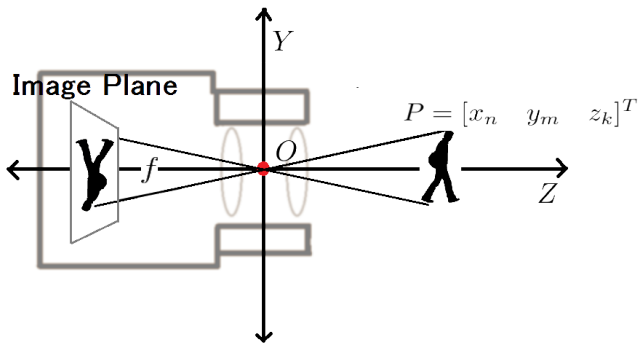


Figure 2.2: Geometry of a pinhole camera.

through the pinhole, before
it hits and exites a sensor element on the image plane. The values retreived from these sensor elements will make up the pixels in an image. The focal length $\boldsymbol{f}$ is the distance from the projection point $\boldsymbol{O}$ to the image plane $\boldsymbol{\pi}$. In a true pinhole camera the image plane will be located behind the lens and the projection point $\boldsymbol{O}$, and the scene projection will be rotated by $\mathbf{180°}$. A virtual image plane placed in front of $\boldsymbol{O}$ is intended to make the figure more straightforward and the mathmatics easier.

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{f_x} & \mathbf{0} & \boldsymbol{c_x} \\ \mathbf{0} & \boldsymbol{f_y} & \boldsymbol{c_y} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{2.1}$$

"Learning OpenCV" by Gary Bradski[Bra08] shows how a pinhole camera can be modelled by a 3 by 3 camera matrix $\boldsymbol{M}$ (equation 2.1). This model accounts for two important differences between the ideal and real pinhole camera. First, the imaging chip will often be displaced from the optical center. This displacement is described by the parameters $\boldsymbol{c_x}$ and $\boldsymbol{c_y}$. The second real world problem is that the pixels on the image sensor are shaped as rectangles, not squares. This is accounted for by $\boldsymbol{f_x}$ and $\boldsymbol{f_y}$, the focal lengths in the $\boldsymbol{x}$ and $\boldsymbol{y}$ direction given in pixels. These values are the product of the actual focal length $\boldsymbol{f}$ and size of the individual imager elements $\boldsymbol{s_x}$ and $\boldsymbol{s_y}$. The reason for using these values in the camera model is because $\boldsymbol{s}$ and $\boldsymbol{f}$ cannot be measured without actually dismantling the camera[Bra08].

2.3.2 Camera Distortion

A downside of the otherwise cheap and useful pinhole camera is camera distortion. The distortion is usually severe enough to render the camera useless as a sensor if it is not calibrated. Calibration in OpenCV accounts for radial and tangential distortion by finding five distortion coefficients [3dC]:

$$\boldsymbol{Distortion_{coefficients}} = (\boldsymbol{k_1} \quad \boldsymbol{k_2} \quad \boldsymbol{p_1} \quad \boldsymbol{p_2} \quad \boldsymbol{k_3})$$

where

$$\boldsymbol{p_i} \quad \boldsymbol{with} \quad \boldsymbol{i} \in \{\mathbf{1}, \mathbf{2}\}$$

are the radial distortion constants, and

$$\boldsymbol{k_i} \quad \text{with} \quad \boldsymbol{i} \in \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$$

are the tangential distortion constants.

2.4 Projective Geometry and Vanishing Points

2.5 Stereo Vision and Depth Perception

Stereo vision and depth perception is one of the core topics within this report. Here, the theory behind a method using two cameras is presented, while some additional methods are mentioned to provide context.

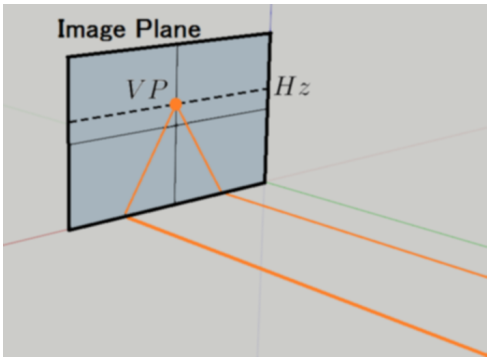


Figure 2.3: Two parallel lines are projected onto an image plane, where they form two lines. These projected lines converge towards a vanishing point on the horizon.

2.5.1 Various Methods

Methods for depth perception in computer vision can be separated into two main categories, active and passive[XWS06]. Active sensors will usually project a light pattern onto the scene to be perceived, before sensing how this pattern is displaced by the topology of the scene. The Kinect sensor and 3d-scanners using laser light are typical examples of active sensors. Passive depth perception makes use of many of the same cues we use to perceive depth. The most common passive sensors extract the depth information by observing observing a scene from at least two different positions.

Optical flow is another important method for depth perception. Optical flow may be either active or passive. The passive variant requires only one camera, but depends on motion and a stream of images to extract depth information. Observing how much some chosen features in a scene has moved in the image frame at $\boldsymbol{t = 1}$ compared to the frame at $\boldsymbol{t = 0}$ is the basis of depth sensing from optical flow. When the camera moves through a scene where all objects are stationary, objects that are far away will naturally have an optical flow field with a smaller magnitude than objects that are close.

2.5.2 Stereoscopic Vision in General

In this project, passive stereoscopic vision is achieved by using two identical (in theory) cameras placed on the same plane. The gist of passive stereoscopic vision is based on the fact that objects close to the camera pair will have a large displacement from the left to the right camera compared to objects that are further away. This concept is illustrated in figure 2.4.

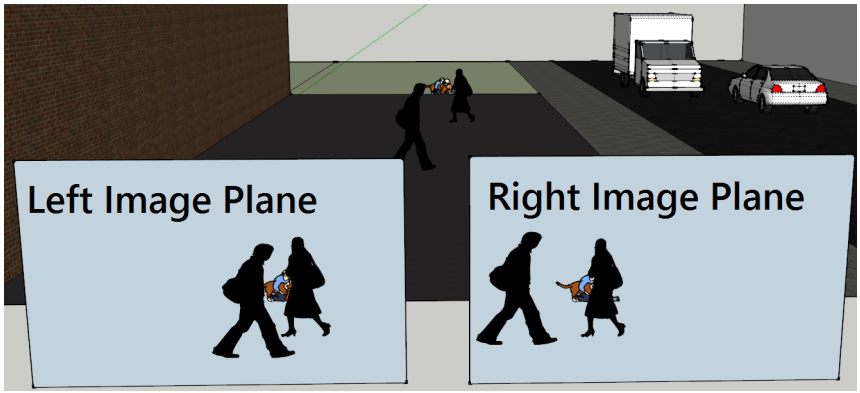


Figure 2.4: Left to right displacement on the image plane based on distance.

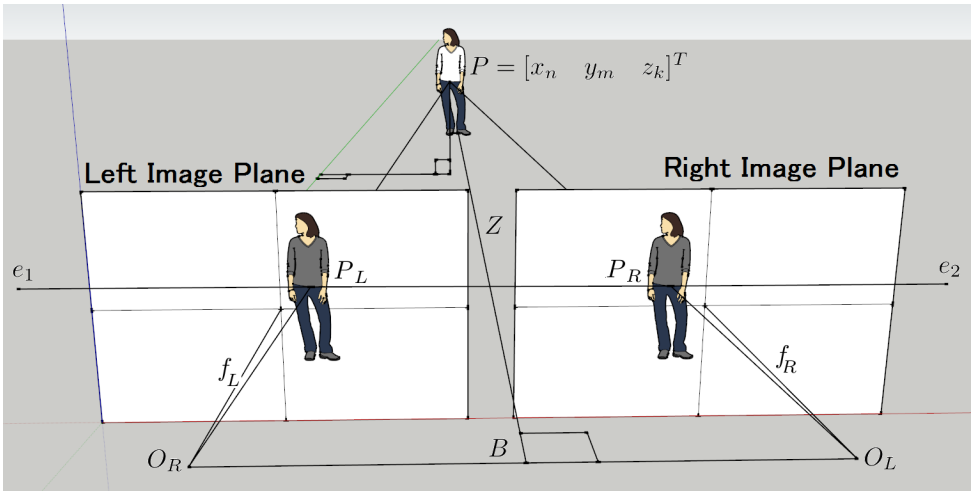


Figure 2.5: Geometry of rectified stereo vision.

Stereo Cameras Figure 2.5 shows an ideal stereo camera model. The model comprise two pinhole camera models where the virtual image planes are located on the same plane. The two image planes are separated by a horizontal translation $\boldsymbol{B}$ which is called the baseline. This implies that the projection point $\boldsymbol{O_L}$ in the left camera, relates to the projection point $\boldsymbol{O_R}$ on the right camera through $\boldsymbol{B}$: $\boldsymbol{O_R = O_L + B}$. Each of the two image planes has a left handed pixel based coordinate system $\boldsymbol{u, v}$, i.e. the origin is in the top left corner and the opposite pixel is in the bottom right corner.

2.5.3 Stereoscopic Vision in OpenCV

The prebuilt version of OpenCV 3.0.0 comes with two stereo matching algorithms: Block Matching (BM) Block Matching (StereoBM) and Semi Global Block Matching (StereoSGBM). Additional algorithms are available if OpenCV is built with, e.g. CUDA.

StereoSGBM [Hir08] blablabalb BM.

Chapter 3
Example

Here is an example of how to use acronyms such as Norwegian University of Science and Technology (NTNU). The second time only NTNU is shown and if there were several you would write NTNUs. And here is an example[1] of citation [NN00].

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

And after the second paragraph follows the third paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original

[1] A footnote

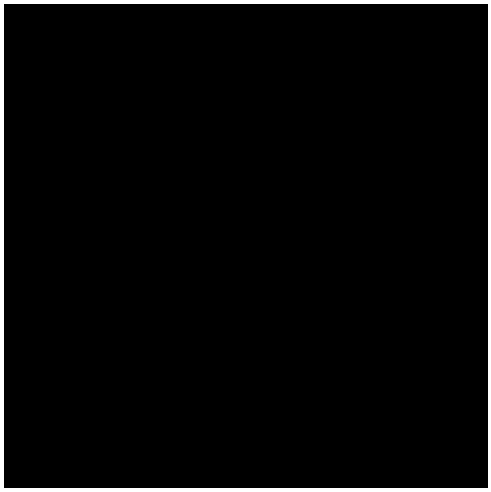


Figure 3.1: A figure

language. There is no need for special content, but the length of words should match the language.

3.1 First section

3.1.1 First subsection with some $\mathcal{M}ath$ symbol

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

- item1
- item2
- ...

3.1.2 Mathematics

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. $\mathbf{\sin^2(\alpha) + \cos^2(\beta) = 1}$. If you read this text, you will get no information $\boldsymbol{E = mc^2}$. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift –

Table 3.1: A table

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p	q	r	s	t
u	v	w	x	y
z	æ	ø	å	

not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sqrt[n]{\boldsymbol{a}} \cdot \sqrt[n]{\boldsymbol{b}} = \sqrt[n]{\boldsymbol{ab}}$. This text should contain all letters of the alphabet and it should be written in of the original language. $\frac{\sqrt[n]{\boldsymbol{a}}}{\sqrt[n]{\boldsymbol{b}}} = \sqrt[n]{\frac{\boldsymbol{a}}{\boldsymbol{b}}}$. There is no need for special content, but the length of words should match the language. $\boldsymbol{a}\sqrt[n]{\boldsymbol{b}} = \sqrt[n]{\boldsymbol{a^n b}}$.

$\boldsymbol{B}$
$\boldsymbol{X_L}$
$\boldsymbol{X_R}$
$\boldsymbol{P_R}` \boldsymbol{P_L}$
$\boldsymbol{P} = [\boldsymbol{X} \quad \boldsymbol{Y} \quad \boldsymbol{Z}]^{\boldsymbol{T}}$
$\boldsymbol{P} = [\boldsymbol{x_n} \quad \boldsymbol{y_m} \quad \boldsymbol{z_k}]^{\boldsymbol{T}}$
$\boldsymbol{e_1}$
$\boldsymbol{e_2}$
$\boldsymbol{O_R}$
$\boldsymbol{O_L}$
$\boldsymbol{X} = [\boldsymbol{x} \quad \boldsymbol{y} \quad \boldsymbol{z}]^{\boldsymbol{T}}$
$\boldsymbol{f}$
$\boldsymbol{Z}$
$\boldsymbol{Y}$
$\boldsymbol{VP}$
$\boldsymbol{Hz}$

Proposition 3.1. *A proposition... (similar environments include: theorem, corrolary, conjecture, lemma)*

Proof. And its proof. □

3.1.3 Source code example

You can refer to figures using the predefined command like Figure 3.1, to pages like page 14, to tables like Table 3.1, to chapters like Chapter 3 and to sections like

Algorithm 3.1 The Hello World! program in Java.

```
class HelloWorldApp {
  public static void main(String[] args) {
    //Display the string
    System.out.println("Hello World!");
  }
}
```

Section 3.1 and you may define similar commands to refer to proposition, algorithms etc.

Chapter 4
Implementation

4.1 Introduction

An obstruction detector and a vanishing point detector are the two attempted implementations presented in this report. The obstruction detector uses stereo vision to perceive depth and distance to possible objects in the path of the robot. The vanishing point detector attempts to find a single vanishing point by detecting lines in the environment before selecting a vanishing point based on line intersections.

4.2 Vanishing Point Detection

4.2.1 Overview

The goal of the vanishing point detector is to provide a setpoint for the robot to steer towards. In other words, steering towards a vanishing point is a good way to reach the end of a hallway or corridor. This was considered to be a good starting point, and possible expantions could be added later. Choosing a method as a basis for a vanishing point detector was not easy. The selected method should be simple, suitable for OpenCV and not go too far beyond the prior knowledge of the author. Another important factor was that spending too much time on this implementation would come at the expense of the obstruction detector. A vanishing point detector method by D. Gerogiannis et. al. [GNL12] showed promise as it was based on line detection, which has good support in OpenCV. The method in [GNL12] appears to be suitable for structured environments with many straight lines, such as hallways, streets and corridors. The steps in the detection procedure are:

1. Detect edges in the image, e.g. by using Canny edge detection.

2. Detect line segments that may be used as vanishing lines based on edges found in step 1. Could be done with the Hough line transform.

3. Filter the detected lines. This is done by modelling new lines by using the major axis of ellipses with very high eccentricity. The ellipses are generated by a split-and-merge algorithm. In short, it will merge similar line segments by assuming that their end points are collinear.
4. Find line intersection points based on the new filtered lines. Each point is stored and assigned a weight.
5. Find the vanishing point among the line intersections based on a voting scheme.

4.2.2 Line Detection

Line detection comprise step 1 and 2 from the list above. OpenCV comes with an implementation of the Canny edge detector ready for use. The detector returns a binary image of the detected edges. Edges are detected by convolving the input image with two kernels $\boldsymbol{G_x}$ and $\boldsymbol{G_y}$. The convolutions will indicate change gradients in the $\boldsymbol{x}$ and $\boldsymbol{y}$ directions which in turn will give the direction of a potential edge. Finally, the detector rejects or accepts potential edges based on two gradient thresholds. Gradients below the lower threshold are rejected, edges above the upper threshold are accepted, and edges between the thresholds are only accepted if their neighbouring gradients are above the upper threshold[can].

At this point, we only have a simple binary image where edges are represented as white pixels on a black background. The next step is to interpret these edges as lines. Line detection is performed by using the probabilistic Hough line transform. This is an already implemented function, which takes in the edge image from the previous step, and return a set of point pairs representing line segments. The benefit of using the probabilistic detector is that it can perceive an edge with a discontinuity as a singe line.

4.2.3 Line Filtering

Line filtering is performed by splitting and merging ellipses until their major axis represents a set of approximations to collinear points.

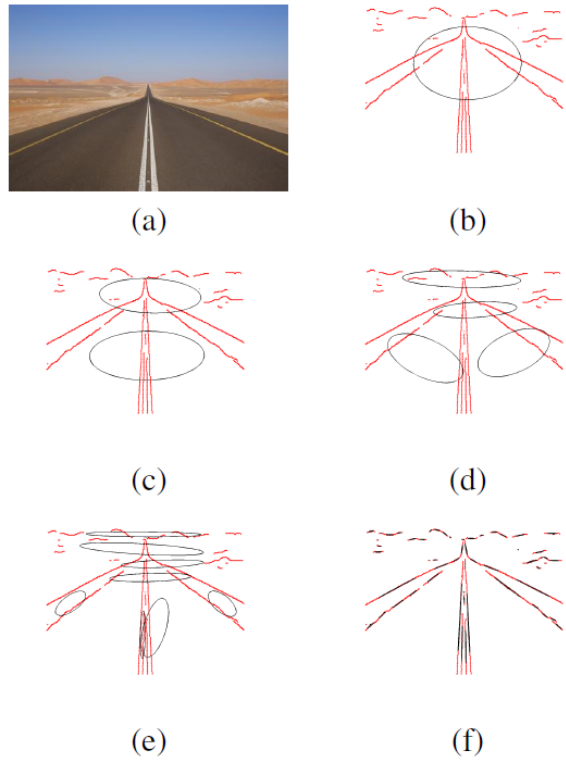


Figure 4.1: Gerogiannis concept of representing lines by eccentric ellipses. This image is taken directly from [GNL12].

The points is the set of points that define the previously detected line segments. This algorithm is called Direct Split and Merge (DSaM), and it is explained in another paper by D. Gerogiannis [GNL11]. Line segments returned from the probabilistic Hough transform will often overlap or be very close to each other. The purpose of this step is to get a cleaner representation of the contours in the environment.

Figure 4.1, taken from [GNL12], illustrates the steps in the DSaM algorithm.

4.2.4 Vanishing Point Detection

When the detected lines have been filtered and stored, they will be passed to the vanishing point detector in the function "getVanishingPoint(lines)". This function will perform two steps (figure **??**):

1. Find, store and assign weights to the points where the lines intersect. Lines that are either too vertical or too horizintal will not be included in the calculations. Intersectionpoints outside the image frame are rejected.
2. Find the vanishing point based on the valid weighted intersection points. This is done by a voting scheme described in [GNL12], and illustrated as a flowchart in figure 4.3.

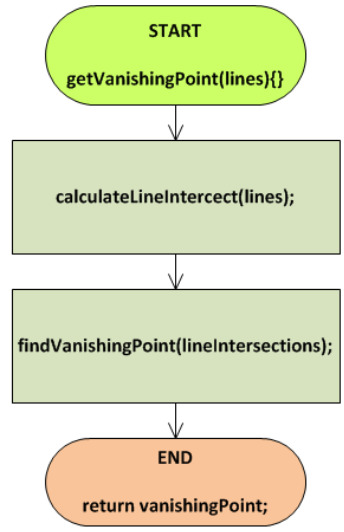


Figure 4.2: Two steps in "getVanishingPoint()".

4.2.5 Vanishing Point Detector Application

Program Structure The vanishing point detector was implemented as a QWidget Application in the Qt Creator IDE. The code excerpt shown in algorithm 4.1 contains the most important image processing steps. A main thread, which may be called the GUI thread, handles all user related input and output. All image processing is done in the class "ImageProcessing" which inherits from QThread. This means that "ImageProcessing" controls a protected function "run()" which contains the threaded code and image processing steps. Figure 4.4 is a sequence diagram that shows the different classes and threads interact. The ellipse filter step is ommitted in the illustration; if it had been included, it would be called between "hough->detect()" and "getVanishingPoint()".

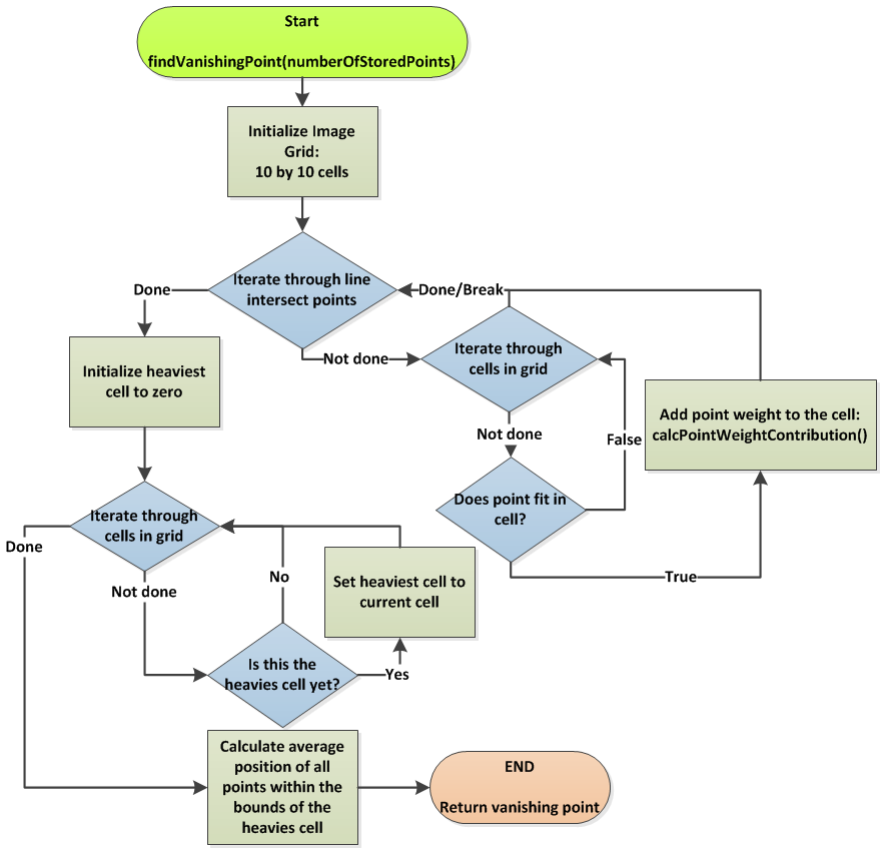


Figure 4.3: Sequence diagram illustrating program execution when the user activates camera feed and VP detection.

Algorithm 4.1 Vanishing point detector loop. Several lines of code are omitted in this example to make the processing more clear.

```
while(){
capture.read(cameraImg);
cvtColor(cameraImg,grayImg,CV_RGB2GRAY);
blur( grayImg, blurredImg, Size(3,3) );
Canny(blurredImg, edgesImg, lowerThresh, upperThresh, 3);
gpu_edgesImg.upload(edgesImg);
lines = detectHoughLines(gpu_edgesImg);
newLines = mLineEllipseFilter.filterLines(lines,originalImage);
Point vanishingPoint = vpDetector.getVp(newLines,cameraImg);
}
```

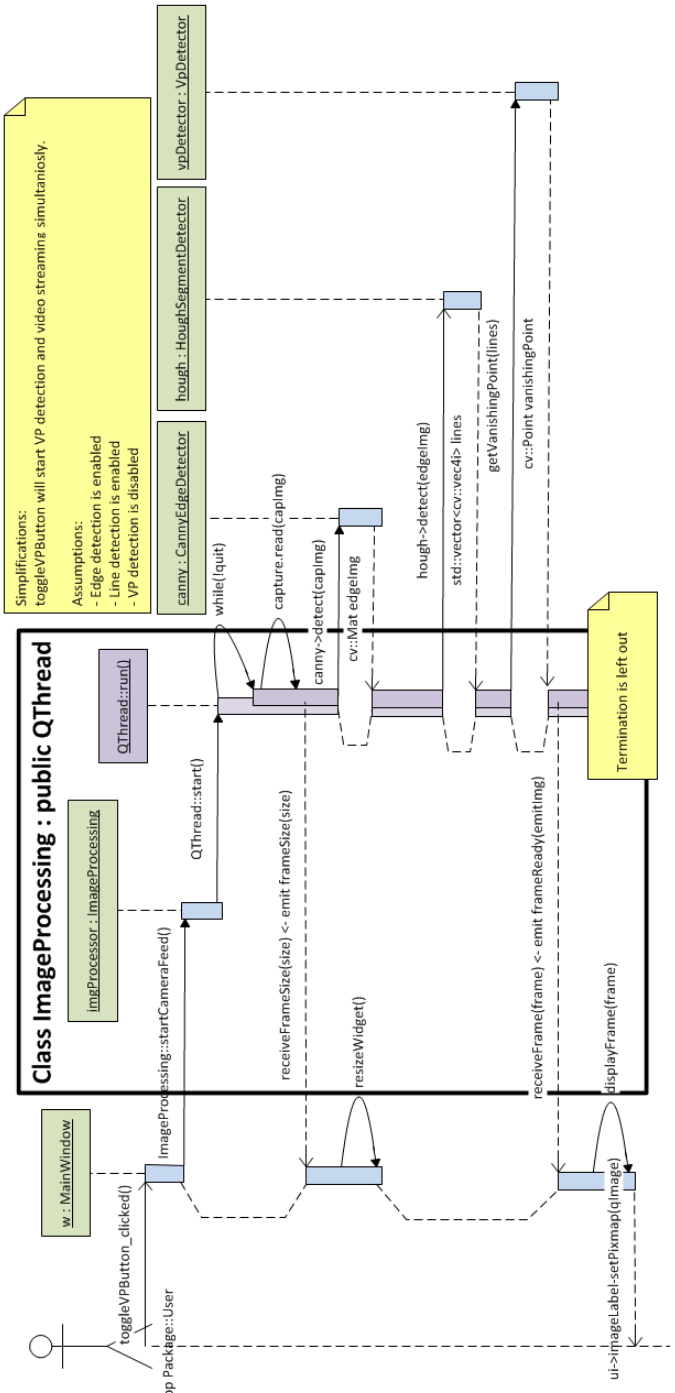


Figure 4.4: Sequence diagram illustrating program execution when the user activates camera feed and VP detection.

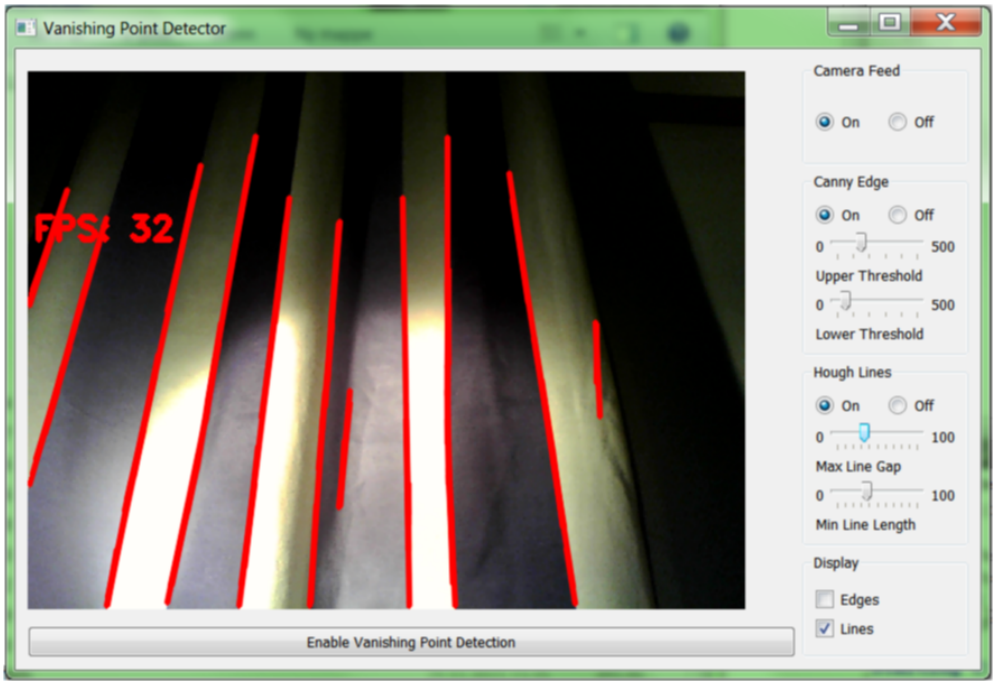
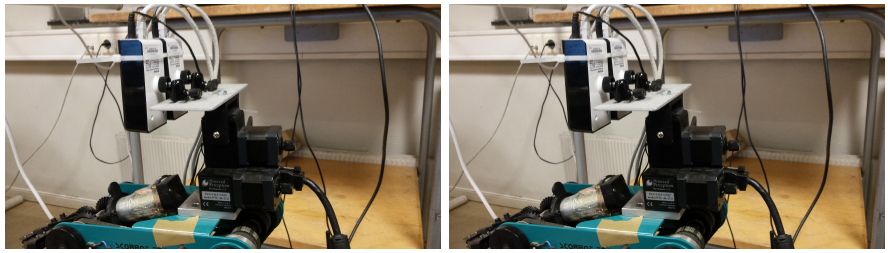


Figure 4.5: Graphical user interface for the vanishing point detector. Note that the detected line segments are actually several lines overlapping each other.

Graphical User Interface A graphical user interface was created so that the parameters for the canny edge detector and Hough lines detector could be tuned on-line. All widgets shown in figure 4.5 have their functionality implemented. The user can turn on the camera feed, in this case from the web camera integrated into the laptop of the author, and switch line and edge detection on or off. Upper and lower edge detection thresholds, as well as line detection parameters can be set by using the sliders. The kernel size for the edge detector is set to 3 by 3, and can not be changed by the user. When both edge detection and line detection is enabled, the user may turn on the vanishing point detector module. In this particular application, the ellipse line filter module is not included.



(a) Camera pair mounted on the pan-tilt module.

(b) Camera pair mounted on the pan-tilt module.

Figure 4.6: The two camera positions.

4.2.6 Cause of Failure

4.3 Depth Perception and Obstruction Detection

4.3.1 Overview

4.3.2 The camera rig

The two IP cameras were moved together to form a stereo camera. This stereo camera was used in two positions. The first camera position is on the pan-tilt module on the robot arm, see figure 4.6a. The second position is just over the LIDAR in front of the robot arm base, see figure 4.6b. The workshop at Institutt for teknisk kybernetikk (ITK) made a mounting bracket, so that the cameras could be placed over the LIDAR. In stereo vision, it is essential that the positions of the cameras relative to each other is constant. One problem encountered throughout the project was that the camera assemby, when placed either at the pan-tilt module and over the LIDAR, was not rigid enough. The severity of this problem was somewhat alleviated by wrapping a strap around both the cameras. This camera rig is ad hoc, i.e. suitable for the purpose of this project, and a better solution should be used for succeeding projects.

4.3.3 Graphical User Interface

Tuning the parameters for stereo matching in OpenCV is a wearisome task, especially without a good graphical user interface. Figure 4.7 shows the user interface which was used to observe how parameter tuning alters the disparity map quality. Not all functionalities were implemented.

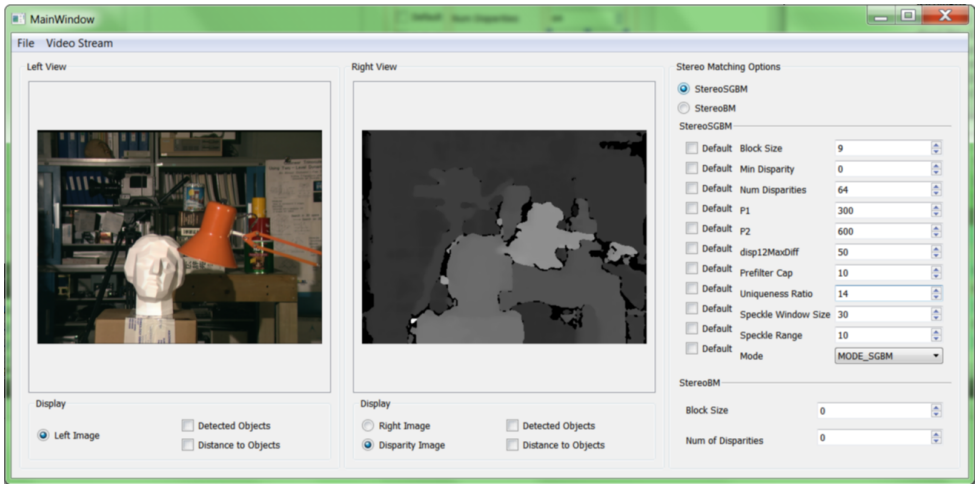


Figure 4.7: Graphical user interface for stereo matching. A disparity map is computed from the Tsukuba samples by using StereoSGBM.

4.3.4 Calibration

As mentioned in chapter 2, all cameras will have some distortion. If the distortion is too severe, as it often will be in the context of stereo vision, the camera must be calibrated. In addition, it was assumed that the image planes were located on the same plane, and that a projection pair, for example the projections $\boldsymbol{X_L}$ and $\boldsymbol{X_R}$ of an object $\boldsymbol{X}$, form two equal epipolar lines, $\boldsymbol{e_1}$ and $\boldsymbol{e_2}$, on the two image planes. In practice, these conditions are achieved through stereo calibration. The second purpose of the calibration procedure is to relate the sensor data to real world quantities, in order to measure the distance to detected objects. Code listings from Practical OpenCV by Samarth Brahmbhatt [Bra13] has been used as a basis for calibration in this project. Some parts of his code is almost unchanged, while other parts of the listings are altered and expanded significantly. There are three steps in the calibration procedure:

1. Single camera calibration.
2. Stereo calibration.
3. Image rectification.

See figure 4.8 for an overview of the calibration procedure. All these steps require a familiar object with known dimensions to calibrate against. Among the three calibration patterns supported by OpenCV, this implementation utilized a black and white chessboard. The chessboard has 6 by 8 squares with sides $\approx \boldsymbol{26\,mm}$ long.

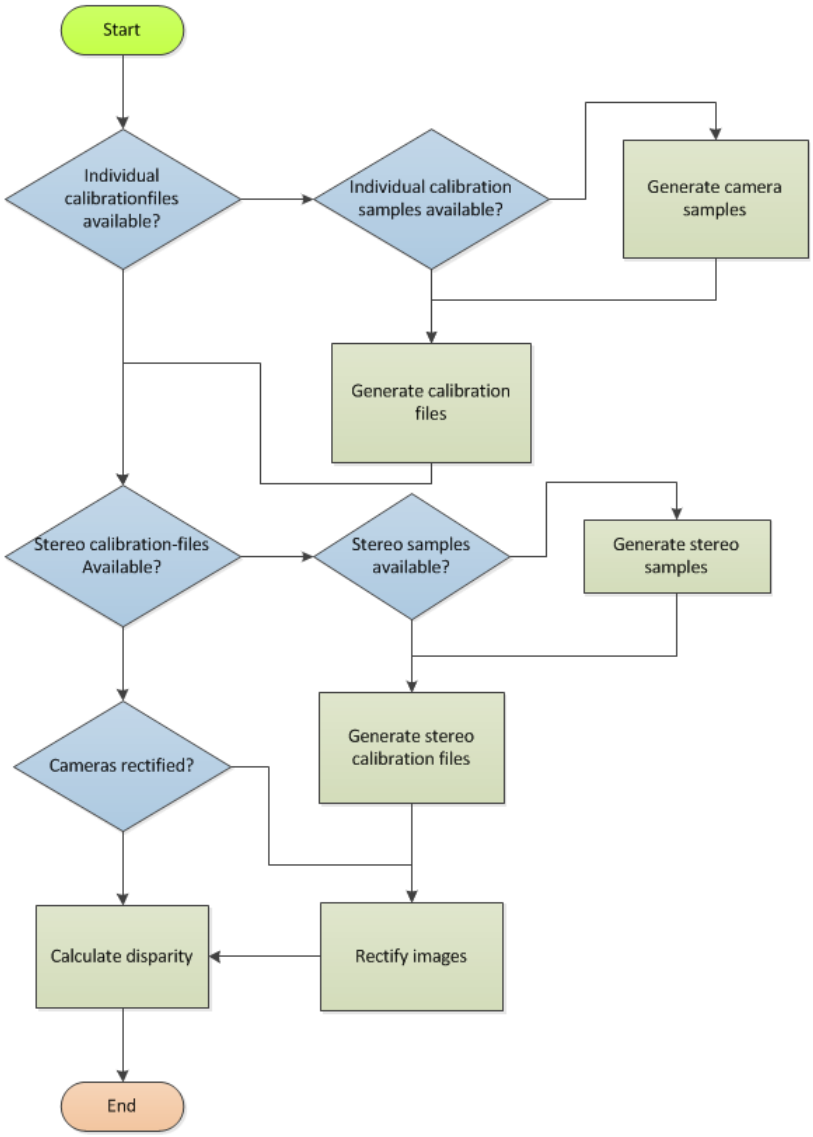
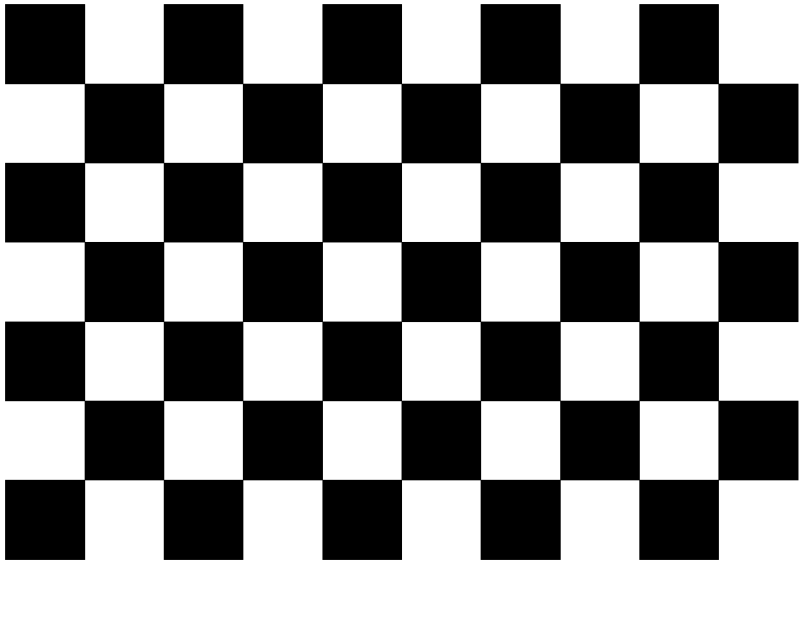


Figure 4.8: An overview of the calibration procedure.



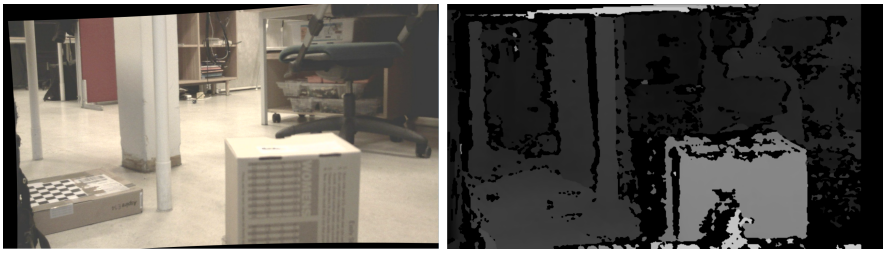
(a) The chessboard calibration pattern. The pattern was printed and taped to a flat surface.

(b) Chessboard detection. This is one of the sample images used to calibrate the camera. Note the distortion in the lower right corner.

Figure 4.9

Single Camera Calibration In this step, the cameras are calibrated separately. The purpose of this calibration procedure is to counter the constant radial and tangential distortion in a pinhole camera, and to relate the image pixels to real world quantities. The result of this procedure is the **3*x*3** camera matrix and the five distortion coefficients mentioned in the theory chapter. The results are stored in a .xml file:

```
<?xml version="1.0"?>
<opencv_storage>
<cameraMatrix type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    1.4478141049219482e+003 0. 6.6274484776761142e+002
    0. 1.4432743079138295e+003 4.7609546427843065e+002
    0. 0. 1.
  </data></cameraMatrix>
<distCoeffs type_id="opencv-matrix">
  <rows>1</rows>
  <cols>5</cols>
  <dt>d</dt>
  <data>
    -2.6128696949919589e-001 3.4600669963821584e-001
    -2.2331413545278616e-003 -2.5710895791919218e-003
    -3.7144316064113458e-001</data></distCoeffs>
</opencv_storage>
```



(a) Left camera image. (b) Disparity map.

Figure 4.11: The result of StereoSGBM.

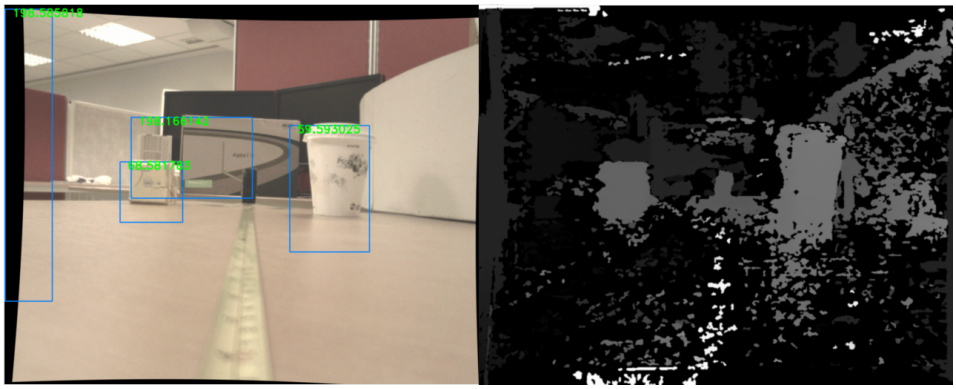
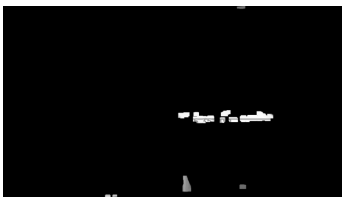
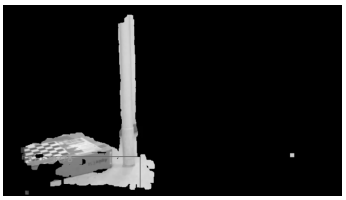
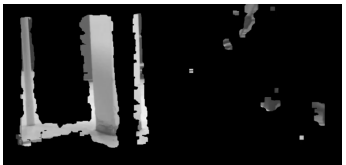


Figure 4.12: Dummy text.

When all the image samples has been read into the program, the program will check if the chessboard can be detected. If the chessboard is present in the image, the position of the corners will be stored in







Stereo Calibration

Stereo Rectification

4.3.5 Stereo Matching

4.3.6 Finding Obstructions

4.3.7 Distance Measurment

4.3.8 Problems Encountered During Implementation

Chapter 5
Assessment

5.1 Vanishing Point Detection

5.2 Depth Perception and Obstruction Detection

5.2.1 Real-time Disparity Map Calculation

5.2.2 Range Measurement and Object Detection

5.2.3 Weaknesses and Limitations

blabla

Chapter 6
Conclusion

References

[3dC] Camera calibration with opencv. http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html.

[Asp13] Petter Aspunvik. Robotisert vedlikehold. Master's thesis, NTNU, 2013.

[Ber13] Mikael Berg. Navigation with simultaneous localization and mapping. Master's thesis, NTNU, 2013.

[Bra08] Gary Bradski. *Learning OpenCV*. O'Reilly Media, 2008.

[Bra13] Samarth Brahmbhatt. *Practical OpenCV*. Apress: Berkeley, CA, 2013.

[can] Canny edge detection with opencv. http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[mon07] A.J. Davison, I.D Reid, N.D Molton, and O. Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6), jun 2007.

[DWB06] H Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, (2), jun 2006.

[GNL11] Demetrios Gerogiannis, Christophoros Nikou, and Aristidis Likas. A split-and-merge framework for 2d shape summarization. *Image and Signal Processing and Analysis (ISPA), 2011 7th International Symposium on*, pages 206 – 211, sep 2011.

[GNL12] Demetrios Gerogiannis, Christophoros Nikou, and Aristidis Likas. Fast and efficient vanishing point detection in indoor images. *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3244 – 2347, nov 2012.

[Hir08] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2), 2008.

[NN00] Firstname 1 Name1 and Firstname2 Name2. A dummy title. *A Fake Journal*, 1(1):000–000, June 2000.

[XWS06] Jian Xu, Han Wang, and Andrew Shacklock. Visual guidance for autonomous vehicles. In *Autonomous Mobile Robots*, chapter 1, pages 5 – 40. CRC Press, may 2006.

Appendix A

Setting up a project with Qt and OpenCV