Investigation into the use of the Microsoft Kinect and the Hough transform for mobile robotics

Prepared by:

Katherine O'Regan

Department of Electrical Engineering
University of Cape Town

Prepared for:

Robyn Verrinder and Assoc. Prof. Fred Nicolls

University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Master of Science degree in Electrical Engineering, by dissertation.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

· May 2014 ·

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced
3. This dissertation is my own work
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signed:
Date: 8 May 2014

“A picture is worth more than ten thousand words” - anonymous

Acknowledgments

I would like to thank the following people:

- Robyn Verrinder and Assoc. Prof. Fred Nicolls for providing help wherever needed, and for helping me to keep my project focused.
- Stefano De Grandis for helping me with problems I encountered along the way.
- Jatin Harribhai and the rest of the Robotics and Mechatronics and Digital Image Processing Research Groups for providing insight into the problem.
- And, last but not least, my parents, for supporting me throughout my Undergraduate and Postgraduate studies.

Abstract

The Microsoft Kinect sensor is a low cost RGB-D sensor. In this dissertation, its calibration is fully investigated and then these parameters are compared to the parameters given by Microsoft and OpenNI. The parameters found were found to be different to those given by Microsoft and OpenNI therefore, every Kinect should be fully calibrated. The transformation from the raw data to a point cloud is also investigated.

Then, the Hough transform is presented in its 2-dimensional form. The Hough transform is a line extraction algorithm which uses a voting system. It is then compared to the Split-and-Merge algorithm using laser range finder data. The Hough transform is found to compare well to the Split-and-Merge in 2 dimensions.

Finally, the Hough transform is extended into 3-dimensions for use with the Kinect sensor. It was found that pre-processing of the Kinect data was necessary to reduce the number of points input into the Hough transform. Three edge detectors are used - the LoG, Canny and Sobel edge detectors. These were compared, and the Sobel detector was found to be the best. The final process was then used in multiple ways - first to determine its speed. Its accuracy was then investigated. It was found that the planes extracted were very inaccurate, and therefore not suitable for obstacle avoidance in mobile robotics. The suitability of the process for SLAM was also investigated. It was found to be unsuitable, as planar environments did not have distinct features which could be tracked, whilst the complex environment was not planar, and therefore the Hough transform would not work.

Contents

List of Figures

List of Tables

List of Definitions

RGB-D sensor — a sensor which has both a colour sensor (RGB) and a depth sensor.

RANSAC — Random Sample Consensus - a parameter estimation and model fitting algorithm

SLAM — Simultaneous Localisation and Mapping, an algorithm which allows a robot to map an unknown environment, while localising itself within the map.

UAV — Unmanned Aerial Vehicle.

Chapter 1

Introduction

1.1 Background to Study

Mobile robotics is an important field of study within the Electrical Engineering community. One very important part of the study of robotics is allowing robots to see, and therefore interact with their environments. This can allow the robot to avoid obstacles in its environment, as well as map and locate itself in its environment (using simultaneous localisation and mapping, or SLAM).

In order to allow complete autonomy in robotics, all processing that allows the robot to interact with its environment needs to be performed in real-time on-board the robot. One of the major problems with this is that typical sensors used in robotics output large amounts of data - for example, the Kinect sensor, which is used in this dissertation, outputs of the order of 300 000 points per scan. If this scan is performed 10 times a second (as it is with the Kinect), the amount of data mounts very quickly.

In order to account for this, and reduce the amount of data that needs to be stored by the robot, feature extraction algorithms are used. Many of these work well for stationary cameras, but do not work well for mobile robots, as the exact location of the camera within the global reference frame is not always known

Therefore, in this study, the Hough transform is investigated. For the Hough Transform, the exact location of the robot in a global reference frame is not necessary. The location of objects in relation to the robot will simply be found. This is first compared to another algorithm using 2-dimensional laser range finder data. This is performed to evaluate whether the Hough transform compares well to other 2-dimensional algorithms before extending it into 3-D. The Laser Range Finder is used for data collection as it is a sensor which has been commonly used in robotics — and therefore provides a good starting point for this study. Its extension into 3-dimensions is then described, and its suitability for use in mobile robotics, both for obstacle avoidance, and feature extraction for SLAM is investigated.

Before the Hough transform is investigated, the Kinect sensor is fully investigated for its accuracy. The Kinect sensor is relatively low-cost, and as such may be more suitable for use that laser range finders (such as time-of-flight sensors), which can be prohibitively expensive. This dissertation will investigate the accuracy of the data collected by the Kinect sensor, and then its use with the Hough transform as a sensor for mobile robotics.

1.2 Objectives of the Study

The SLAM process involves many different “parts” or sections which make up the SLAM algorithm. These are shown in Figure 1.1. In this figure, the parts of the SLAM process which are investigated in this study are shown in red. These two parts are an investigation of the processing and calibration of the sensor (in this case the Kinect), and the use of the Hough transform as a feature extractor. Therefore, the main objective of the study is to determine whether the 3-D Hough transform along with the Kinect sensor can be used in various different ways for Mobile robotics. Specifically, the effectiveness of the Hough transform in reducing the amount of data that needs to be stored on-board is investigated.

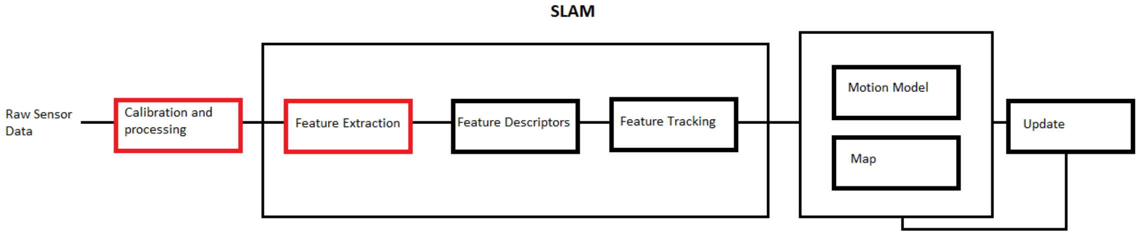

Figure 1.1: Figure showing SLAM process, with the parts investigated in this study shown in red.

First, the method for converting the raw disparity data output by the Kinect sensor into a point cloud, is investigated. The camera system is also fully calibrated. The focal length values provided by Microsoft are an average value for Kinect devices, and are therefore likely to be inaccurate for individual devices. The OpenNI values will also be used for comparison. The disparity values will then be converted into depth values (after the conversion parameters have been found), for each of these three sets of values. The differences are determined, and the results are analysed. This is performed to ensure that the data are as accurate as possible before feature extraction is performed. The effects of lighting conditions as well as the reflectivity of the objects in the scene on the output of the Kinect sensor are determined. The purpose of this is to evaluate the Kinect depth sensor’s performance in indoor environments.

Second, a development and analysis of two feature extraction algorithms for use on

2D laser range finder data are conducted. These algorithms are the split-and-merge algorithm and the Hough transform. The efficacy of each of the algorithms is determined. The speed and the accuracy of the algorithms are measured (by measuring the processing time, and whether the correct lines are fitted to the data). This is performed to determine how the Hough transform compares to the Split-and-Merge in 2D for line extraction. The reason for using the 2D laser range finder is because it is a sensor which is commonly used in robotics. It is also relatively easy to use, and provides useful data to test the 2-dimensional versions of these two algorithms on.

Finally, the Hough transform will be extended into three dimensions. Any pre-processing of the data that is required will be developed and tested, and finally the efficacy of the Hough Transform for reducing the number of points input into SLAM will be determined.

Based on the above experiments, conclusions are drawn and recommendations for future work will be made.

1.3 Software and Equipment

For this study, all of the code is developed by the author (with the exception of the Calibration code used in Chapter 3, and the Edge Detectors used in Chapter 5). The code was developed in MATLAB 2011b for Linux.

All the tests in this study were run on a Sony Vaio laptop with the following specifications:
Processor: Intel Core i7-2640M CPU at 2.80 GHz x 4
Memory: 4.8 GiB
Operating System: Ubuntu 12.04 LTS, 64 bit
Disk: 358.4 GB.

Two different sensors were used in this study. First, a Hokuyo Laser Range Finder with the following details:
Model: Hokuyo URG-04LX Scanning Laser Range Finder.
Light source: infrared laser with wavelength 785 nm
Scan area: 240°
Angular Resolution: 0.36°
Scanning frequency: 10 Hz

The other sensor is an XBox 360 Kinect sensor. Two of these are used in this dissertation, both of them with the same specifications.

1.4 Scope and Limitations

The scope of this project is to use the 3D Hough Transform as a feature extraction algorithm on the output of the Kinect sensor. In this study, it is used as a data reduction technique to limit the amount of data that would need to be stored by a mobile robot. The Kinect is also fully calibrated and the method for converting the disparity data to depth data and then to a point cloud is investigated. Only the disparity data collected from the Kinect sensor are used, the RGB data are ignored for the purposes of this project. As the focus of this study is on characterisation and calibration of the Kinect sensor and on the extension of the Hough Transform to 3D, feature tracking and improvements of the SLAM algorithm itself are not investigated.

The data collected for this project will all be sparse data - one or two boxes in a small (4 m x 4 m) environment with distinct walls. These data are sufficient to determine the suitability of the Hough Transform. The Kinect is moved around the boxes to simulate the movement of a robot. It will not be placed on board a robot, nor is it used in any other environments. The environment will be kept sparse, to aid the Feature Extraction. Only two Kinects are used for the calibration, as this was all that was available. Data from an online dataset created by Sturm *et al* [1] is used in Chapter 5.

1.5 Plan of Development

This dissertation contains seven chapters in total. The contents of each of these chapters is described below.

The first chapter is this introduction. It contains a brief introduction to the problem, as well as the objectives of the study, the methodology used in this project and the scope and limitations of the project.

The second chapter is a comprehensive literature review. It contains a more substantial background and history section, as well as a full review the current work with the Kinect in computer vision applications. There is also a section describing how the Kinect has been used for robotics up to now. Work on the calibration and methods for converting the disparity data from the Kinect are also reviewed. Finally, the literature about the Hough Transform and its development is reviewed, along with other feature extraction algorithms that have been used with the Kinect and for robotics.

The Chapter 3 investigates the conversion of the disparity data from the Kinect to depth data, and then to a point cloud. The Kinect sensors are then fully calibrated. The focal length obtained is compared to the focal lengths given by OpenNI and Microsoft. The parameters for the disparity-to-depth mapping are then determined for each of the focal lengths. The raw data are then converted to depth for each of the sets of parameters, and the differences which the sets of parameters cause are analysed. Finally, the effect of different environmental conditions on the Kinect output is determined.

The fourth chapter contains the theory behind the Hough transform in 2-dimensions and the split-and-merge algorithm. Then, the two algorithms are tested for various parameters (such as speed and accuracy), and are then compared to determine how well the Hough transform compares.

The Chapter 5 contains the theory of the development of the Hough Transform from 2D to 3D. It also contains a basic speed test of the Hough Transform in 3D. Then, edge detectors are tested, as it is decided that pre-processing of the data is required. The theory of the edge detectors is discussed, and the algorithms are tested. Finally, the best edge detector is chosen and used on the raw point cloud data before it is input into the Hough Transform. The speed and efficacy of the Hough Transform at extracting planes in the data is then determined. The final algorithms is run on multiple datasets to determine these factors (speed and efficacy).

The sixth chapter contains the conclusions drawn from all the tests run in the earlier chapters. The research objectives are analysed, and the research questions are answered.

The seventh chapter contains the recommendations for future work based on the work done in this dissertation.

Chapter 2

Literature Review

This chapter gives an overview of the current state-of the art in using the Kinect sensor for applications in computer vision. It also reviews the work in using the sensor for mobile robotics. Feature extraction algorithms that could be applied to the resulting data are identified and discussed, and their suitability for use in various applications in mobile robotics is investigated.

First, the use of the Kinect sensor in various applications will be reviewed, followed by a more specific review of the work using the Kinect for Mobile Robotics. The work detailing the Calibration and Modelling of the Kinect Sensor will also be reviewed. Thereafter, various feature extraction algorithms (as used in mobile robotics) will be investigated.

2.1 The Kinect Sensor

This section will review the current state-of-the-art in terms of using the Kinect sensor for various applications. The first subsection reviews the work using the Kinect sensor for various computer vision applications. The second subsection will review the literature that involves the applications of the Kinect sensor for mobile robotics.

2.1.1 Application of the Kinect Sensor in Engineering and Computer Vision

The Kinect sensor has been used for many applications in Computer Vision and Engineering. The Kinect has been used in a clinical context for rehabilitation [2, 3, 4, 5], prevention of future ergonomic problems [6] and improving the quality of life of people with disabilities [7]. Additionally, it has been used for applications in human-computer interaction [8, 9, 10, 11], as well as in education [10, 12, 13]. It has also been used extensively in robotics (Section 2.1.2).

For clinical rehabilitation, the Kinect was used in multiple ways. Most of the applications involve helping people with motor control disabilities (whether through a stroke or SCI (Spinal Cord Injury) or TBI (Traumatic Brain Injury)). These rehabilitation tools have taken multiple different forms [2, 3, 4, 5].

Lange *et al.*. present a game-based rehabilitation tool using the Microsoft Kinect sensor in [2]. It is a novel, interactive game that can be used for balance training in adults with a neurological injury (either SCI or TBI). The game is an interactive game where the patient had to reach out to grab "gems" from the walls of a "cave". One issue that they discovered was that most of the patients could not perform the calibration pose, or could not perform it for long enough for the camera to calibrate. The game itself was positively received by the patients and the doctors. A simpler game is also presented in [4].

Another rehabilitation tool is presented in [5], where the Kinect tracks whether or not a specific movement is performed correctly by the patient. A baseline is formed by the user before what is called an "intervention" phase — where the patient is required to perform a specific movement a certain number of times. This was shown to have positive effects on the ability of the patient to perform the movements. A similar technique is used in [3].

In [6], a system is developed to record the postures of people in the workplace. The aim was to prevent musculo-skeletal injuries caused by bad posture. In [7], the Kinect is used both to help with the rehabilitation process and to improve patients quality of life while in rehabilitation. Tele-rehab is a fairly common method of helping stroke patients. One of the major problems, however, is that most systems can not distinguish between the paralysed and healthy sides of the body. This system monitors patients in every day activities and relays the information to doctors so that they can adjust their treatment.

The other main area in which the Kinect is used is for human-computer interaction. In [8], a solution to the problem of recovering and tracking the position, orientation and full articulation of the human hand is developed. The Kinect sensor is used so that this can be done with markerless observations. The authors use a model-based system, to track the hand. They achieve robust 3D tracking of hand articulation in close to real-time. The study is theoretical, but they state that it could be used in a context to allow for a robot to understand human grasping.

A more practical study is presented in [9]. Here, the authors investigate how a machine can be made to understand human hand gestures using a Kinect sensor. They apply their system to arithmetic computation and a rock-paper-scissors game. Their system works well, and can detect a wide array of gestures. The rock-paper-scissors game can determine the winner between the computer and a human, showing that the computer can correctly identify which hand gesture the person is making.

The Kinect is used for hand-tracking and rendering for use in wearable haptics in [11]. A hand-tracker is developed that allows for animation of a hand avatar in virtual reality. The position of the fingertips is measured by the Kinect using a tracker that is developed by the authors.

The Kinect has also been investigated for use in education. In [10], its use for teaching natural user interaction is investigated. The authors propose that the Kinect offers a novel way to teach natural interaction in a classroom setting. They present various activities which can be used with the Kinect and the libraries given in OpenNI to teach human-computer interaction and natural user interaction.

The use of the Kinect for robotics education is investigated in [12]. They propose that the Kinect is a good sensor to introduce robot sensing to students. They show that the Kinect can be used to teach students about the complexities of robotic sensing, and to introduce various concepts, such as data fusion and obstacle avoidance.

Finally, in [13] the potential of the Kinect to be useful as an interactive technology for use in a teaching and learning environment is investigated. The authors state that the Kinect could be used to enhance the teachers' use and manipulation of multimedia in the classroom, as well as to encourage student interaction. They show that the implementation of the Kinect has some technical constraints, for example large classroom space, lack of easy-to-use development tools and long calibration time.

The Kinect has clearly been used for many different applications, and even within applications has been used in various different ways.

2.1.2 Use of the Kinect sensor in Mobile Robotics

This section presents a review of related work in which a Microsoft Kinect has been used on board a mobile robot.

One issue with using other sensors for mobile robotics is that sensors such as laser range finders are usually expensive, and these costs can be prohibitive [14]. The Kinect is fairly low-cost in comparison [15]. This means that the use of the Kinect sensor, and other similar low-cost RGB-D sensors is growing rapidly within the field of Mobile Robotics.

The Microsoft Kinect Sensor has, been used for three different applications in Mobile Robotics. The first, and most common application, is that of putting a Kinect sensor on a robot to allow it to interact with the environment, as well as to map and navigate through this environment. This application is shown in [14, 16, 17, 18, 19]. The second application, shown in [20], uses the Kinect sensor to observe multiple robots in a scene, identify them and determine how they move through the scene. This is used to produce ground truth data of how the robots moved in the environment. The final application of the sensor is shown in [21, 19]. In these studies the Kinect is used to allow humans to interact with robots. The focus of this literature review is on the first application — using the Kinect sensor to allow a mobile robot to interact with (or interface to) its environment.

There are two main ways in which a Kinect can be used to help a robot interact with its environment. First, it can be used on board a robot to allow detection of objects and to help the robot manipulate them [19, 22] . The second is that the Kinect data can be used by the robot to navigate in an environment. The latter will be the focus of this review, as the Kinect is being investigated for navigation on Mobile robots.

The Kinect has been used for navigational purposes on different types of robots, and in many environments. In [14], it is used in an indoor environment for navigation and target tracking. The authors use a fuzzy logic controller on board the robot for control and target tracking. They also send data over a network for processing with pattern recognition software. The Kinect is used as a replacement for high-end expensive systems. The system works fairly well, but requires further work to achieve target selection and registration.

The Kinect has also been used on-board UAVs (unmanned aerial vehicles) such as quadrotors [18]. The Kinect has been used in applications such as basic localisation using odometry [23], navigation and target tracking [14], depth based navigation and localisation [16], and RGB-D Mapping [24], and RGB-D SLAM [17]. Some of these implementations use both depth and RGB data from the Kinect [14], while others use only the depth data [16].

In [20], the Kinect is used in a ground-truth detection system for use in RoboCup. The authors use the Kinect because it is low-cost and portable. They used the Kinect to map the location of the ball, and robots on the "field". The sensor is only used to observe the scene, rather than actually being mounted on-board a robot. Another reason for using the Kinect is that no special identifiers on the robots are required; due to the inclusion of depth information. They suggest that their system could be used in any application where a low-cost ground-truth detection system is required.

In [19], the use of the Kinect in robotics is investigated. The authors give various applications where it could be used in conjunction with robots. They say it can be used for evaluating and sensing obstacles, grasping objects, and manipulating objects in an environment. They believe it could also be used for human recognition. The Kinect is used for human interaction in [21]. The authors designed a system which allows a humanoid robot to interact with a non-expert user. It is used to identify natural body gestures, which are used to control the robot.

2.1.3 Calibration and Modelling of the Kinect Sensor

Since the Kinect sensor is a fairly recent development, there is very little literature concerning the calibration and modelling of the sensor. There are other range sensors that have been investigated [25]. However, these are operationally different from the Kinect sensor, as they are time-of-flight based depth sensors, rather than a stereo-camera pair which uses triangulation to find depth. There are a few studies that do describe the calibration parameters of the Kinect sensor [11, 25, 26, 27, 28, 29].

The most in-depth calibration of the Kinect is presented in [25]. In this study, the disparity-to-depth mapping of the Kinect depth camera is described in detail. The process is developed from first principles. This process is also described, though in less detail, in [29].

In [25], the authors also describe the process of calibrating the IR-camera. They calibrate the resulting IR image, rather than the raw image, due to differences in pixel density. They find that the pixels are square, and give a final focal length of 586 pixels (px). The baseline is found for the stereo arrangement, for use in the disparity-to-depth mapping. Finally, they determine the lens distortion of the IR-camera.

Many of the other papers assume that the lens distortion is negligible, but most do not justify this assumption [26], [11]. Those that do account for lens distortion ([28], [25]), obtain very similar results for the lens distortion parameters.

Intrinsic calibration is presented in all the papers described above, however, only one paper ([25]) gives the results of this calibration. In [11], results are given for the calibration for the centre point of the image. They used the CLNUI platform to obtain this. Both [25] and [28] use the MATLAB toolbox developed at CalTech [73].

A number of the papers discussed above [26], [27], [28] focus on the calibration of the depth camera with respect to the RGB camera, rather than only the IR-camera. In [27], the authors state that the calibration of the IR projector-camera pair, as given by Microsoft, may not be accurate due to sensor drift, and therefore they choose to re-calibrate the sensors.

2.2 Feature Extraction

This section will review the current state-of-the-art for Feature extraction used in Mobile Robotics. There is also a review of the work related to the Hough Transform, including its applications. The use of the split-and-merge feature extraction is also reviewed.

2.2.1 Feature Extraction for Mobile Robotics

One concern with using the Kinect sensor for mobile Robotics is that many robots have limited on-board computing power. The Kinect sensor outputs massive amounts of data [16] (over 300 000 pixels per scan). The processing power on board most robots is simply not fast enough to deal with the amount of data. This problem has been addressed in a number of ways.

In [14], this problem is addressed by using a server to collect the data from a robot. The Kinect sensor is used as the sensor for path-planning, navigation, obstacle avoidance and object tracking. The server is used to recover all the data from the Kinect sensor, as the processing power on-board the robot is considered to be incapable of real-time processing of the data. The server processes the data, and provides a supervisory role in the control of the robot.

Biswas and Veloso also address this problem in [16]. Instead of using a server to process the data, they choose to reduce the volume of the point cloud. A new algorithm called "Fast Sampling Plane Filtering" (FSPF) is introduced and used along with a local RANSAC (Random Sample Consensus) to complete the objective. They develop the new algorithm because most algorithms that are currently used for plane extraction on raw 3D point clouds are unsuitable in robotic navigation and localisation due to their high computational requirements. To test their algorithm, they perform a basic comparison between FSPF and a Simulated Laser Rangefinder localisation. Their results were encouraging, although the tests were fairly basic, and therefore more testing needs to be performed in order to determine the true efficacy of their algorithm.

The next problem to be solved to use the Kinect sensor for mobile robotics is that of feature matching and registration. There are, once again, a few different approaches that have been used.

One method that has been used extensively is using feature descriptors such as SIFT (Scale Invariant Feature Transform), SURF (Speeded-Up Robust Features), BRIEF (Binary Robust Independent Elementary Features), and ORB (Oriented FAST, Rotated BRIEF). SIFT, SURF and ORB are used in [17] for the feature matching for the SLAM front-end. SIFT and SURF are used because the authors consider them to be the best known feature descriptors, and ORB (which is based on the BRIEF detector) is used because it is significantly faster than SIFT and SURF. They are used in this application as feature descriptors, rather than extractors, and the data used is RGB-D.

Another approach to feature matching is presented in [30]. This approach compares two different types of scan matching techniques. First, matching as an optimisation problem is investigated. They use Iterative Closest Points (ICP). The second method used is feature based matching. This extracts edge points from the image and then uses them to create pairs of points between two images.

Both of the techniques outlined in the above two paragraphs have their merits, although the techniques presented in [17] are considered to be the current state-of-the-art method for matching features in successive images (whether they are RGB images, depth images or RGB-D images).

RANSAC, SIFT, SURF and ORB all have problems in terms of their use for depth-based SLAM. RANSAC requires the location of the camera to be known [31]. This makes it unsuitable for SLAM applications. SIFT, SURF and ORB all require detailed objects in the scene in order for them to work. This is unsuitable for depth data, which is usually fairly sparse in detail.

In [32], a comparison is made between various line extraction algorithms for use with a 2D range finder. The authors conclude that the split-and-merge algorithm is the most suitable for mobile robotics - specifically SLAM. The Hough transform is also used. These two algorithms are investigated in detail in the two section below.

2.2.2 Hough Transform

The Hough transform was first described by Rosenfeld in [33], based on the work of Hough in [34]. Since then, the Hough Transform has been developed in a number of ways. The first development was to use the r-θ parameterisation of the lines, instead of using the $y = mx + c$ parameterisation. This was introduced by Duda and Hart in [35] where the authors state that their reason for changing the parameterisation is that Hough's method of using the slope-intercept parameterisation is complicated because neither the slope nor the intercept is bounded. Using the r-θ parameterisation solves this problem.

The Hough transform was then extended to find curves, rather than lines in pictures. This was first done by Duda and Hart in [35] and was also investigated in [36]. They use an extension of the Hough Transform to find circles in pictures. Another extension is given in [37] where the authors use the method given by Duda and Hart in [35] to find any curve of specific orientation.

Ballard then extended this idea to detection of arbitrary shapes in grayscale images [38]. The work describes how the Hough Transform can be used to detect analytic curves (of the form $f(\mathbf{x}, \mathbf{y}) = \mathbf{0}$) in images. In this paper, Ballard describes a system where any arbitrary non-analytic shape can be transformed into Hough space. This mapping can then be exploited in a number of ways - to detect the same shape, but rotated or scaled, through simple transformations. A method for detecting composite shapes is also presented. The major difference in their work is that directional edge information is used.

Bias and noise in the Hough transform is investigated in [39] and [40]. In [40], a complete survey of the Hough Transform is presented, along with its uses. The authors discuss ideas for implementation and compare it to other transformations, such as the Radon Transform (of which the Hough Transform is a special case). The authors also consider where the Hough Transform could be used, due to its usefulness in detecting shapes in images containing noisy data.

The next advancement of the Hough Transform is the development of the Randomised Hough Transform, first presented in 1990 [41]. In this method, instead of transforming one pixel into parameter space, n pixels are chosen at random (where n is the number of parameters in the curve that is to be detected). All of these points are then mapped to parameter space. The authors do this by solving a simultaneous equation involving the two parameters (note: this, in the case of the straight line, involves mapping two points on a line, to a single point in parameter space - representing that line). The improvement found using this method is investigated in [41], [42]. In [41], the Randomised Hough Transform is further developed to make it more efficient.

Multiple algorithms are investigated along with the standard Hough Transform and the Randomised Hough Transform in [42]. The first is the Probabilistic Hough Transform, first presented by Kiryati in 1991 [43]. This algorithm makes the improvement of using a probabilistic method for selecting a subset of the original point set to use for the Hough Transform. The author suggests that the small decrease in performance is justified by the large reduction in execution time.

The next algorithm investigated in [42] is the Adaptive Probabilistic Hough Transform [44]. This algorithm monitors the voting grid. As soon as one (or more) significant peaks are detected in the grid, the voting process is terminated. Only the cells in which those peaks were detected are used in the rest of the algorithm. To do this, a list of the cells with the highest voting number is kept and ordered after each vote is added. To maximise efficiency, this list is only created after a group of points has been used in voting. Once a "winner" emerges in the list, the voting process is terminated. This process results in a faster, more efficient algorithm, due to the reduction in voting time.

In [42] the above methods (Standard HT, Randomised HT, Probabilistic HT and Adaptive Probabilistic HT) are compared in terms of their distinctive characteristics (whether it is "complete" (deterministic), whether it contains a stopping rule, whether it contains an adaptive stopping rule, whether it deletes selected planes from the point set, whether it only touches one cell per iteration and whether it is easy to implement). The Randomised Hough Transform has most of these characteristics (it is not deterministic and it does not have an adaptive stopping rule. The algorithms are also compared on the execution time, for which the randomised Hough transform is determined to be the best on this parameter.

One of the main drawbacks to the Hough Transform is the high computational cost of the voting scheme [45]. This is why many of the improvements and investigations involving the Hough Transform have involved making the algorithm more computationally efficient, particularly in the voting step of the algorithm [44, 43]. In [45] another method for improving the voting schemes presented, that allows real-time use of the Hough transform. It also provides a cleaner voting grid and makes the algorithm more robust. The approach operates on clusters of points which are found to be nearly colinear. Votes are then cast for the cluster using an oriented elliptical-Gaussian kernel which models the uncertainty in the line associated with that kernel.

Extension into 3-Dimensions

The Hough Transform has also been extended to allow plane fitting in 3Dimensions as presented in [46, 47, 18]. Most extensions use a variation on the Randomised Hough Transform [18, 48]. They all use the Hesse parameterisation of the plane:

$$p_x \cos\theta + p_y \sin\phi \sin\theta + p_z \cos\phi = \rho \tag{2.1}$$

This equation fully parameterises the plane in three-dimensions, and can therefore be used in the same way as the Hough Transform in 2Dimensions. It is more fully explained in Chapter 5.

The Randomised Hough Transform is used in [18] for the flight control of a quadrotor helicopter. They use it with the Kinect sensor to track the movement of the helicopter in space. The Hough Transform is used to detect the ground plane. This allows the Kinect data to be used as ground-truth data, so the control of the helicopter can be optimised.

Dube and Zell also use the Hough Transform in conjunction with the Kinect sensor for 3D plane extraction [48]. They use a variation on the Randomised Hough Transform to reduce to the amount of data stored. They use the Hough Transform in conjunction with the LO-RANSAC algorithm, which estimates model parameters in noisy sets. The output is the set of parameters for the most likely model. They transform only three points (output by the RANSAC algorithm) into parameter space - similar to the approach of the Randomised Hough Transform. Their algorithm appears to work well - allowing plane extraction in around 1 ms. However, the algorithm can only be used to detect walls, rather than objects in a scene, due to the way the algorithm samples the data. Another optimisation to the 3D Hough transform is proposed in [49]. They address the problems of noisy data, high memory requirements and computational complexity. They do this to allow the algorithm to be used in robotic applications.

The Hough transform has also been used for extraction of other objects, such as spheres [50]. In this paper, Camurri *et al.* compare 4 different Hough transform algorithms for finding spheres in dense point clouds. The algorithms are differentiated by the number of points drawn together in teh computation. They also propose a new method which combines the advantages of the algorithms for faster and more accurate detection.

Dumitru *et al.*. [51] present a method of using the Hough Transform with a 3D laser scanner for interior reconstruction. They propose only reconstructing the architectural features of a building by automatically removing cluttered and foreground data. The software can detect openings, such as windows and doors.

In [31] a method is presented for creating a 3D plane-based map using a mobile robot. The Hough Transform and RANSAC are used separately to extract 3D planes from point cloud data, and these two algorithms are compared. The authors find that the Hough Transform is more robust, as it is not as susceptible to noisy points as RANSAC. They also find that RANSAC is considerably faster, running in about one third of the time of the Hough Transform.

Finally, Woodford *et al.*. present two improvements to the Hough Transform in 3D [52]. The first of these is the Intrinsic Hough Transform which solves the problem of the large memory requirement of the Hough Transform by exploiting the fact that the parameter space will be sparse. The second is "minimum-entropy" Hough, which removes incorrect votes and substantially reduces the number of modes in the posterior distribution. They demonstrate that these improvements make the Hough transform highly accurate.

Applications of the Hough Transform

The Hough Transform has been used in medical applications [53, 54, 55, 56]; mobile robotics [57, 58, 59, 61, 31, 18]; and in other applications [62].

The Hough Transform has been used in multiple applications in the medical industry. A version of the Hough Transform is used in [53] for recognising tumours in radiographs. The authors split the problem into five parts. First, the lungs within the radiograph and secondly the potential tumour locations are found. Then, the boundaries of these regions are found. These regions are used to search for nodules, and finally the nodules are searched to detect tumours. Their procedure correctly found the tumour content in 5 out of 6 radiographs.

The algorithm is also used in [54] with chest radiographs. In this study, the Hough transform is used for automatic detection of the rib contours. The authors state that this could be used in conjunction with an algorithm that detects tumours. The method would be used to prevent the ribs from being segmented as tumours, finding the frame of reference for the location of the lesions or tumours and finally computing the boundary of the chest cavity. Their method works well, and can be implemented on a very small computer (16-bit).

The Hough Transform has been used more recently on images of the eye [55], to track the progression of tumours or other diseases. The authors' method allows the identifications of lesions or tumours to be improved, or for information from other sources to be compared. They use a point correspondence technique to achieve this.

Another way in which the Hough Transform has been used is for detection of the longitudinal fissure in tomographic head images [56]. This is used to segment the left and right hemispheres of the brain. This allows the ratios of inter-cranial structures to be determined. It can also be used to develop a method for registering the head in a scan-independent coordinate system. To achieve this, they use the Hough Transform in conjunction with a Sobel edge detector. They find their algorithm to be robust with respect to noise in the data, and to anatomic anomalies.

The Hough Transform has also been applied extensively in Mobile Robotics applications. It has been used for mapping environments using a robot [58, 31], SLAM [59, 61], and to determine the location of robots in a system [57, 18].

In [59], the authors use the Hough Transform for Simultaneous Localisation and Mapping in an underwater environment. An imaging sonar is the sensor in the system. The Hough Transform is then used to segment lines from these sonar data. One of the problems that they faced is that the robot was not stationary when the scan was performed, and therefore they need to correct for this before applying the Hough Transform. The technique was successful, although it was slow, and therefore the SLAM algorithm they used was delayed.

In mapping applications, the Hough Transform has been used for mapping applications with two-dimensional range scan data [58] as well as Kinect–type depth data [31]. In [58] a 2D Hough Transform is used to fit a line to the data obtained. The result is then used to map an area. In [31] a 3D Hough Transform is used with the Microsoft Kinect sensor. They find the algorithm to be robust, but slow in comparison to other algorithms.

The Hough Transform has been used for two robotics applications involving the tracking of robots in a system. Stowers *et al.*[18] use the Hough Transform to determine ground-truth data in a quadrotor helicopter system. They use the Hough Transform to segment the floor plane in the image, and therefore the ground truth of the robot system can be determined. In [57] the Hough Transform is used to determine the pose and location of robots in an environment.

The Hough Transform has been used in other applications too, for example in [62], where it is used for a computer-based barcode recognition system. The transform is used to segment the lines and spaces in the barcode. Their technique is found to be robust and successful in solving the problem of automated barcode reading.

2.2.3 Split-and-Merge

The split-and-merge algorithm originates from the work of Pavlidis *et al.* in 1974 [63]. It has been used multiple times for mobile robotics [64], [65], [66], [67].

In [64], [65] and [67] the Split-and-Merge algorithm is used as a line extractor from 2D laser range finder images for use in mapping with a Mobile Robot. Borges and Aldon [65] improve the algorithm to a Fuzzy Split-and-Merge algorithm, which incorporates fuzzy clustering on a Split-and-Merge framework. The authors show that their algorithm is robust and compares well in both qualitative and quantitative tests with other algorithms.

Einsele [66] uses the Split-and-Merge algorithm in a similar but not identical way. It is used for localisation of a robot in unknown indoor environments. They use a laser range finder on board a mobile robot to estimate the location of the robot. The landmarks that may be useful for this process are segmented from the scans using the Split-and-Merge algorithm.

2.3 Use of other sensors in Mobile Robotics

Other sensors have also been used extensively in mobile robotics, such as laser scanners [32, 64, 65]; cameras [68, 69, 70] and sonar scanners [59, 61]. The laser scanners and cameras are primarily used in above-ground environments [32, 64, 68], whilst the sonar scanners are primarily use underwater [59], although they have also been used in above-ground applications [60].

The above-ground environments in which these sensors are used include indoor environments [64], [69], aerial environments [68] and outdoor environments [70]. The underwater environments explored include marinas [61].

These sensors have also been used extensively for SLAM. There has been a lot of work on using a single camera for SLAM [68], [69], which has been dubbed Mono-SLAM. The major issue with Mono-SLAM is that depth data can only be extracted once two successive scans have been made - so that a form of stereo vision can be used. However, using cameras has advantages in terms of tracking, as there is more detail in RGB images.

Laser scanners and sonar scanners have also been used for SLAM [59]. These scanners output depth data directly, which means that the problem that occurs with single

RGB cameras does not occur. However, most sonar scanners are 2Dimensional - meaning that only a single “slice” of the scene is seen by the scanner.

Chapter 3

Characterisation of the Kinect Sensor

This chapter covers the conversion of the raw Kinect data into a 3D point cloud, and the calibration of the built-in IR camera. It also discusses the Kinect sensor's operation. The chapter is split into three sections. The first section deals with the interpretation of the raw Kinect data. It also demonstrates the process of finding the point cloud from the raw depth data. The second section covers the calibration of the sensor. The third section will investigate how the Kinect sensor reacts to different environments.

The accuracy of the Kinect sensor is important, as the accuracy of the sensor directly relates to the accuracy of the features that are extracted in a later section. Since these features will be inputs to SLAM, their accuracy is an important factor. The accuracy of the SLAM map depends on this accuracy, and therefore the Kinect's accuracy must be explored.

3.1 Operation of the Kinect depth sensor

The Kinect depth camera is a structured light type depth camera, which uses a disparity measurement to determine the depth (see Section 3.1.2).

The Kinect depth sensor consists of two devices — an laser projector and an IR camera (see Figure 3.1). The laser projector and camera form a stereo pair of "cameras". In a standard stereo camera set-up, the two cameras both receive light from the scene, and then the two images are compared to triangulate the distance from the cameras to an object in the scene.

In the case of the Kinect arrangement, the laser projector projects a pattern of light and dark points, known as a scatter pattern. This pattern is then reflected by the scene, and the IR-camera receives these reflections. The scene will distort the pattern

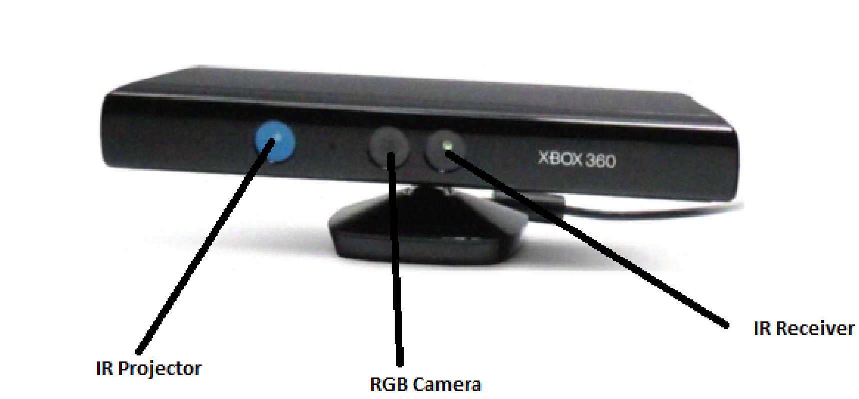

Figure 3.1: Diagram of the Kinect sensor showing the locations of the laser projector, IR receiver and RGB camera [18].

in various ways. Objects which are closer to the projector will distort the pattern more than those at the "reference plane". The reference plane is the plane where, when the Kinect was originally calibrated, the scatter pattern was "set" as a reference pattern. This "reference pattern" is then used along with the pattern received by the IR camera to find the disparity between the reference pattern and the received pattern. For a standard stereo camera arrangement, the reference plane is assumed to be at infinity — because objects at infinity will appear identical to the two cameras. This is not true for the Kinect sensor — this is discussed below. These disparity values can then be converted into a depth map. Finally, this depth map can be converted into a point cloud.

This section will discuss how the Kinect sensor (or the post-processing) performs the operations described above.

3.1.1 Disparity calculation

The first operation of the Kinect sensor is to determine the disparity between the IR-image received by the IR camera and the reference pattern in memory generated by the laser projector. This process takes place on-board the Kinect. The reference image is stored in the firmware of the Kinect. Microsoft obtain the reference image at a known distance of 6 m [74].

For every pixel in the IR image, the Kinect considers a small correlation window (9 by 9 pixels). This correlation window is used to compare the local pattern in a corresponding neighbourhood with the reference pattern at that pixel. The best match will give an offset from a known depth — the disparity [74].

Next, a further interpolation procedure is performed for the best match, giving a sub-pixel accuracy of around 1/8 of a pixel [74]. Given the disparity for each pixel, and the known depth of the reference image, the depth of the objects in the scene can be determined from this process, which is described below.

3.1.2 Disparity to Depth calculation

A triangulation process is used to perform the disparity-to-depth calculation. This process is illustrated in Figure 3.2.

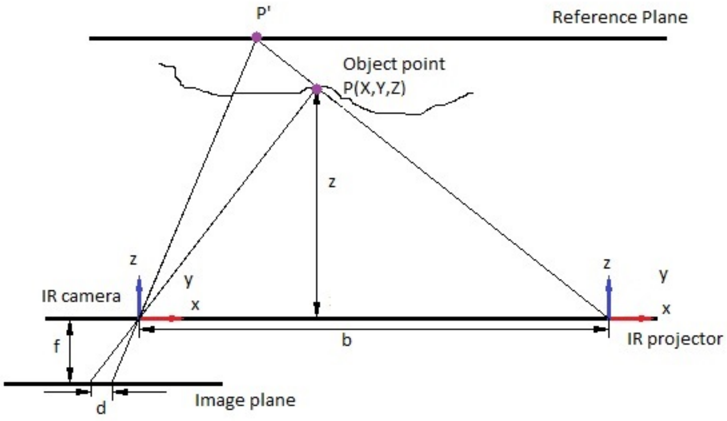

Figure 3.2: Figure showing the relationship between disparity and depth. The laser projector is on the right, and the IR camera is on the left. The value b is the baseline — the distance between the two cameras. P is a point, from the scatter pattern, produced by the laser projector, projected onto an object in the scene. P' is the same point as it would appear on the reference plane (i.e. how it appears in the reference pattern). The focal length is f, d is the disparity, and z is the depth. Note that the y-direction is into the page

Point P is a point in the scene which reflects a point from the scatter pattern produced by the laser projector to the IR camera. This point is offset from the reference plane. Point P' is the same point projected onto the reference plane from the projector. The "z" axis of the IR camera is orthogonal to the image plane and orthogonal to the Reference plane. It also passes through the optical centre of the camera. The IR camera and projector are aligned in the y and z directions, and parallel along the x-direction, but they are offset by a parameter b — the baseline distance (or the distance between the laser projector and IR camera)- in the x-direction. The reference plane shown in the figure is the plane where the Kinect sensor will not find any distortion of the received image from the reference image — i.e. it is the plane where the IR-camera will see the reference image. It is important to note that, because the IR camera and laser projector are offset only in the x-direction, the distortion of the image will only occur along the x-axis.

The relationship between the baseline b in metres, the raw disparity d in pixels, the focal length f in pixels and the depth z in metres is given by the following equation:

$$z = \frac{f \times b}{d} \tag{3.1}$$

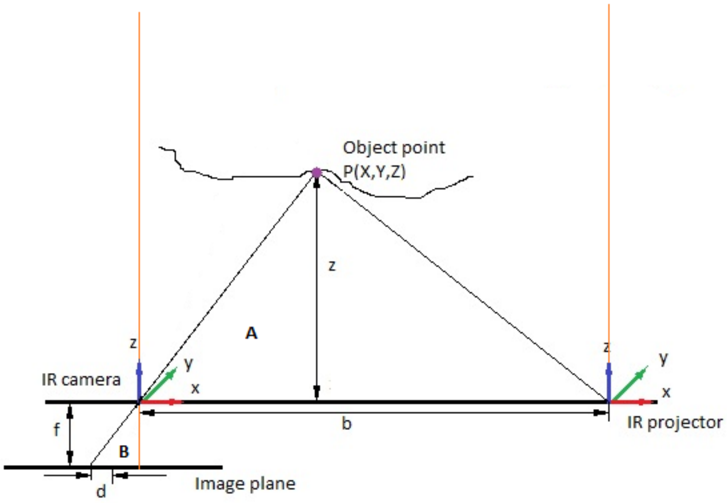

Figure 3.3: Figure showing the relationship between disparity and depth, where the reference plane is at infinity.

This equation can be derived by assuming the reference plane is at "infinity". At infinity, the rays that represent the "true image" (i.e. the rays to P' in Figure 3.2), become parallel. This results in the figure shown in Figure 3.3. In this illustration, the rays to the reference plane are shown in orange. The disparity, if there are no objects in the scene, is zero (as the rays are parallel, and therefore the images for the left and right cameras are identical). In this figure, there are two similar triangles — marked A and B. These triangles allow one to derive this equation:

$$\frac{z}{f} = \frac{b}{d} \tag{3.2}$$

Therefore, by changing the subject of the formula to z, Equation 3.1 is derived.

The Kinect does not output disparity that agrees with the assumption that at zero disparity, the depth is not infinite — because the Kinect arrangement is not the same as a standard stereo camera arrangement, therefore the reference plane is not at infinity, but at 6 m. This means that if there are only objects at 6 m in the scene, the disparity output will be zero (as there will be no difference between the reference image and the received image). Therefore, for the Kinect camera, for objects between 0 and 6 m, this equation is true. The Kinect outputs the disparity in pixels for each (x, y) in the data. The Kinect returns values between 0 and 2047 (i.e. an 11-bit integer) — linearly with depth. The relationship between Kinect disparity, d_k and the normalised disparity, n_d, is given by the following equation:

$$n_d = \frac{1}{8}(d_{off} - d_k) \tag{3.3}$$

The factor of $\frac{1}{8}$ is there because the values of d_k are in $\frac{1}{8}$ pixel units. The factor d_{off} is known as the disparity offset and is particular to each Kinect device.

The final equation for the mapping from Kinect disparity d_k to depth z is given by:

$$z = \frac{b \times f}{(\frac{1}{8} \times (d_{off} - d_k))} \tag{3.4}$$

A calibration of the Kinect IR-camera (as given in the section 3.2) will find the focal length f, distortion parameters and lens centre of the IR-camera. We can then use known points in disparity images to determine the baseline and the disparity offset using least squares. This process is described in section 3.2, where the "typical values" of all the parameters are verified and errors resulting from uncertainty in these parameters are quantified.

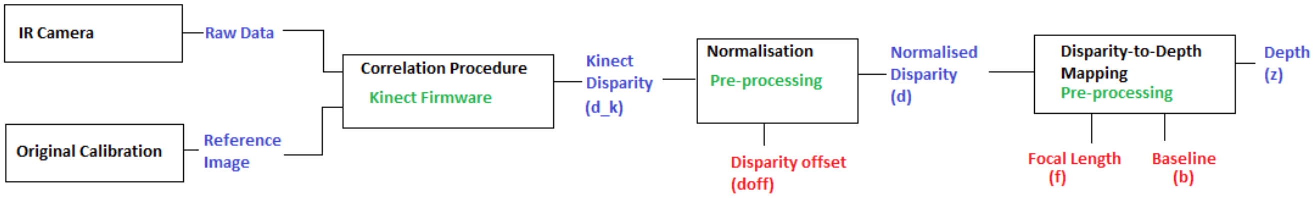

Figure 3.4: Flow diagram showing the processes used to get from the raw IR image to depth. All the parameters and processes used are shown in the diagram.

Figure 3.4 shows a flow diagram of the entire process from the Raw IR image to the calculation of the depth. The processes are shown in the boxes. Each process has a name and a "location", either in the firmware or in the pre-processing of the data which shows where the process happens. The red parameters are parameters that come from an external source (for example focal length). The blue parameters are those obtained in the processes.

3.1.3 Depth to Point cloud calculation

For this section, data were obtained using a Microsoft Kinect sensor and OpenNI, and these data were then used to convert the data into a point cloud. An image of a 500 mm by 500 mm box was taken with the camera 1 m and then 1.5 m away from the box. A different method for performing this is shown in [25]. This method is not used here, as it is not described in sufficient detail.

The raw data coming from the Kinect is the distance from the image plane to the feature, or object, in the scene. Initially, the image plane was assumed to be at the lens of the camera. This means that the data are the orthogonal distances in millimetres from the object to the plane at the lens of the camera. If we assume that the z-axis is orthogonal to the image plane, and the x-axis is horizontal in the image plane, and the y-axis is vertical in the image plane, this means that every pixel in the data is the z-coordinate for that pixel in the point cloud. In order to obtain a point cloud, the x- and y- coordinates need to be determined from the data.

Determining the Angles

In order to find the x- and y- coordinates for every pixel in the point cloud, first the angle from the centre of the image plane to every pixel needs to be determined. This was done using basic trigonometry and the two images described at the beginning of the section.

The test set-up is shown in Figure 3.5 in top, side and front views.

At a distance of 1 m from the Kinect sensor, the length of the box in the image (in pixels) is 288 pixels in both directions. The camera is in the centre of the box, and the angles, θ and ϕ can be determined from trigonometry as follows (Note: θ is equal to ϕ).

$$\begin{aligned}\tan\theta &= 0.25/1\\ \theta &= \arctan(0.25)\\ \theta &= 14.0375^\circ\end{aligned}$$

Therefore the angular resolution of the sensor is 0.0975° per pixel.

This is then verified by repeating the protocol at 1.5 m. The same result was obtained for the box at this distance.

From the data sheet for the Kinect Depth Camera (a Micron MT9M001) [75], the horizontal viewing angle is 57°, and the resolution in the horizontal direction is 640 pixels. Therefore, the angular resolution is $57/640 = 0.0891^\circ$ per pixel. This closely matches the angular resolution calculated above (which is 0.0975° per pixel). This is within 0.01 degrees per pixel. Note that, because pixels are square, the resolution in

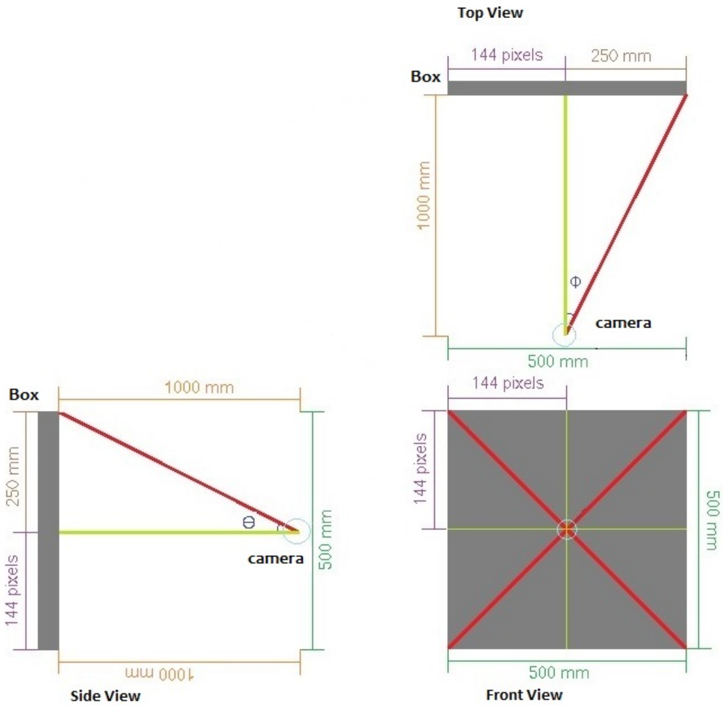

Figure 3.5: Test setup for determining the angular resolution of the Kinect sensor. The grey block is the box used. The setup up is shown in top, side and front views. The red lines represent the rays from the camera to the corners of the box. All the relevant measurements are shown.

the horizontal and vertical direction is the same. The viewing angle in the vertical direction is 47°.

Finding the Point Cloud

The data can now be converted from the depth data into a 3-dimensional point cloud using trigonometry. The angles for each pixel in the image are known from the previous section, and the distance z_r is known — this is the z distance from the image plane to the object in the scene. The angles and lengths are shown in Figure 3.6.

So, working through the trigonometry to find x_r:

$$\begin{aligned} \rho &= \frac{z_r}{\sin(\theta)} \\ r &= \rho \times \cos(\theta) \\ x_r = r \times \cos(\phi) &= \rho \times \cos(\theta) \times \cos(\phi) \\ \rightarrow x_r &= z_r \times \frac{\cos(\phi)}{\tan(\theta)} \end{aligned} \tag{3.5}$$

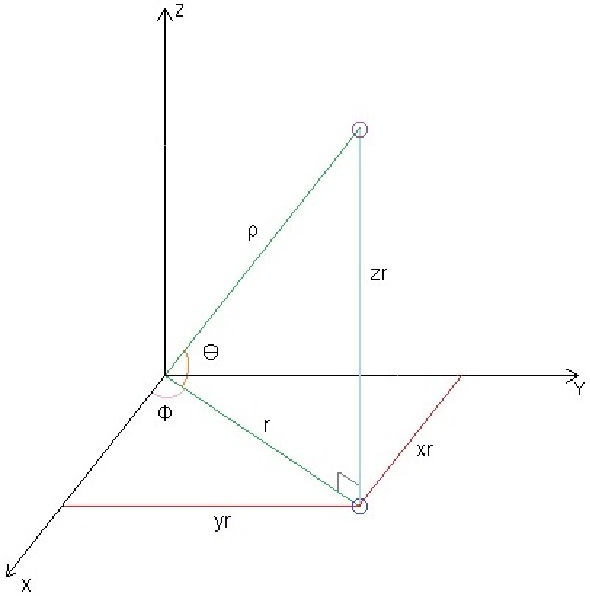

Figure 3.6: Diagram showing the angles and lengths for the trigonometry of finding the point cloud. ρ is the distance from the origin to the point. r is the distance from the origin to the point when it projected into the x-y plane. The angle ϕ is the angle between the x-axis and this projected point. θ is the angle from the x-y plane to the point.

And, to determine y_r:

$$y_r = r \times \sin(\phi) = \rho \times \cos(\theta) \times \sin(\phi) \rightarrow y_r = z_r \times \frac{\sin(\phi)}{\tan(\theta)} \tag{3.6}$$

We now have a point cloud. In order to check that the point cloud was correct, the two images of the 500x500 mm box were processed using the above algorithm. Then, the size of the box in mm in the final point cloud was checked. This conversion was found to be correct, as in the converted image, the box comes out to be approximately 500 mm by 500 mm. For the image where the box is placed 1.5 m away, the box comes out to be about 516 mm by 480 mm. This comes to an error of about 4 percent, which is fairly small. This is probably due to noise in the image used to find the parameters, and in the 1.5 m image.

3.2 Theoretical Meaning of the Calibration Parameters

To find the camera parameters for the Kinect IR-camera, the standard MATLAB calibration toolbox is used [73]. The images that are input into the calibration toolbox are generated using a standard calibration grid with squares of size 4.5 cm which was printed on standard printer paper. The black squares were then covered with heavy duty black masking tape, to prevent any light from being able to get through them.

This calibration grid was then stuck against a window. Sunlight could therefore get through the white squares of the grid, but not the black squares. Sunlight contains light of all frequency — including IR. The IR-projector is then covered with a piece of masking tape, so that the IR-camera did not pick up any points from the scatter pattern. IR-images were taken from multiple angles and distances. Two of these images are shown in Figure 3.7. The two images shown in Figure 3.7 are two extreme examples of the images used for calibrating the IR-camera. The image on the left is an image taken by pointing the Kinect directly at the grid. The image on the right is taken by pointing the Kinect at the grid from an extreme angle (note: the exact angle is not known, as the calibration toolbox allows the user to calibrate without knowing the exact angle and distance of the camera from the grid). At least 20 different images are needed for the MATLAB toolbox to work correctly.

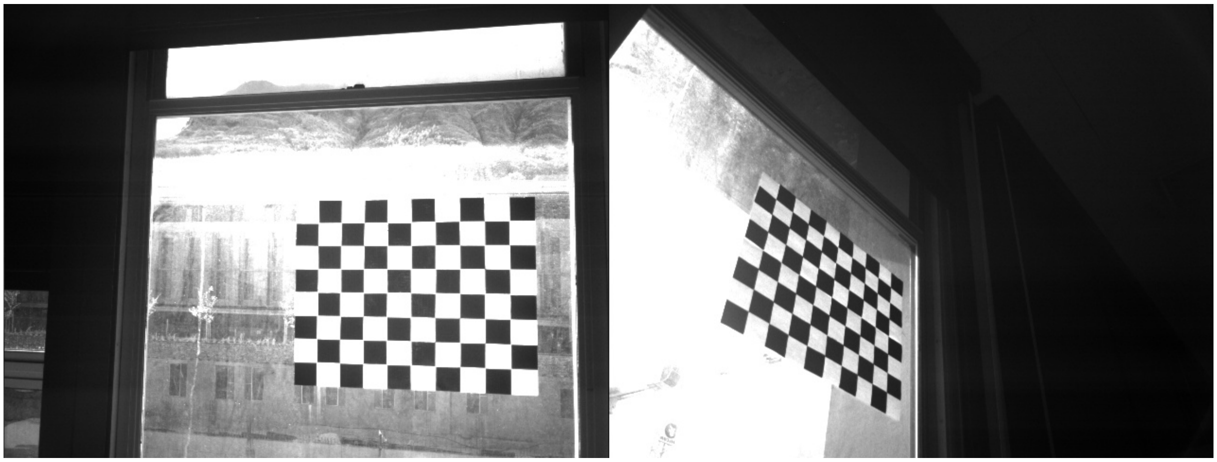

Figure 3.7: Two of the images used for calibration. The images should include multiple angles. The left hand image shows an image where the camera is pointing directly at the calibration board, and the right-hand one shows the camera at an extreme angle.

Twenty-one images of this type were taken for the experiments. These images are input into the MATLAB toolbox. Then, the size of the grid (in this case 4.5 cm by 4.5 cm) and the corners of the grid are given to the system. The toolbox then automatically finds the corners in the grid. This is shown in Figure 3.8 for the same two images shown in Figure 3.7. The toolbox appears to extract the corners in the grid well. If the corners are not correct, there is an option to input an initial guess for the distortion of the image, until the correct corners are found.

The toolbox then performs the calibration using the 21 input images. The parameters are found, and then optimised. As default, the skew and the 6th distortion coefficient are set to zero. Initially, this is assumed to be a good assumption. This assumption will be tested in a later section (Section 3.2.2).

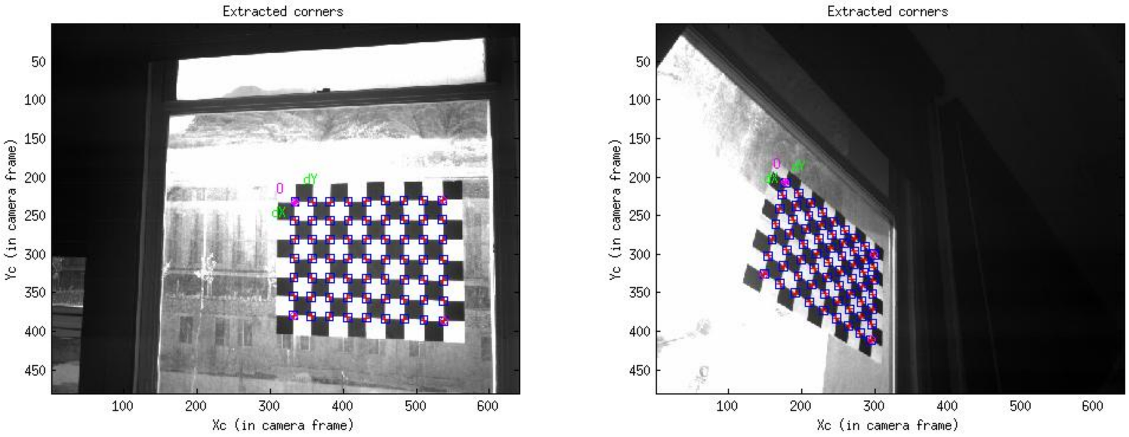

Figure 3.8: Figure showing the corners of the grid, as extracted by the MATLAB toolbox. The blue squares with the red dots inside should be directly over the corners of the grid.

3.2.1 Outputs of the Calibration Toolbox

The calibration toolbox outputs the following intrinsic parameters:

- $fc(1)$ and $fc(2)$ — these are the focal lengths in the x- and y-directions respectively
- $cc(1)$ and $cc(2)$ — this is the location of the principal point of the camera in the x- and y-direction
- $kc(1)$ and $kc(2)$ — these are the first two radial distortion coefficients of the lens of the camera
- $kc(4)$ and $kc(3)$ — these are the first and second tangential distortion coefficients, respectively.
- $\alpha_c(1)$ and $\alpha_c(2)$ are the skew coefficients — these are initially zero

The meanings of these parameters are described below, and their effects on the output of the camera are described

Effect of the distortion parameters on the output of the camera

For this section, a standard pinhole camera model will be described. This model will then be modified according to the parameters that were discussed above.

A standard pinhole camera model assumes that the projection of a point P onto the image plane is found by connecting a line from the point P in the scene to the camera centre. The point where this line passes through the image plane is the projection of the point onto the image plane, p [71]. This process is illustrated in figure 3.9

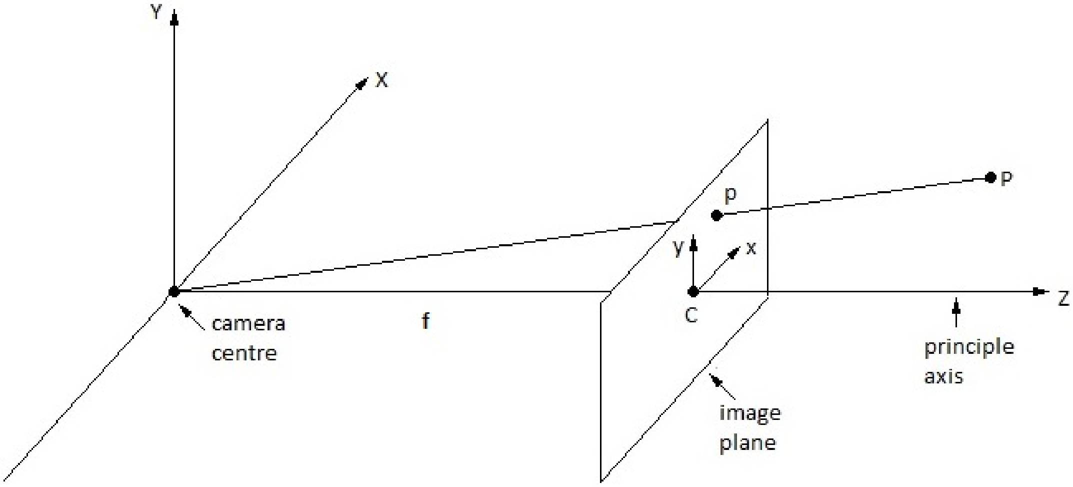

Figure 3.9: Illustration of a standard pinhole camera projection model. c is the centre point of the image plane, p is the point projected into the image plane. The principle axis and camera centre are labelled.

From Figure 3.10 (which is a side view of Figure 3.9), and using similar triangles, the following mapping for a point $P = (X, Y, Z)^T$ in the scene to a point p is:

$$(X, Y, Z)^T \rightarrow (\frac{f \times X}{Z}, \frac{f \times Y}{Z}, f)^T \tag{3.7}$$

The last coordinate can be ignored as it is constant, and then any point $P = (X, Y, Z)^T$ projects to a point at $p = (\frac{f \times X}{Z}, \frac{f \times Y}{Z})^T$ in the image plane.

If a point $P = (X, Y, Z)$ is in the scene, the normalised, pin-hole projection of this point onto the image plane is given by:

$$p_n = \begin{bmatrix} X/Z \\ Y/Z \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{3.8}$$

The projection equation for the system is given by:

$$p = KK \times p_n \tag{3.9}$$

Where KK is a matrix. For a standard pinhole camera, the matrix KK is given by:

$$KK = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The equation KK is modified when parameters like principle point offset and skew are introduced. Lens distortion causes a direct modification of p. The first distortion that

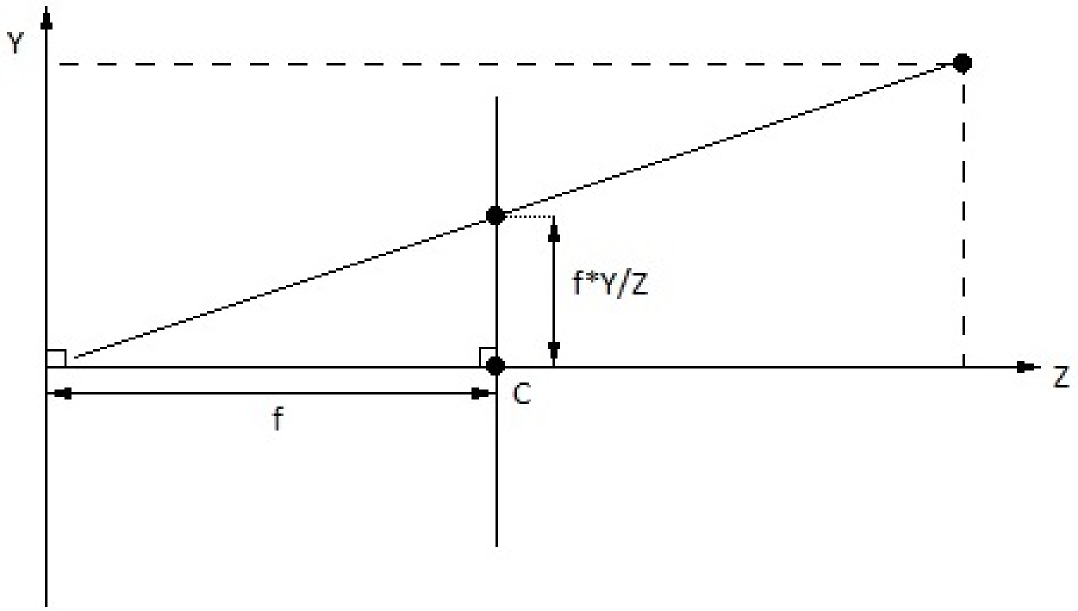

Figure 3.10: Figure showing the "side view" of the pinhole camera model. The focal length (f), centre point (c) and projected points are shown.

will be considered is an offset of the principle point.

Initially, the principle point is assumed to be at the origin of the image plane (i.e. c is at point $(x, y) = (0, 0)$. In practice, this is not always true. This causes the coordinates of point p to be offset. This offset means that point $P = (X, Y, Z)^T$ projects to the point $p = (f \times X/Z + c_x, f \times Y/Z + c_y)$. This changes the matrix KK to be:

$$KK = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

This change in the equation KK fully accounts for an offset in the principle point, which is given by the MATLAB toolbox as cc.

The second property that causes a distortion which causes a change in the matrix KK is the skew parameter. Skew is caused when the angle between the $x-$ and $y-$axes is not 90^o. The skew parameter causes the matrix KK to become:

$$KK = \begin{bmatrix} f & \alpha_c & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Generally, the skew is considered to be zero. This assumption is tested later in the section.

The final adjustment of the KK vector that needs to take place is the adjustment that would account for a focal length that is different in the $x-$ and $y-$directions. This is accounted for by adjusting the KK vector as follows:

$$KK = \begin{bmatrix} f(1) & \alpha_c & c_x \\ 0 & f(2) & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

where $f(1)$ and $f(2)$ are the focal lengths in the $x-$ and $y-$directions respectively.

The radial and tangential distortions, K_1, K_2, P_1, P_2 (Note that P_1 and P_2 are not the same as the point in the scene, $P = (X, Y, Z)^T$) are accounted for by directly changing the point p. The following are the formulas required to change the point p_n to the distorted point p_d:

$$p_d = \begin{bmatrix} p_d(1) \\ p_d(2) \end{bmatrix} = (1 + K_1 r^2 + K_2 r^4) \times p_n + dp \tag{3.14}$$

where $r^2 = p_n(1)^2 + p_n(2)^2$ and :

$$dp = \begin{bmatrix} 2 \times P_2 \times x \times y + P_1 \times (r^2 + 2 \times p_n(1)^2) \\ P_2 \times (r^2 + 2 \times p_n(2)^2) + 2 \times P_1 \times x \times y \end{bmatrix} \tag{3.15}$$

This method for accounting for lens distortion is presented by Brown in [72].

The final formula for the mapping between the normalised pinhole projection point p_n and the final, corrected point p_p is:

$$\begin{bmatrix} p_p(1) \\ p_p(2) \\ 1 \end{bmatrix} = KK \begin{bmatrix} p_d(1) \\ p_d(2) \\ 1 \end{bmatrix} \tag{3.16}$$

Where:

$$KK = \begin{bmatrix} f(1) & \alpha_c & c_x \\ 0 & f(2) & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

and:

$$p_d = \begin{bmatrix} p_d(1) \\ p_d(2) \end{bmatrix} = (1 + K_1 r^2 + K_2 r^4) \times p_n + dp \tag{3.18}$$

where $r^2 = p_n(1)^2 + p_n(2)^2$ and dp is as described in Equation 3.15. The above equations fully characterise the distortions in the IR-camera of the Kinect.

3.2.2 Calibration of the Kinect Sensors

The calibration toolbox described above is used in this section to find the relevant parameters for the Kinect IR sensor. The calibration was run 30 times with a different user input each time (the user input is different because the the user cannot click on exactly the same pixel every time. It also changes because the user may click just outside the grid, or just inside the grid). The results are shown in the table in appendix A. These data are represented as box plots, shown below. Each box plot represents one of the parameters output by the calibration algorithm. The red line on the boxplot represents the mean of all the data collected. The bottom of the blue box represents the 25th percentile of the data, whilst the top represents the 75th percentile. The black dotted lines represent the rest of the data (that is not considered to be an outlier). In some plots, there are "outliers" to the data. These are represented as a red cross.

Calibration of Kinect 1

The first figure (3.11) shows the boxplots for the focal length (in the x and y directions), and the centre point (in the x and y directions). The next set of boxplots (Figure 3.12) represents the first two radial and tangential distortion coefficients for the data.

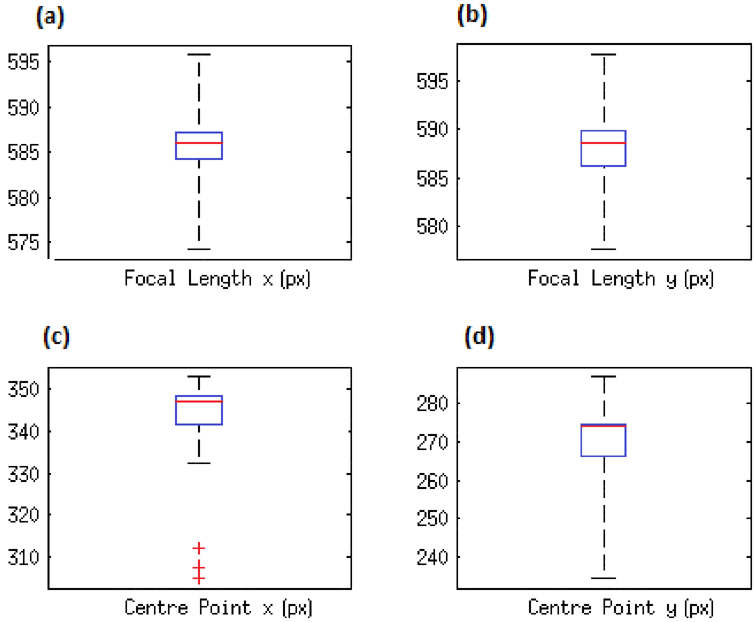

Figure 3.11: Boxplots showing the median and 25th and 75 percentiles of the focal length and centre point for Kinect 1. (a) and (b) show the boxplots relating to the focal length in the x and y directions. (c) and (d) show the boxplots for the centre point in the x and y directions. The red + signs represent the points which are considered by MATLAB to be outliers.

The following table shows the median of each of the sets of data:

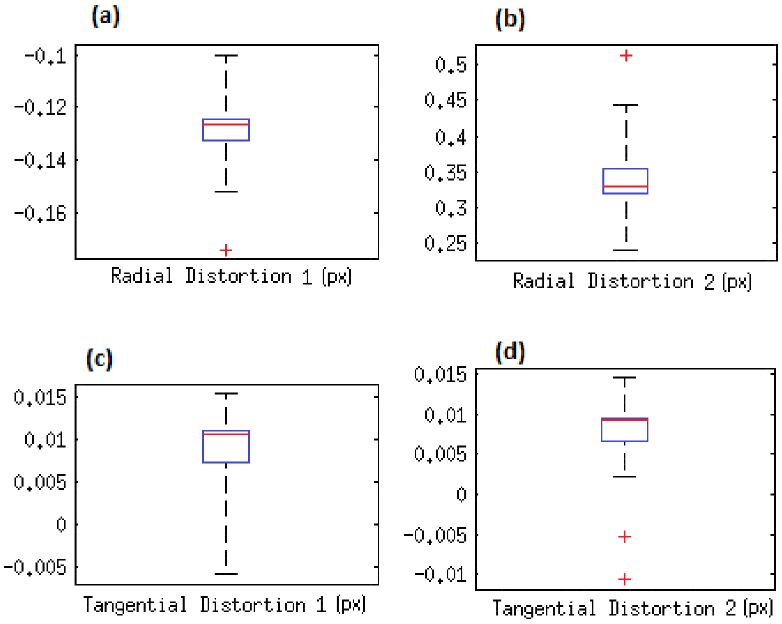

Figure 3.12: Boxplots showing the median and 25th and 75 percentiles of the tangential and radial distortion for Kinect 1. (a) and (b) show the radial distortions (left being the first coefficient, and right being the second coefficient). (c) and (d) show the first and second tangential distortion coefficients.

Table 3.1: Table showing the parameters found for Kinect 1

Parameter	**Value**
Focal Length x	586 px
Focal Length y	588 px
Centre Point x	347
Centre Point y	273
Radial Distortion 1	-0.12623
Radial Distortion 2	0.32846
Tangential Distortion 1	0.010565
Tangential Distortion 2	0.009265

Discussion of the calibration parameters of Kinect 1

The top two boxplots in Figure 3.11 represent the focal length data. From these boxplots, it is clear that the focal length data has a fairly even distribution about the norm. This is clear because the median is approximately in the centre of the box representing the 25th and 75th percentiles, which means that the data are approximately normally distributed. The median of the data for the focal length in the x-direction is approximately 586 pixels. This is slightly higher than expected (Microsoft give a value of 580 pixels as the focal length value for the Kinect). In the y-direction, the median of the focal length is approximately 588 pixels. Once again, this is slightly higher than expected. Also, the two focal length values are slightly different, which is unexpected — they should be closer than they appear to be.

The bottom two boxplots represent the centre point of the camera. From these two plots, it is clear that the distribution is not as well distributed as for the focal length. For the centre point in the x-direction, there are three outliers which are in the region of 305 pixels — these were excluded by Matlab as outliers. The median of the data is very close to the 75th percentile, showing that the distribution is skewed. Another way of showing this skew is to compare the median (347) and the mean (341). These values are quite different, once again showing a skew in the data. The median of the data is at 347 pixels. This value is higher than expected. The expected value is 320 pixels — the centre of the image. This is quite high in comparison. The effect of this will be measured in the next section. For the centre point in the y-direction, there are no points considered to be outliers, however, the median is almost identical to the 75th percentile, showing a large skew in the data. The median is 273 pixels. Once again, this is higher than expected. The expected value is 240.

The top two boxplots in Figure 3.12 represent the radial distortion of the data. The first radial distortion coefficient is evenly distributed. Although the median is quite close to the 75th percentile, the blue box is fairly narrow, showing that the data are quite narrowly distributed. There is one outlier in the data. The median of the data is -0.12623. This is small, but not necessarily negligible, and therefore the first radial distortion coefficient should not be discounted. The fact that the first radial distortion coefficient is negative is indicative of a "pincushion" type distortion. The second radial distortion coefficient is distributed quite similarly to the first radial distortion coefficient, however the median is closer to the 25th percentile. There is, once again, one outlier. The median is 0.32846. This is higher than the first radial distortion coefficient, although the second coefficient is multiplied by p_n^4, rather than p_n^2, and therefore it contributes less to the problem. However, because it is higher the the first radial distortion coefficient, it should not be discounted.

The bottom two boxplots in Figure 3.12 represent the tangential distortion of the data. The distribution of the first tangential distortion coefficient is quite skew — the median is very close to the 75th percentile of the data. The median of the data is at 0.010565. This is very small. The distribution of the second tangential distortion coefficient is also very skew. The median is, once again, very close to the 75th percentile of the data. There are two oultiers. The median of the data is 0.009265, which is, once again, very small. Due to the fact that both the first and second tangential distortion coefficients are very small, they can be discounted from the distortion matrix.

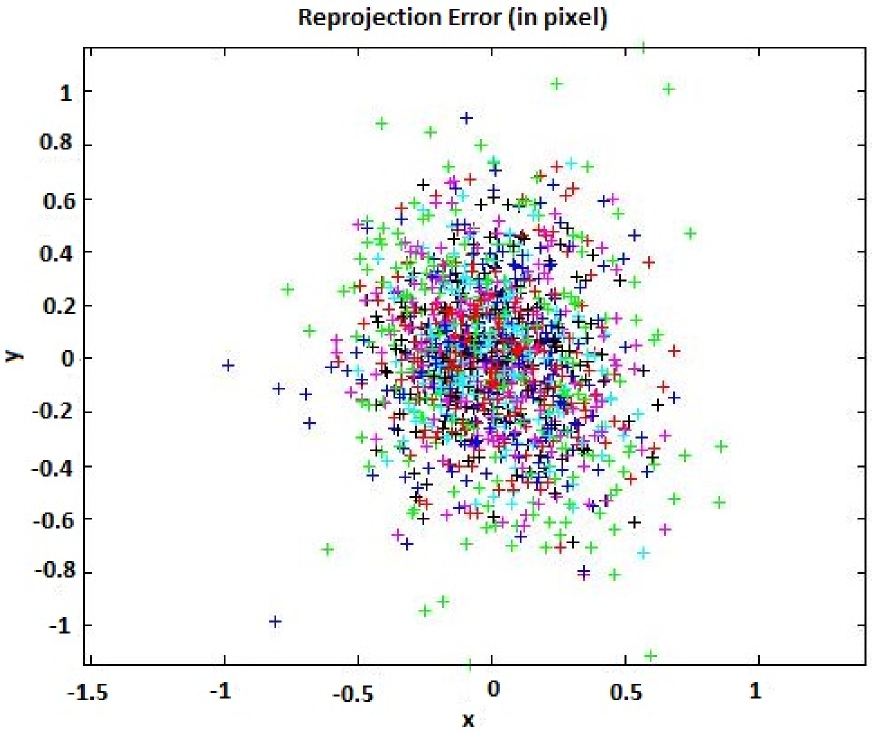

Figure 3.13: Figure showing the reprojection error for each point in each image after calibration. The different coloured crosses represent different images. The crosses are evenly distributed around the origin. The reprojection error is low for all points. This shows that a second optimisation in the calibration is unnecessary

The reprojection error after calibration for each corner in each image is shown in Figure 3.13. This reprojection error is quite low, and therefore a second calibration optimisation is deemed unnecessary.

The position of the grid relative to the world can also be plotted using the MATLAB toolbox. The world-centred and camera-centred views of this are shown in Figure 3.14.

The distortion model of these data, which is shown in Appendix A is very similar to the distortion model for the Kinect found by other studies on the Kinect sensor [25].

Calibration of Kinect 2

The second Kinect sensor is calibrated in the same way as the sensor above, and the data were collected, these data are shown in Appendix A. Boxplots are generated for these data, and the discussion of these boxplots is below. The boxplots in Figure 3.15 show the data for the focal length and the centre point of Kinect 2. Figure 3.16 shows

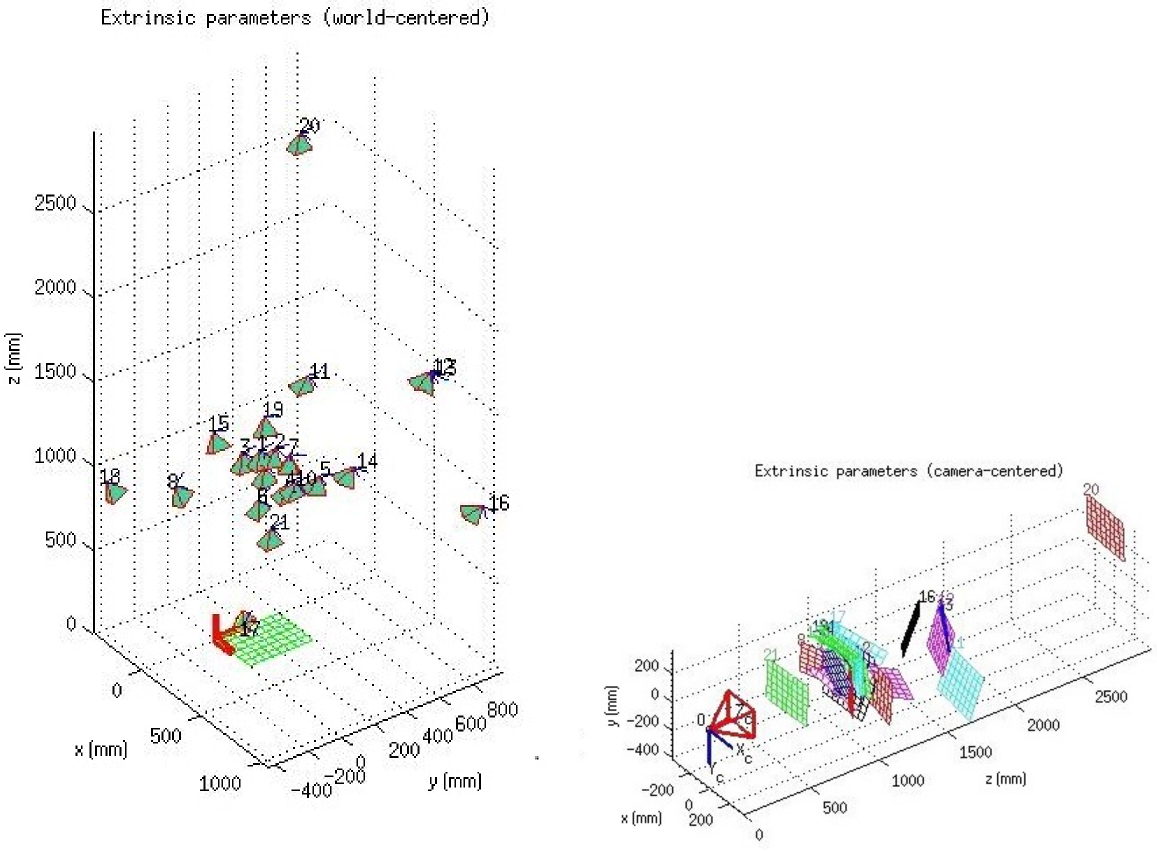

Figure 3.14: The position of the camera and the grid, relative to the world and to the camera. This shows the variation in the images used for the calibration. There is a wide range of distances and angles in the images taken, which is necessary for the calibration procedure.

the boxplots for the first and second Radial Distortion Coefficients, and the first and second Tangential Distortion Coefficients. The data used for these boxplots are shown in Appendix A.

The following table shows the median of each of the sets of data:

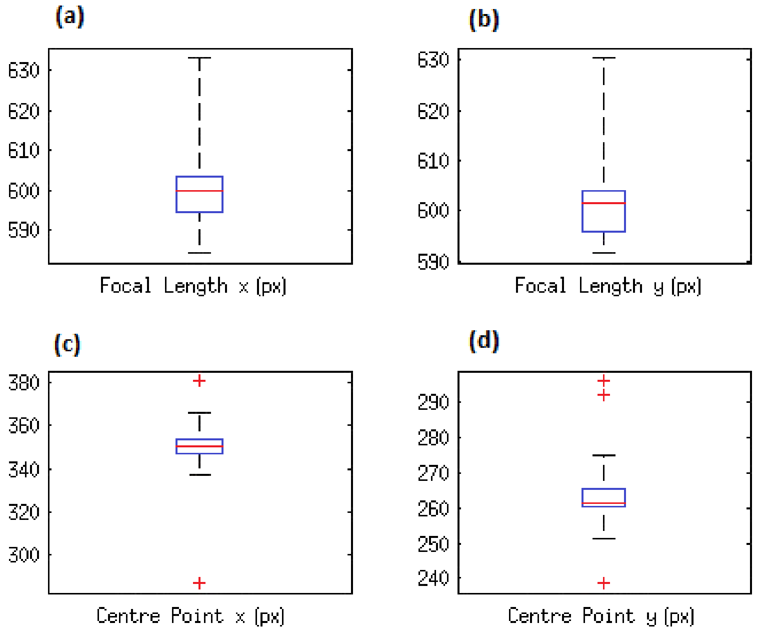

Figure 3.15: Boxplots showing the median and 25th and 75 percentiles of the focal length and centre point for Kinect 2.

Table 3.2: Table showing the parameters found for Kinect 2

Parameter	Value
Focal Length x	599 px
Focal Length y	601 px
Centre Point x	350
Centre Point y	261
Radial Distortion 1	-0.13028
Radial Distortion 2	0.45278
Tangential Distortion 1	0.01189
Tangential Distortion 2	0.0154

Discussion of the parameters found for Kinect 2, and comparison with those for Kinect 1

The top two boxplots in Figure 3.15 show the median and 25th and 75th percentiles of the data collected for the focal length of the second Kinect sensor. The focal length in the x-direction is fairly even about the norm. This can be demonstrated by the fact that the mean and the median are very near — the median is 599.6126 and the mean is 599.4915 (note: these numbers are quoted to 4 decimal places as the data generated is given to 7 decimal places, and reduced to 5 for display in the Appendix. Therefore, 4 decimal places is used here to ensure that the quoted accuracy isn't higher than it should be). The median is also approximately in the centre of the 25th and 75th percentiles of the data. The focal length in the y-direction is also fairly evenly distributed about the norm, as, once again, the median and the mean are very close together. However, for the y-data, there is a slight skew in the data — as is demonstrated by the fact that the median is much closer to the 75th percentile than to the 25th percentile. The median of the y focal length data is 601.5. As for the focal length for Kinect 1, the focal lengths are much higher than expected. The focal lengths in the x- and y-direction are also, once again, slightly different. They are also much higher than those for Kinect 1 — showing that the focal length of different sensors does differ quite substantially. This means that the Kinect sensors are not consistent in their focal lengths. This is probably due to differences in manufacturing.

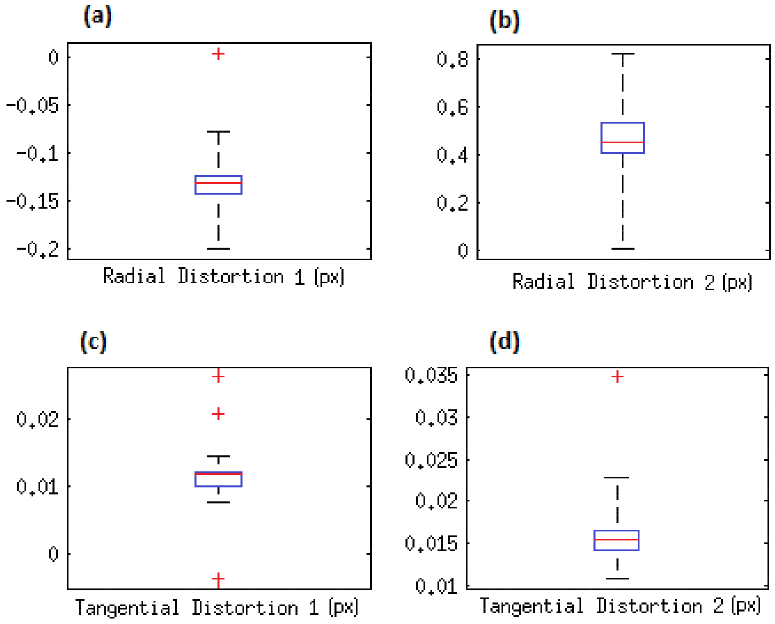

Figure 3.16: Boxplots showing the median and 25th and 75 percentiles of the tangential and radial distortion for Kinect 2.

The bottom two plots in Figure 3.15 show the median and 25th and 75th percentiles of the data collected for the centre point of Kinect 2. The centre point in the x-direction has a median which is slightly higher than the mean. The data appears to be well distributed, as the median line is approximately in the middle of the box representing the 25th and 75th percentiles. The box is also quite narrow. There are two outliers in the data — one above and one below the rest of the data. The median is 350 pixels — which is higher than expected, and also slightly higher than that for Kinect 1. The boxplot for the centre point in the y-direction shows a clear skew in the data — the median is much closer to the 25th percentile than to the 75th percentile. There are 3 outliers in the data — 2 in the region of 290 and one in the region of 240. The mean is slightly higher than the median in this case. The median is 261.5 pixels — still quite high in comparison to the expected value (240), but quite a lot lower than the value for Kinect 1 (273). This shows that the centre point is not constant across Kinect sensors.

The top two plots in Figure 3.16 show the median and 25th and 75th percentiles of the first two radial distortion coefficients for Kinect 2. The first radial distortion coefficient appears to be fairly well distributed — the blue box is fairly narrow, and the median line is approximately in the middle of the box. The mean and the median are also very close — indicating that the data distribution is fairly evenly distributed. The median of the data is -0.13028, which is, once again, small, but not necessarily negligible, and therefore should not be discounted. There is also one outlier which appears to be positive. This was probably caused by a mistake during the calibration procedure, as a positive coefficient means that for that test, the lens distortion changed shape (concave to convex). Once again, the fact that the first radial distortion coefficient is negative indicates and pincushion type distortion. The median is very close to that for the first Kinect sensor — although not the same. For the second radial distortion coefficient, the data are slightly skew — the median is closer to the 25th percentile than the 75th. The box is also quite wide — indicating a wide distribution. The median for the data is 0.45278, which is quite different to the mean for the data (0.44597) — and therefore the data are not evenly distributed. The coefficient is higher than that for the first Kinect. It is also higher than the first radial distortion coefficient. Therefore, this coefficient should also not be discounted.

The bottom two plots in Figure 3.16 shows the median and 25th and 75th percentiles of the first two tangential distortion coefficients of the second Kinect sensor. The boxplot for the first tangential distortion coefficient shows a large skew in the data — the median is very close to the 75th percentile. There are three outliers in the data — two above the median and two below. The box representing the 25th to the 75th percentile is narrow — indicating narrowly distributed data. The median is at 0.01189, which is quite close to the mean for the data. This is very close to that for the first Kinect sensor, and is also very small. The second tangential distortion coefficient is much better distributed — the median is approximately in the centre of the box indicating the 25th and 75th percentiles. There is only one outlier in the data — possibly caused by a mistake during the calibration procedure. The median is 0.0154, which is close to the mean. The value is slightly higher than that for the first tangential distortion coefficient, but is still small enough that both the first and second tangential distortion coefficients can be discarded.

The reprojection error after calibration for each corner in each image is shown in Figure 3.17. This reprojection error is quite low (although it is quite a lot higher than for the first Kinect), and therefore a second calibration optimisation is deemed to be unnecessary.

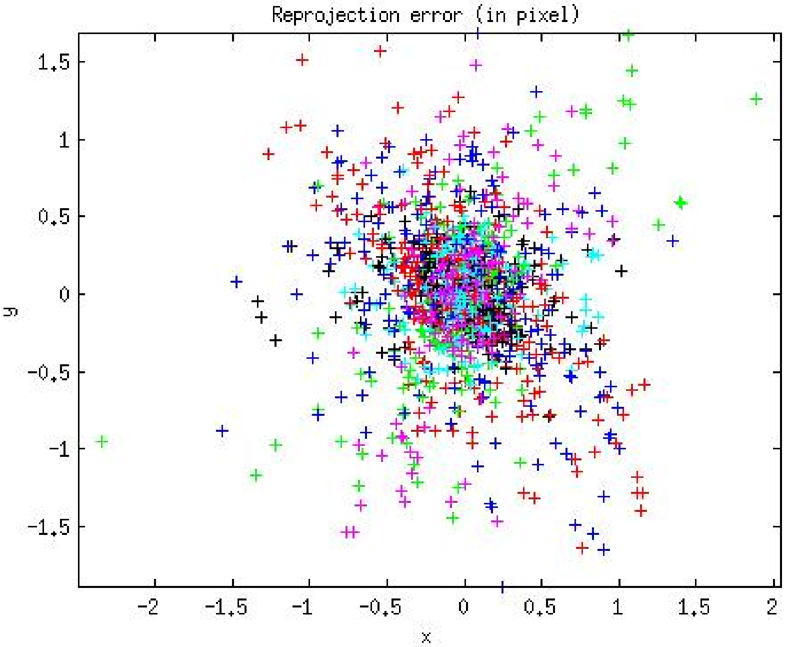

Figure 3.17: Reprojection error after calibration for the second kinect camera. This shows the how much and in what direction the points move after calibration.

Discussion of the effect of the Radial Distortion on the output of the camera

The effect of the lens distortion on the image will now be discussed. The reprojection of the points back onto the image, after applying the calibration parameters is shown in Figure 3.18. The original image points are shown as + signs, and the reprojected points are given by circles. The direction of the change is shown by the arrow. This picture shows that the change in the points is very small — The circles are directly above the + signs. Therefore the effect of the lens distortion is found to be negligible, and can be discounted. This was done for each image in turn, and the effect of the lens distortion was found to be negligible for both Kinects in all the images.

In earlier sections, it was assumed that the 6th distortion coefficient is zero. This is now definitely a valid assumption, as the effect of the lens distortion caused by the first two coefficients has been found to be negligible, and therefore the 6th one can be discounted. The skew can also be discounted for the same reason (the image seems to be well calibrated without including skew in the calibration procedure.)

3.2.3 Verification of the parameters used in the disparity-to-depth mapping

This section will deal with the verification of the values given by Microsoft for the parameters, b, f, and d_{off}, used in the Kinect disparity-to-depth mapping shown in

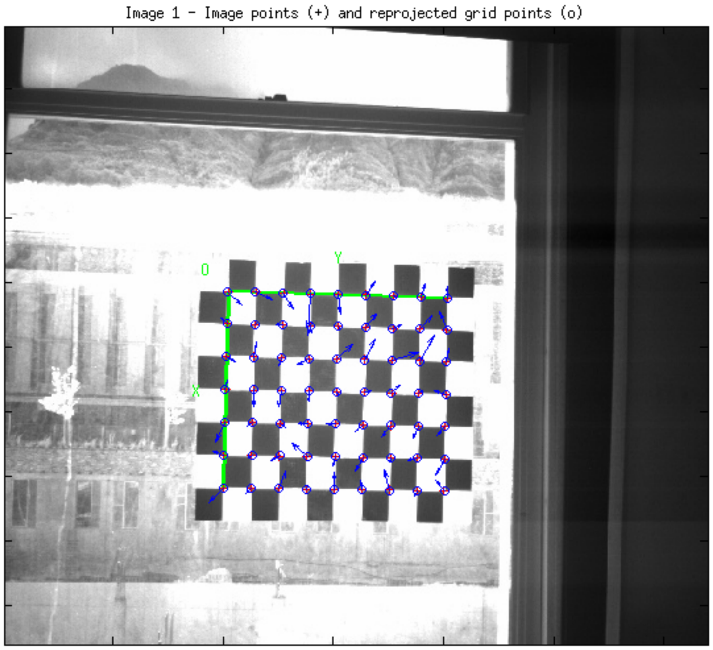

Figure 3.18: Figure showing the effect of the calibration parameters on the image points. The original points are shown as + signs, and the reprojected points are shown as circles. The direction of the change is shown by the arrows.

section 3.1.2. The parameters given by Microsoft as "standard values", and those given by OpenNI as the values they use for their disparity-to-depth mapping, for these parameters are:

Table 3.3: Table showing the baseline, focal length and disparity offset given by OpenNI and Microsoft for the Kinect sensor

Parameter	**Microsoft standard value**	**OpenNI value**
Baseline (b)	7.5 cm	7.5 cm
focal length (f)	580 px	575 px
Disparity offset (d_{off})	1090 px	1090 px

The first value which can be verified is the focal length. In section 3.2.2, the focal length is determined for the two Kinect sensors which were available. The first Kinect had a focal length of 586. This are clearly quite different to the value given by Microsoft. However, in the formula, the baseline b is multiplied by the focal length f, and therefore such a difference in the focal length may not affect the disparity to depth mapping if $b \times f$ is approximately constant (i.e. the baseline changes when the focal length changes).

The baseline and the disparity offset are calculated using an image which is a known distance away, and then using least squares to solve for these two parameters in the following way. First, we need to rearrange formula 3.4 so that it is in the form $y = mx + c$, and m and c involve the unknown variables, b and d_{off}. This is done as follows:

$$\begin{aligned} z &= \frac{b \times f}{\frac{1}{8} \times (d_{off} - d_k)} \\ \frac{1}{8} \times (d_{off} - d_k) \times z &= b \times f \\ d_{off} - d_k &= \frac{8 \times b \times f}{z} \\ d_{off} &= b \times \frac{8 \times f}{z} + d_k \\ d_k &= -b \times \frac{8 \times f}{z} + d_{off} \qquad (3.19) \\ y &= m \times x + c \qquad (3.20) \end{aligned}$$

Where:

$$\begin{aligned} y &= d_k \\ m &= -b \\ x &= \frac{8 \times f}{z} \\ c &= d_{off} \end{aligned}$$

Tests were performed to determine the d_k value for various z values. These tests were performed by taking images with the Kinect pointing at a blank screen. The centre

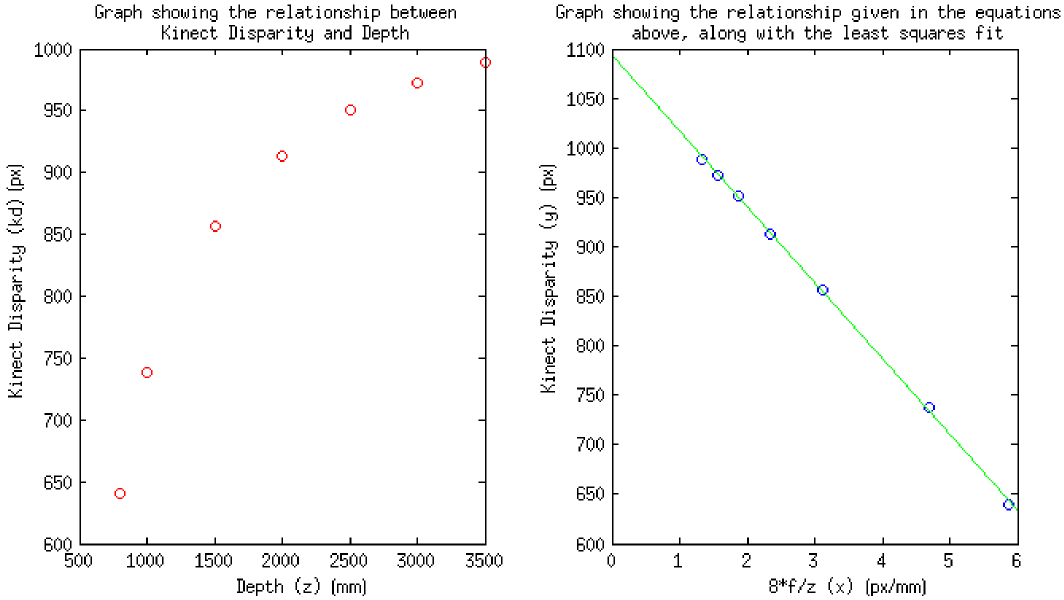

Figure 3.19: Graphs showing the relationship between z and d_k (on the left), as well as the relationship between x and y (blue dots in the right-hand image), and the least squares interpretation of them (the green line fitted to the blue dots)

point of the image (347, 273) pixels — as found in the previous section, will be used to determine the disparity value for seven z values — 800 mm, 1000 mm, 1500 mm, 2000 mm, 2500 mm, 3000 mm and 3500 mm. These z and d_k values are given in the table below.

Table 3.4: Table showing the values for z and the corresponding d_k value

z [mm]	d_k [px]
800	640
1000	738
1500	857
2000	913
2500	951
3000	972
3500	989

These values for z and d_k are now used in the least squares problem described in formula 3.19. The focal length used for this is 586 pixels (note that Kinect 1 was used for this). These values are plotted on a graph in Figure 3.19. There is a second graph which shows the values for x and y as determined by the formula above, along with the least squares solution to the problem. It is clear from the first graph that as the depth increases, so does the Kinect disparity. This is expected, because as depth (z) gets larger, $b \times 8 \times f/z$ gets smaller (for all other factors held constant). This number will always be smaller than the disparity offset d_{off} (due to the derivation of the formula). This means that as the "x" term becomes smaller, the "y" term gets larger, so this makes sense. The second graph shows that the least squares fit to the values is very good, as the data are clearly linear (as expected). The values that are obtained from the least squares operation for d_{off} and b are 1093 and 76.6996 respectively. This mean that the values given for baseline and the disparity offset by Microsoft (and OpenNI) are slightly different — the effect of this will be determined in the next chapter.

Determination of the effect of inaccuracy of b and d_{off} on the d_k to z mapping

The effect of the errors in the disparity offset and baseline will now be investigated. The formula shows that if $b \times f$ is constant, the mapping from d_k to depth will not be affected. However, a change in d_{off} will change the mapping, potentially in a significant way. For this section, the value of the focal length will be changed, and the value for the disparity offset will be calculated for all other factors constant. The factor $b \times f$ will then be compared to the values given by Microsoft and OpenNI for these values. Then the effect of the change in the focal length on the mapping from d_k to z will be determined by finding the value for z for various different values of d_k for each of the different values determined for f, b and d_{off}, to determine what difference it will make to the depth calculation if the Microsoft standard values, or the OpenNI values are incorrect.

First, the effect of the focal length on the baseline b and disparity offset d_{off} will

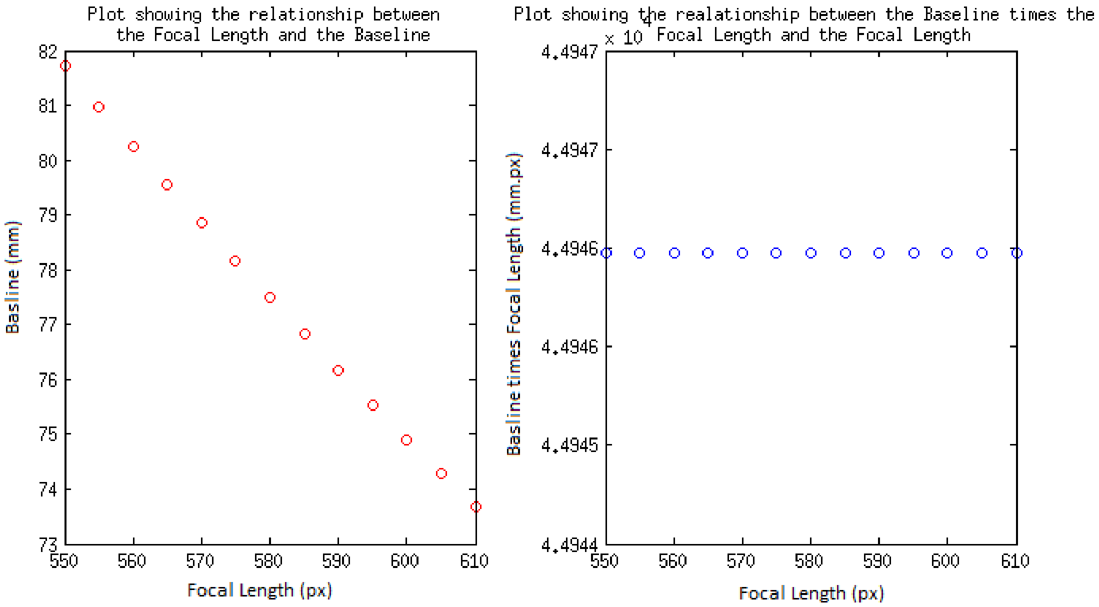

Figure 3.20: Graphs showing the relationship between f and b, as well as the relationship between $f \times b$ and f

be determined. To do this, All the z and d_k values in the above table will be used. The focal length will be varied from 550 to 610 pixels in 5 pixel increments. The baseline and the disparity offset are then determined for each of the focal lengths. The disparity offset is found to be constant at 1093 pixels. The baseline, and baseline multiplied by focal length are then plotted against the focal length. These plots are shown in Figure 3.20.

The focal length times the Baseline is clearly constant over the range that is being considered. This means that a change in the focal length should not change the output of the disparity-to-depth mapping, as long as the *baseline changes proportionally* to the focal length. The product of the baseline and the focal length is 44946 for our values. Microsoft give this value to be 43500, and OpenNI give this value to be 43125. The percentage difference can now be calculated using the following formula:

$$\%difference = \frac{f_1 - f_2}{f_2} \times 100 \tag{3.21}$$

The Microsoft value (f_1) has -3.2% difference to our calculated value (f_2), whilst the difference between the Microsoft (f_1) and OpenNI (f_2) values is only 0.86%.

To determine the effect of the difference in these factors, various values of Kinect Disparity will be used. The corresponding depth for each of the values for $b \times f$ and d_{off}

will be calculated. These values will then be compared to determine the effect of the change in the parameters on the mapping. These outcome of these tests are shown in Appendix A. The $b \times f$ and d_{off} pairs for each of the pairs are:

Table 3.5: Table showing the differences between the values of $b \times f$ and d_{off} for the Microsoft, OpenNI and the values calculated in this thesis

Parameter	Calculated	Microsoft	OpenNI
$b \times f$	44946	43500	43125
d_{off}	1093	1090	1090

The data in the tables in Appendix A are represented on the graph in Figure 3.21. The red graph represents the depths obtained from the Calculated values, minus the depth obtained using the Microsoft values. The blue line represents the values using the Calculated values, minus the values obtained using the OpenNI values.

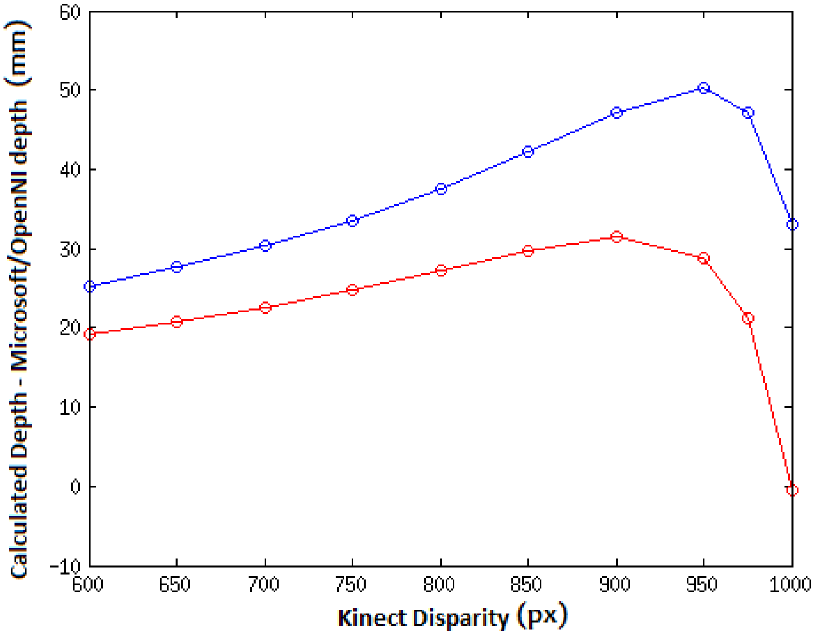

Figure 3.21: Graph showing the difference between the depths obtained using the calculated value, and those obtained using the Microsoft and OpenNI values

The figure shows that up until $d_k = 975$, the value obtained using the calculated values is higher than that obtained using the Microsoft and OpenNI values. This is expected. At these values, the small change in the value of d_{off} has little effect on the disparity to depth calculation. However, at 975, the difference between the calculated values and the Microsoft and OpenNI values drops rapidly. This is due to the difference in the value of the disparity offset. The Microsoft and OpenNI depths become greater than those obtained using the values calculated. This is due to the influence of d_{off}. As the disparity values approach d_{off}, the effect of the change in d_{off} will become much more noticeable, as the *percentage* difference of $d_{off} - d_k$ becomes greater (although there will still only be an absolute difference of 3 between the calculated and OpenNI values). This causes the great discrepancy between the values after the disparity becomes greater the 975.

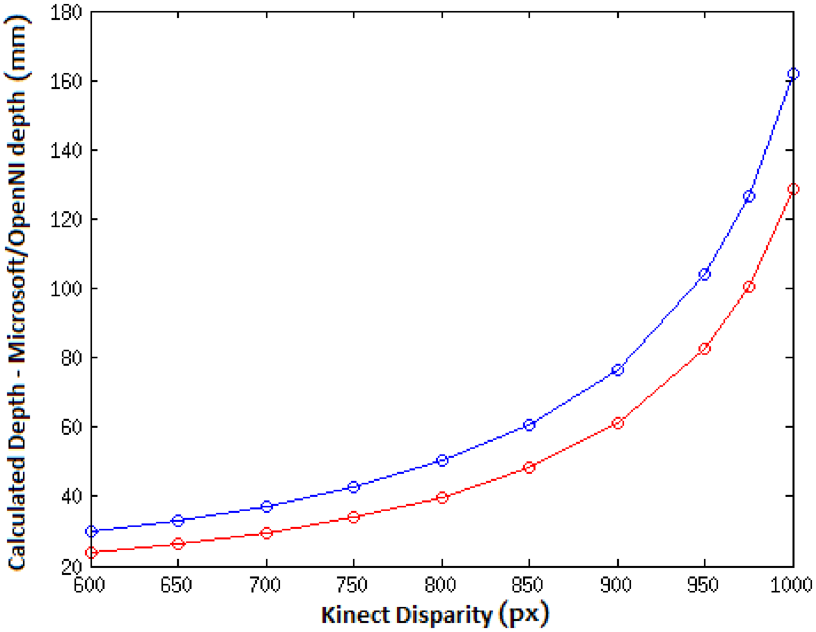

Figure 3.22: Graph showing the difference between the depths obtained using the calculated value of $b \times f$ — with $d_{off} = 1090$, and those obtained using the Microsoft and OpenNI values for $b \times f$ and d_{off}

Figure 3.22 shows the difference if we let $d_{off} = 1090$ for the calculated values of $b \times f$, as well as for the Microsoft and OpenNI values. This graph shows clearly that the cause of the steep drop at $d_k = 975$ in Figure 3.21 is caused by the small change in d_{off}, as there is *no* drop when the value of d_{off} is constant for all of the different values of $b \times f$. This experiment shows that the value of d_{off} is extremely important for large depths (over 2500mm), as it begins to affect the values quite a lot at around 3000mm ($d_k = 975$). The value of z is affected before $d_k = 975$, however the change is not large (within the region of 30 mm).

It is also clear from Figure 3.22 that the value of $b \times f$ affects the disparity-to-depth mapping a lot at high d_k (within the region of 18 cm). This difference is plotted as a percentage of the depth measurement calculated (Note: this is done holding d_{off} constant at 1090). Figure 3.23 shows the result of this test.

Figure 3.23 shows that the *percentage* difference between the calculated values for $b \times f$ and those given by Microsoft and OpenNI is *constant*. The percentage difference for

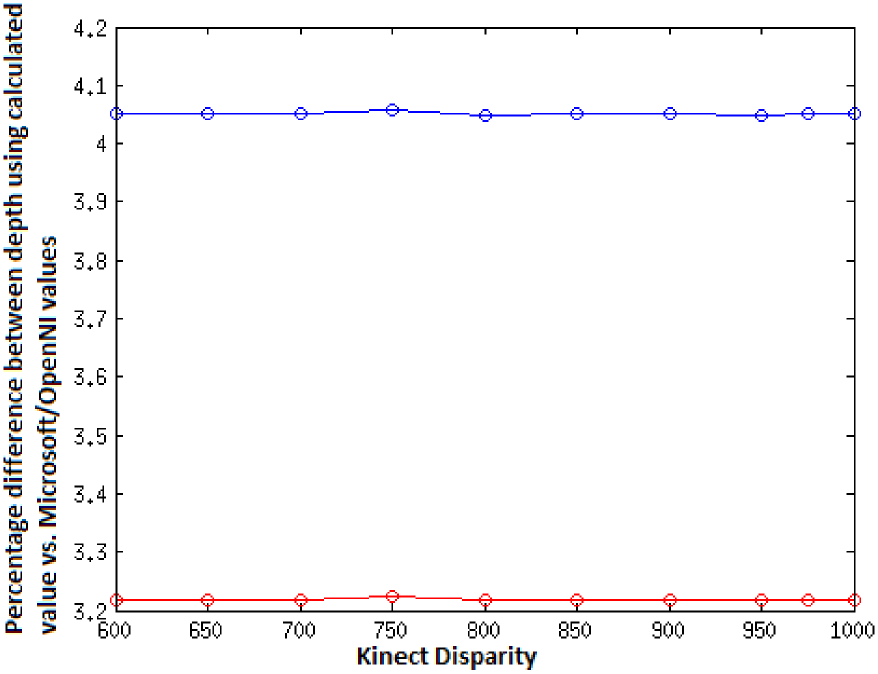

Figure 3.23: Graph showing the percentage difference between the depths obtained using the calculated value of $b \times f$ — with $d_{off} = 1090$, and those obtained using the Microsoft and OpenNI values for $b \times f$ and d_{off}

our values compared to the Microsoft values is approximately 3.2%, and the difference between the calculated values and the OpenNI values is approximately 4%. The percentage difference in the calculated value and the Microsoft value is approximately 3.22%, and the percentage difference between the calculated value and the OpenNI value is approximately 4.05%. These two data points indicate that a certain percentage increase in $b \times f$ corresponds to the same percentage increase in z. This will now be tested more rigorously by using more data points. The value of $b \times f$ will be increased and decreased by 2%, 4%, 6%, 8% and 10% from our calculated value, for 5 different d_k values (600, 700, 800, 900, 1000). Depths will then be determined for each of these values, and the percentage difference calculated. The results of this test are shown in Figure 3.24. Figure 3.24 clearly shows that a percentage difference in $b \times f$ causes an equivalent percentage difference in z. This is as expected from previous experiments, as well as from the formaula above ($b \times f$ is the factor by which everything else is multiplied to get z).

In conclusion, the value of the focal length is important, as it changes the value of d_{off}. It also changes the value of $b \times f$ from the values given by Microsoft and OpenNI. The value of d_{off} has a significant impact on the disparity-to-depth mapping for depths greater than 2500 mm. The value of $b \times f$ also affects the value of z — The percentage by which the value for $b \times f$ is different from the actual value causes an equivalent

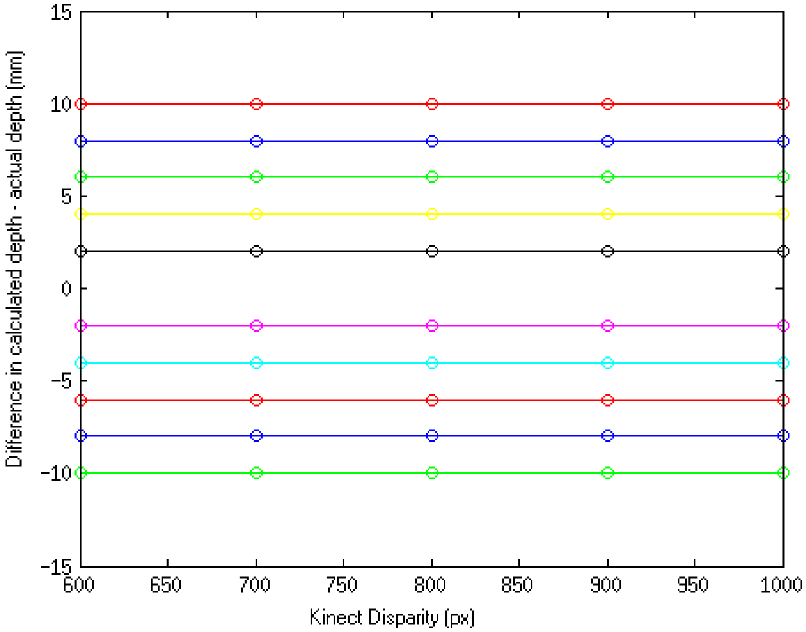

Figure 3.24: Graph showing the percentage difference between the depths using different values of $b \times f$. Each of the colours represents a different value of $b \times f$

percentage difference in the value of z. This means that each Kinect sensor should be calibrated separately as there is a large technical variation between the devices. Uncalibrated devices will results in large errors in depth data.

3.3 Reaction of the Kinect to different situations

This section discusses how the Kinect sensor deals with different types of surfaces. First, the Kinect will be tested using a black screen and a white screen, to determine if there are any more errors for either screen. The same tests will then be used to determine how the Kinect reacts to reflective and non-reflective surfaces under different lighting conditions.

3.3.1 Comparison of Black and White Surfaces

A screen was covered in white fabric and then in black fabric, and then the Kinect was used to take images of the screen from different distances. A picture of the equipment setup is shown in Figure 3.25.

The Kinect was placed at seven different distances from the screen (800 mm, 1000 mm, 1500 mm, 2000 mm, 2500 mm, 3000 mm, 3500 mm). The maximum and minimum distances for each image are then calculated. The exact distance to the screen is

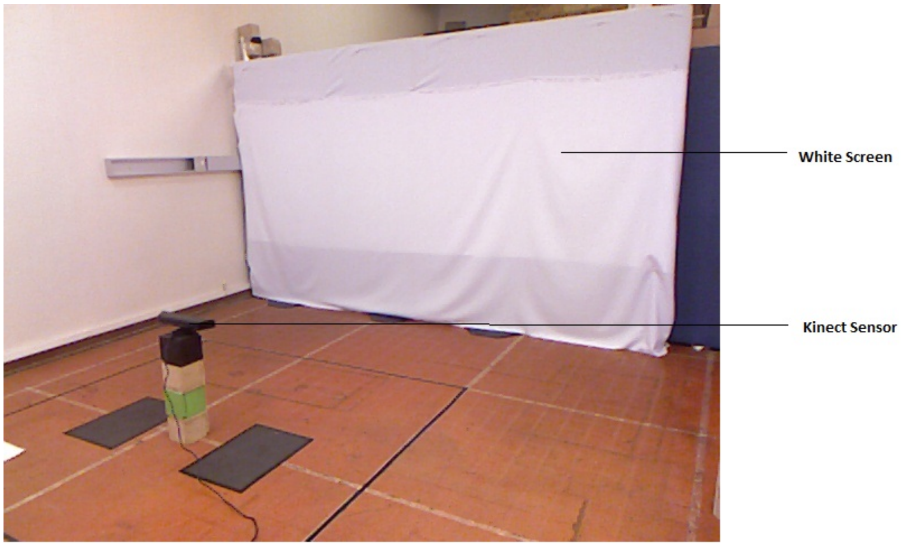

Figure 3.25: Picture of the test setup for the black and white tests.

known, and therefore the difference between the maximum and minimum of the black screen and the white screen will be determined. The data for this chapter are given in Appendix A. Note that because the screen is slightly too small, all the images had to be truncated to ensure that only the values for the pixels that are actually returns from the screen are used. To do this, the image of the screen from the furthest distance is used, and the rows and columns which are actually the image of the screen are determined. Only these pixels are used in the calculation of the maximum and minimum. Also note that the OpenNI values for depth are used in this section. This is for the sake of simplicity.

The results of this experiment are plotted in Figure 3.26 and Figure 3.27. The values that are plotted are the difference between the maximum value and the actual value (Figure 3.26), and the difference between the actual value and the minimum value (Figure 3.27).

In the plots of Figures 3.26 and 3.27, the blue and red lines represent the values for the black screen, and the green and cyan lines represent the value for the white screen. From Figure 3.26, the maximum distances for the Black screen are far more regular — They seem to increase in a parabolic manner with the total distance from the screen. This is true of both sets of measurements. However, the difference between the maximum and the actual distance is much higher for larger distances with a black screen than with a white screen.

Figure 3.27 show that the values for the difference between the actual value and the minimum value are much less regular than the value for the maximum minus the actual value, for the black screen. However, the white screen is slightly improved, although

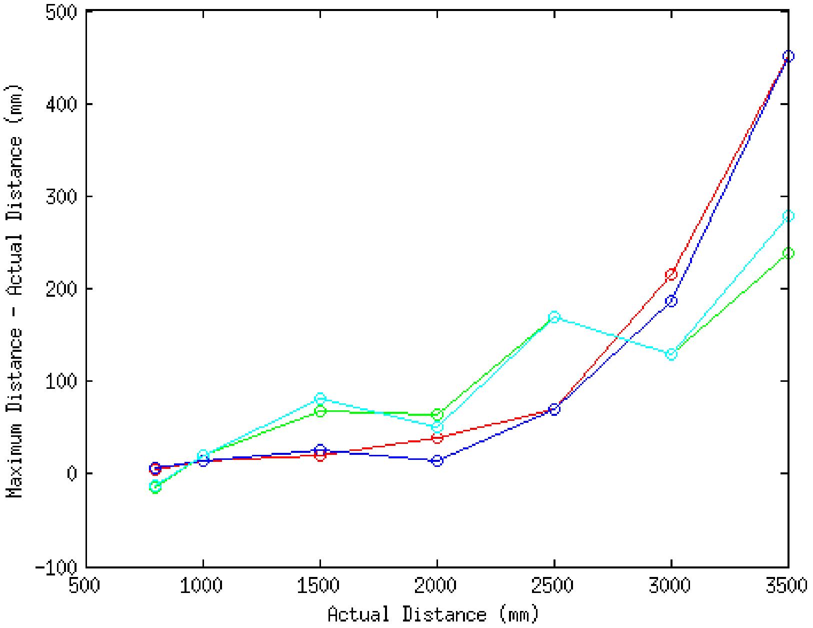

Figure 3.26: Plot of the Maximum Distance minus the Actual Distance. The red and blue lines represent this value for the black screen, and the green and cyan lines for the white screen. There are two sets of data for each screen, and these are represented by the different colours for each screen (i.e. blue represents measurement 1 for the black screen)

the value for the black screen still appear to be more regular than those for the white screen. The values are also much less varied. For the maximum values, there was a large difference between the values for the black screen and those for the white screen. For the minimum values, this is not so, the values for the white screen are lower, but not drastically.

This test shows that the Kinect does not seem to react differently to black and white surfaces under the same lighting conditions. Therefore, this does not need to be considered when using the Kinect sensor for depth measurements.

3.3.2 Reaction of the Kinect to different lighting conditions with Reflective and Non-reflective surfaces

For this section, a very similar test setup is used to the one used in Section 3.3.1. The screen was covered in aluminium foil. The lights in the room were turned on, and the windows were opened. The Kinect was then placed at various distances (800 mm, 900 mm, 1000 mm, 1200 mm, 1400 mm, 1600 mm, 1800 mm, 2000 mm, 2200 mm, 2400mm, 2600 mm, 2800 mm, 3000 mm, 3200 mm, 3400 mm, 3600 mm). The number of ”error points” (i.e. points which give a depth of 0, due to the Kinect not being able to calculate the distance) is calculated, and then these are compared. Note that because the

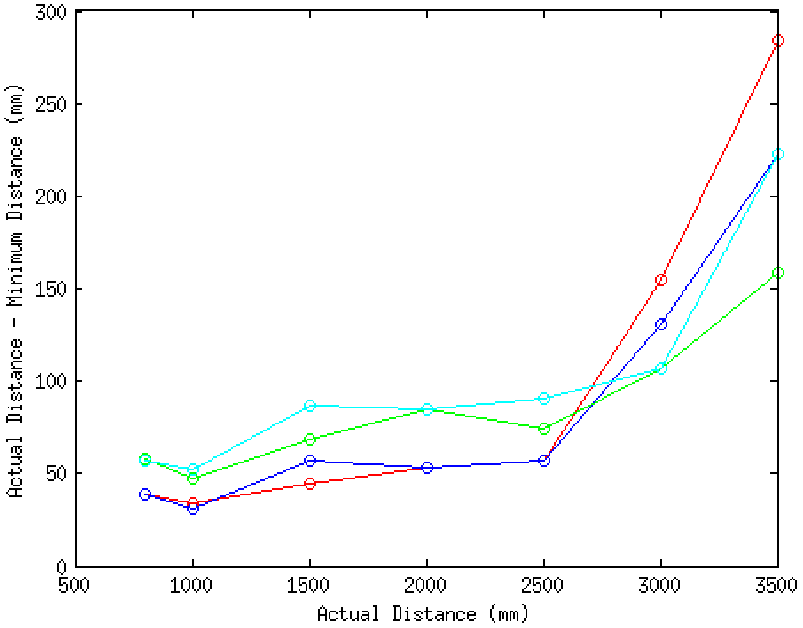

Figure 3.27: Plot of the Actual Distance minus the Minimum Distance. The red and blue lines represent this value for the black screen, and the green and cyan lines for the white screen

screen is slightly too small, all the images had to be truncated to ensure that only the values for the pixels that are actually returns from the screen are used. To do this, the image of the screen from the furthest distance is used, and the rows and columns which are actually the image of the screen are determined. Only these pixels are used in the calculation of the number of errors. The same thing is done with the screen covered in white paper (to simulate a non-reflective surface). The number of errors for each distance for the non-reflective and reflective surface is recorded in Appendix A.

The number of errors for the non-reflective screen is consistently zero. There is the occasional exception, however, the numbers are very low in comparison to the total number of pixels in the image. These are therefore discounted. The errors for the reflective surface, however, are large. The errors for each of the 4 sets of measurements (2 sets for high light and 2 sets for low light) are plotted in Figure 3.28. The red graph represents the first set of high light measurements, and the blue graph the second set. The green and cyan lines represent the first and second low light measurements, respectively.

The graph shows that the number of error points in the data increases approximately linearly with the distance from the screen (for distances between 1500 and 3000 mm). It seems to ”level out” at low distances (800 mm to 1200 mm) and at high distances (3000 mm - 3600 mm), however at mid-distances it is fairly linear. The graphs of the low light conditions appears slightly more linear than that for high light conditions. The

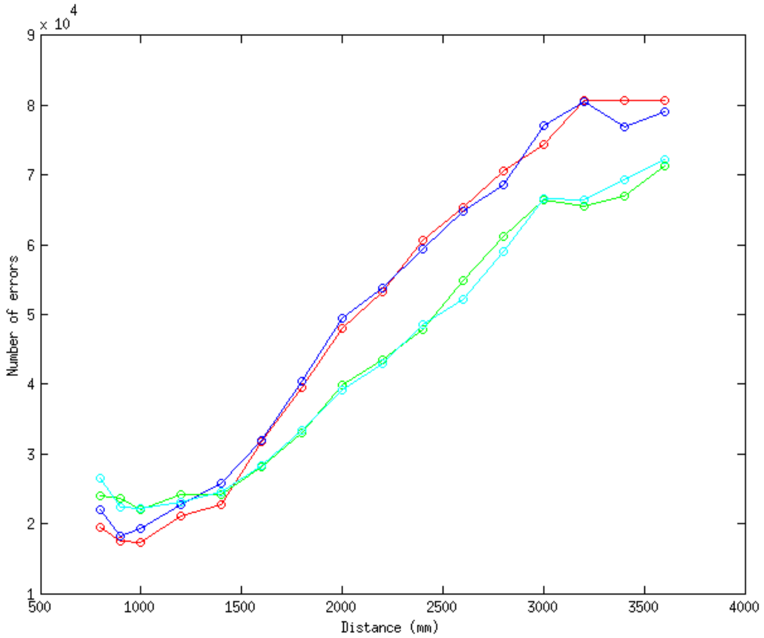

Figure 3.28: Plot of the number of errors vs. the distance from the screen for the reflective surfaces. This shows that the number of errors increases as the distance increases. The red and blue lines represent the two sets of data in high-light conditions, whilst the green and cyan lines represent the data for low-light conditions.

low light conditions also appears to result in fewer errors at higher distances (within the order of 10 000 errors). However, at low distances, the number of errors is much closer to that for the high light condition.

The data collected in this section clearly shows that the Kinect reacts rather badly to a reflective surface, as there are many more error points when a reflective surface is placed in front of the sensor as opposed to a non-reflective surface. There are many errors for reflective surfaces, whilst the number of errors for the non-reflective surface is consistently zero, regardless of the distance from the target or the lighting conditions. The number of errors for the reflective surface increases with the distance from the target, and is also slightly worse under high light conditions compared to low light conditions. This indicates that the Kinect should not be used in environments where there are lots of reflective surfaces, however the lighting condition does not affect the operation of the Kinect sensor.

Chapter 4

Development of Feature Extraction Algorithms

Feature extraction is very important in mobile robotics, especially for SLAM. Mobile robots generally have limited storage capacity, and therefore storing huge amounts of data is impractical. In most common SLAM algorithms, (for example EKF-SLAM and FastSLAM), the computational complexity also increases drastically with the number of features [79, 77]. For EKF-SLAM, the computational complexity is $O(n^3)$ [79] where n is the number of landmarks. For FastSLAM, the computational complexity is $O(M \log K)$, where M is the number of particles in the particle filter and K is the number of landmarks [77]. This chapter investigates algorithms that can be used for line extraction on data collected using a Hokuyo Laser Range Finder. The data obtained using the Hokuyo Laser Range finder are of a single box in a large room (so that the walls are far enough away that they don't affect the image). The settings for the range finder are such that one measurement is taken per degree (so 240 points are output). Most of the images, however, only have 22 - 68 data points. This is due to the fact that the box is placed in various places around the range finder, and therefore the entire 240 degree range is not used. These datasets are used for the experiments in most of the experiments shown in this chapter (with the exception of the few places where other data is described). This is done so that the data is consistent.

A laser range finder is used as the primary input sensor in mobile robotic navigation for a number of reasons. The first reason is that it outputs a 2-dimensional point-cloud which is much simpler than the output of the Kinect sensor. This 2D point cloud, although simple, provides a good basis for understanding of how similar algorithms work on point cloud data. Secondly, the number of points output is much smaller than for the Kinect sensor, and therefore the algorithms will run much faster.

Initially, two algorithms are tested in two dimensions for use on Laser Range Finder Data for line extraction. These algorithms are a split-and-merge type algorithm and a 2-dimensional Hough transform. These two algorithms are then used in a series of tests designed to allow a comparison of their effectiveness. This is used to determine whether the Hough transform performs well in 2-dimensions before extending it into 3-dimensions.

This chapter comprises of two main parts. The first part is the theory of the two feature extraction algorithms. The two algorithms are then tested individually for processing time, and to determine the best values for the parameters that are required for the algorithm (for the split-and-merge algorithm this parameter is the threshold, and for the Hough transform, these parameters are the discretisation of r (the depth) and the discretisation of the angle, θ). The two algorithms are also tested for the effect of the number of lines fitted to the data on the speed. Finally, the two algorithms are compared based on the results presented.

4.1 The Algorithms

For this section, the data are assumed to be in the form of a set of (x, y) points, where the sensor is at point $(0, 0)$.

4.1.1 Split-and-Merge

The Split-and-Merge Algorithm as presented below is very similar to the algorithm presented by Nguyen *et al.* in [32]. This algorithm takes as its input a 2D point cloud, containing straight lines. The algorithm outputs the (m, c) parameters of each lines found in the point cloud. The algorithm determines the placement of the lines by running through the point set and fitting lines to the set. A threshold is then used to determine whether the line fitted is a "good" or a "bad" fit to the data. If the line is considered to be a "bad" fit, the point cloud is split (see step 4). This process is repeated until all the lines fit the data correctly. The lines are then checked to see if any of them are co-linear. These lines are then merged. A step-by-step algorithm is presented below:

Step 1: A set s_1 is initialised as follows:

$$\mathbf{s_1} = (\mathbf{x}, \mathbf{y}) \tag{4.1}$$

where $\mathbf{s_1}$ is a set containing all of the points in the data, $\mathbf{x}$ is a vector containing all of the x-values of the points, and $\mathbf{y}$ is a vector containing all of the y-values in the set.

Step 2: A line $y = mx + c$ is then created by joining the first and last points of the set together. This is done as follows (assuming that the set contains a points):
1. $p_1 = (x_1, y_1) = (\mathbf{x}(1), \mathbf{y}(1))$
2. $p_2 = (x_2, y_2) = (\mathbf{x}(a), \mathbf{y}(a))$
3. $m_1 = (y_2 - y_1)/(x_2 - x_1)$
4. $c_1 = y_1 - m_1 x_1$

This gives a line defined by $y = m_1 x + c_1$.

Step 3: The orthogonal distance between each point $p_i = (x_i, y_i)$, for i between 1 and a in set $\mathbf{s}$ and the line defined by m_1 and c_1 is found. This is done using the following algorithm:

```
1. $p_3 = (x(i), y(i)) = (x_3, y_3)$.
2. $p_4 = (0, c_1) = (x_4, y_4)$.
3. $m_2 = (y_4 - y_3)/(x_4 - x_3)$.
4. $\theta_1 = \arctan(m_1)$.
5. $\theta_2 = \arctan(m_2)$.
6. if $\theta_1 > \theta_2$
$\longrightarrow \theta = \theta_1 - \theta_2$.
if $\theta_2 > \theta_1$
$\longrightarrow \theta = \theta_2 - \theta_1$
if $\theta_2 = \theta_1$
$\longrightarrow$ The point is on the line
$\longrightarrow \theta = 0$.
7. $d_s = \sqrt{((x_4 - x_3)^2) + ((y_4 - y_3)^2)}$.
8. Orthogonal Distance $d = d_s \times \sin(\theta)$,
```

where m_2 is the gradient of the line connecting the x-intercept of the line found in Step 2, and θ_1 is the angle of the line found in Step 2 and θ_2 is the angle of the line with gradient m_2. Number 6 determines which line has a larger gradient, and therefore how to determine θ. In figure 4.1, the first two cases are shown.

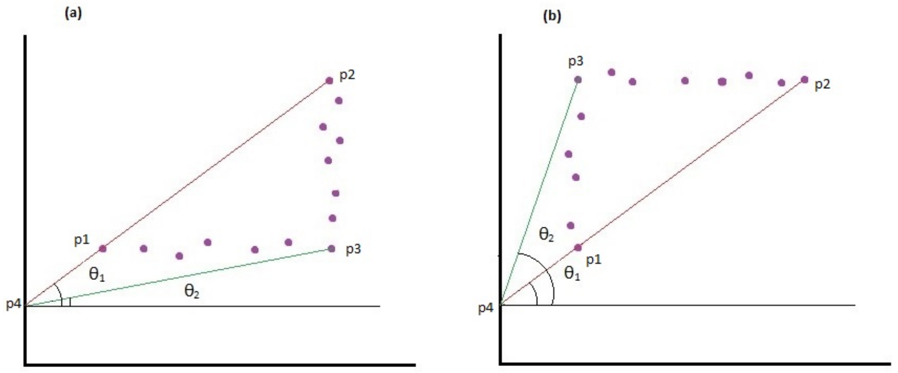

Figure 4.1: Figure showing two of the cases for Number 6 - (a) shows when $\theta_1 > \theta_2$ and (b) shows when $\theta_2 > \theta_1$. (a) shows that when $\theta_1 > \theta_2$, θ_2 must be subtracted from θ_1 in order to obtain the correct angle. The opposite is true for when $\theta_2 > \theta_1$

Step 4: Determine whether the maximum orthogonal distance is greater than some threshold. If it is, the set $\mathbf{s}$ is split at the point with the maximum orthogonal distance from the line. This creates two new sets of points, $\mathbf{s_1}$ and $\mathbf{s_2}$.

Step 5: If the result of Step 4 is two new sets, repeat Steps 2 - 4. If the set is not split in Step 4, end the program and return m_1 and c_1. Note: for the repeats of Step 2, a least-squares fit is used on the points, rather than using the method of joining the first and last points in the set.

Step 6: The lines are then checked to see if they are co-linear (or near-colinear, within some range of m and c). These lines are then merged. Note that this step is regularly unnecessary for the types of data that are created using a Laser Range finder. This is discussed more in a later section.

It is important to note that, in this implementation of the algorithm, only angles which are not vertical will be able to be found. This is because when the angle is vertical, the gradient is infinite (or undefined) - and therefore not computable. This is not considered to be a problem, and true vertical lines are very unlikely to occur in real data.

4.1.2 2-Dimensional Hough Transform

The Hough transform is the second algorithm used for comparison. It uses an alternative method to find lines - a voting method, rather than a least-squares method.

The original algorithm by Hough [34] is as follows. Consider any point in the xy-plane, (x_i, y_i). The general equation for any line through this point is given by $y_i = mx_i + c$ for any values of m and c. If this equation is written in terms of m and c, we get $c = -x_i m + y_i$, which is a similar form. If this equation is plotted in the mc-plane, it is a single line for any (x_i, y_i) pair. Now consider a second point (x_j, y_j) in the xy-plane. This point will also correspond to a line in mc-space (also known as parameter-space, or Hough-space). The point where these two lines intersect in mc-space is the (m, c) value of the line that passes through both point (x_i, y_i) and point (x_j, y_j) in xy-space. In fact, every point on that line will pass through that specific point (m, c) in Hough-space.

Conceptually, every point (x, y) could be transformed in this way, and then plotted in Hough-space The principle line in xy-space could be determined by the intersections of these lines. However, there is a practical problem with this approach. As the line connecting the points tends towards the vertical, the magnitude of the gradient (m) of the line tends towards infinity. One way around this problem (presented by Duda and Hart [35]) is to use the polar representation of a line:

$$x \cos\theta + y \sin\theta = \rho \tag{4.2}$$

Figure 4.2 shows the geometrical interpretation of the parameters ρ (r) and θ in the xy-plane. It is important to note that a horizontal line has $\theta = 90^o$ and ρ equal to the

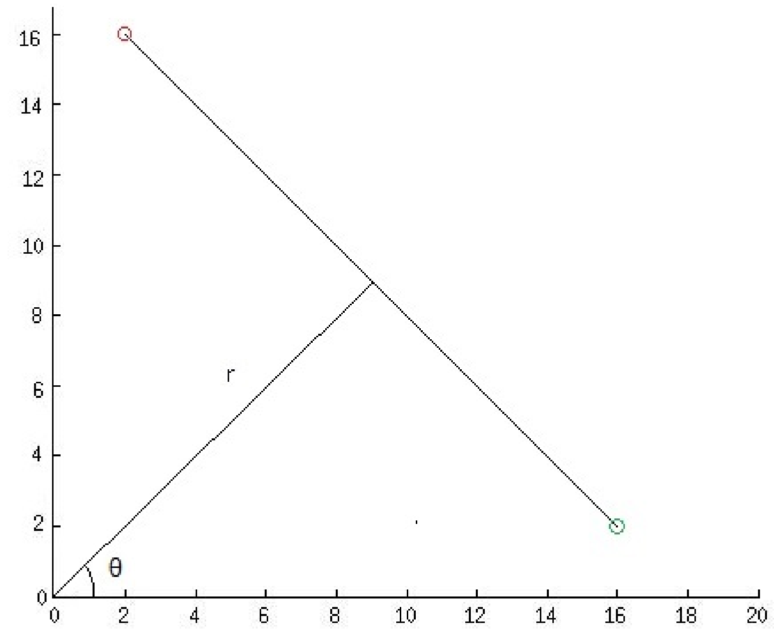

Figure 4.2: Geometric interpretation of the parameters r (ρ) and θ of a line. The green and red dots represent two points in the $x - y$ plane. The line that passes between them is shown. This line can be represented by θ and r.

positive y intercept. A vertical line has $\theta = 0^o$ and ρ (r) equal to the positive x intercept. This parameterisation converts each point into a sinusoid in parameter space. Each of these sinusoids represents the family of lines that pass through a particular point (x_k, y_k) in the xy-plane. The intersection of the sinusoids represents the (r, θ) value of the line that passes through both of the points.

Once we have three or more points in xy-space, we require a voting scheme because the points may not necessarily be on the same line. We, therefore, form a grid structure in the Hough-space. Each cell in the grid is "voted for" by the sinusoids that pass through that cell. In this way, any number of points can be fit to a line in xy-space. The choice of the grid number is very important for this. The size of the grid determines the accuracy of the co-linearity between the points. However, due to the inaccuracies in the measurement from either the Kinect or the laser range finder, making the grid number too small will result in the Hough transform not being effective for extracting lines from the data.

From the theory described above, this is the algorithm that is used to perform the Hough transform:

Step 1: A set $\mathbf{s} = (\mathbf{x}, \mathbf{y})$ is initialised in the same way as the set was initialised for the Split-and-Merge algorithm.

Step 2: All of the points in the set $\mathbf{s}$, (x_i, y_i) are transformed into sine waves in the Hough Space described by (r, θ) (where θ ranges from $1 : 180$ degrees) through the following formula:

$$r = x_i \cos(\theta) + y_i \sin(\theta) \tag{4.3}$$

In the implementation used for this project, the value of the formula is evaluated at discrete intervals of 1 degree. This is because the code uses a for loop to determine all the values of the formula at intervals of 1 degree. This is the most efficient way of implementing the code. The size of the interval could be changed if deemed necessary.

Step 3: A grid is then initialised. For this implementation, the grid only needs to be initialised in the r direction, as the values for θ are discrete (in the code - this is discussed in a later section). The grid in the r direction is initialised as follows:
1. The maximum ($max(r)$) and minimum ($min(r)$) values of r are determined
2. The range of r values is then determined by $range(r) = max(r) - min(r)$
3. The required number of grid lines ($gridno$) is then chosen (this is how the grid number is changed). The range of r is then divided by this number to get the borderlines of the grid. $gridnumber = range(r)/gridno$
4. Each of the borderlines is then determined (the first line will be $min(r)$, the second $min(r) + gridnumber$, etc. up until the last line which will be $max(r)$. These values define the lines in the grid which will be used for the voting scheme.

Step 4: The voting matrix is filled. Each row represents one of the ranges of r (i.e. the first line in the matrix represents r between $min(r)$ and $min(r) + gridnumber$). Each column in the matrix represents a single value of θ between 1 and 180 degrees (because θ is discrete). The voting matrix is filled by determining which “blocks” each of the sine wave found in Step 2 pass through. This is done by taking each of the values (for each value of θ), and determining which r range it fits in. This is repeated for every sine wave. In this implementation, the voting matrix is actually a 3-dimensional matrix - the x-direction represents r, the y represents θ, and the z-direction represents each sine wave. Each block (r, θ, z_i) is either a 1, if the sine wave passes though the grid block r, θ, or otherwise it is a 0.

Step 5: In this step, the maximum number of votes for each grid block (r, θ) is determined by adding all of the ”ones” in the matrix. The lines which pass through that grid block are then removed from the matrix, and this process is repeated until there are no ones left in the matrix. The line parameters are saved for the relevant grid block.

The above process describes the method used in the code given in the Appendix for the Hough transform in 2-dimensions. This algorithm is tested fully in the next section.

4.2 Tests of the Split-and-Merge Algorithm

In this section, the first parameter to be determined is the optimal threshold for the split-and-merge algorithm. The effect of the threshold on the efficacy of the algorithm, for data at different distances is be determined. The algorithm will also be tested to determine how well lines are fitted to the data.

4.2.1 Determination of the optimal threshold for the Split-and-Merge algorithm

The aim of this test is to determine the best threshold to use for the type of data that are output from the laser range finder. The threshold for these specific data should be found, and the effect of this threshold on the outcome of the split-and-merge algorithm determined.

It is expected that a threshold which is too small will result in too many lines being fitted to the data. This is because the threshold determines whether the furthest point from the fitted line is "too far" from the line, and therefore whether the dataset should be split. If the threshold is too small, this threshold will be exceeded too often, and therefore the data will be split more times than it should be. This is illustrated in Figure 4.3.

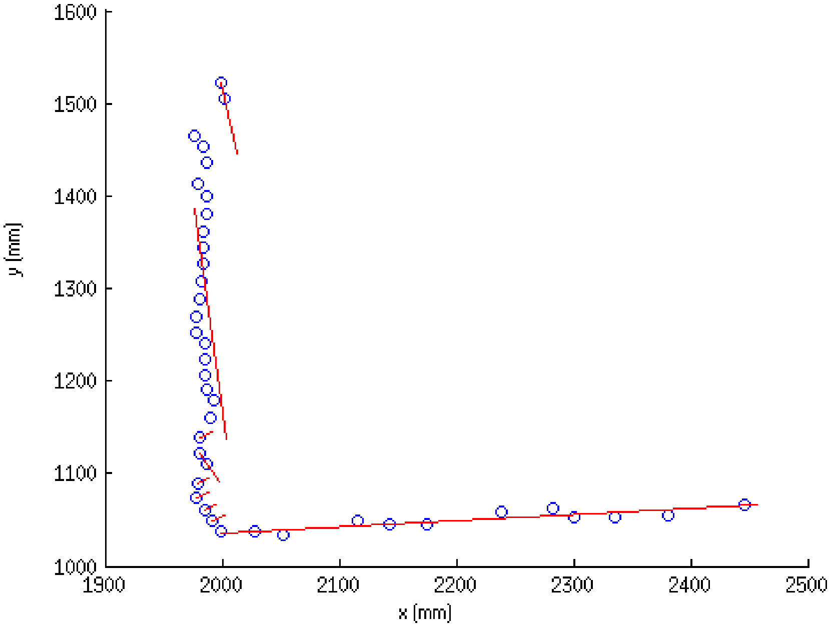

Figure 4.3: Figure showing the case where the threshold for the split and merge algorithm is too small, and therefore the data are split more times than they should be by the algorithm

On the other hand, a threshold which is too high will cause too few lines to be fitted to the data. This is because the threshold is never exceeded (or exceeded too few times) for the data, and therefore the dataset is never split. This is illustrated in Figure 4.4. A correct threshold will result in the correct number of lines being fitted to the data. This is also shown in Figure 4.4.

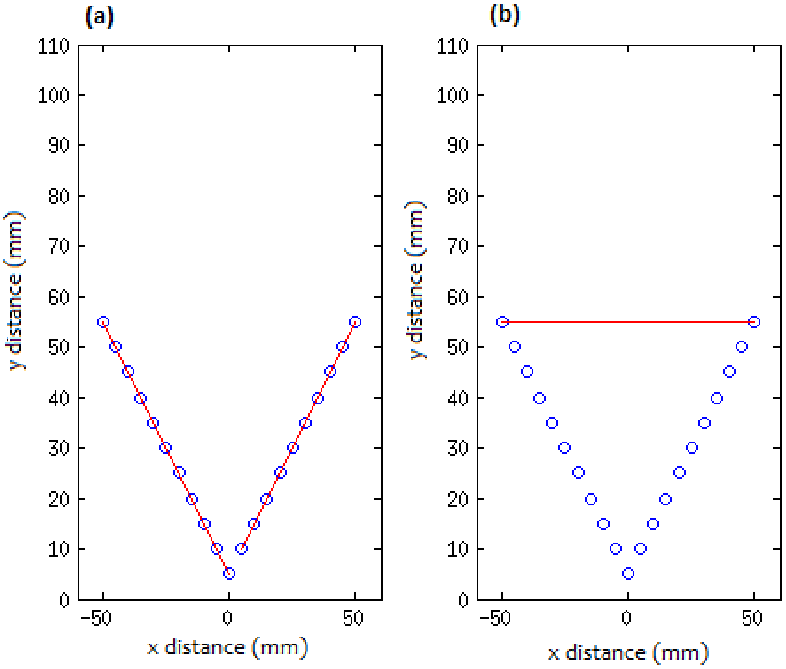

Figure 4.4: Figure showing a correct split (a) and an incorrect split (b) caused by a threshold which is too high for the split and merge algorithm

To determine the correct threshold for the data, a test will be run on 20 different point clouds, collected with the laser range finder. These datasets were generated by placing a box at various distances in the $x-$ and $y-$ direction from the sensor (which is at the origin). The range of distances from the laser range finder to the corner of the box is from -3 m to 3 m in the $x-$direction and from 1 m to 3 m in the $y-$direction. These distances are chosen for two reasons. The first is that the maximum range that the laser range finder can determine is 4 m, therefore this chosen as the maximum location for the box, because that has a distance of approximately 4 m to the corner of the box from the origin. It is important to note that the laser range finder cannot resolve the two sides of the box correctly at this distance, but the data are still useful, as they are in a straight line (see Figure 4.5). The second reason for this range is that with a mobile robot, objects within these ranges are important, whereas objects which are further away have higher uncertainty associated with their measurement and can therefore be discarded. The laser range finder has a maximum angle resolution of 270°, although for the purposes of this experiment, only objects in front of the laser range finder (i.e. objects which are at a positive y distance) are considered.

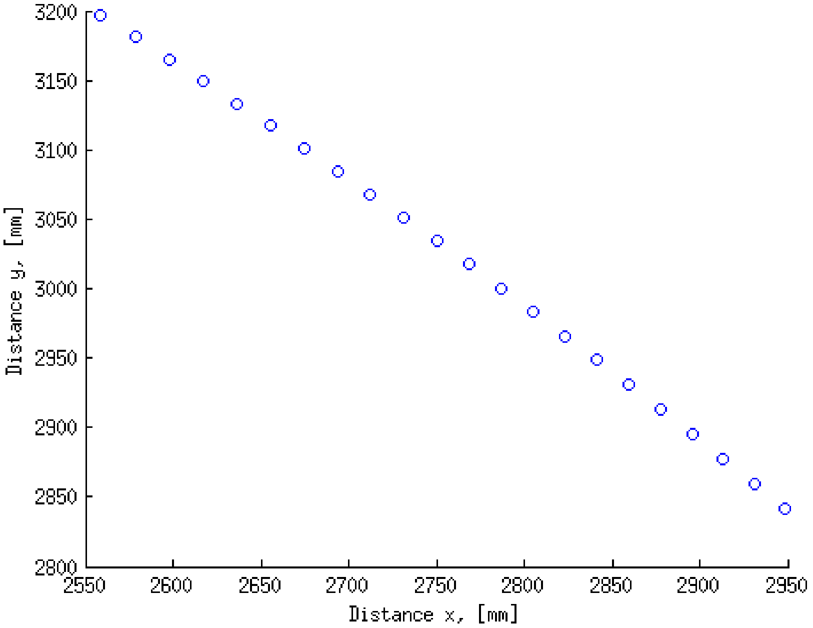

Figure 4.5: Figure showing the point cloud when the box is placed at $x = 3$ m and $y = 3$ m. The box is not properly resolved by the laser range finder, however the result does appear to be a straight line, and is therefore still used for this experiment. The laser range finder is at the origin.

These data are then processed by the split-and-merge algorithm. The results are recorded by determining the correct number of lines that should be fitted to the data. The correct number of lines for each set of data was verified by the author, and is entirely dependent on what parts of the box the laser range finder can "see". This means that when the box is directly in front of the sensor, only the front of the box is visible, and therefore the correct number of lines is 1. When the box is offset, the correct number of lines is 2 - as two sides of the box are visible to the laser range finder. The exception is when the box is placed at 3 m is the $y-$direction and at either -3 m or 3 m in the $x-$direction. At this distance, because the corner of the box is more than 4 m away, the sensor cannot correctly resolve two sides of the box, and therefore the box appears as a straight line to the sensor (see Figure 4.5. Therefore, for this case, the correct number of lines is 1. These numbers of lines are recorded in a table in Appendix B, along with the location of the box for each dataset.

The number of lines fitted for each of the thresholds is then determined and compared to the correct number of lines. These data are presented in a table in Appendix B. The thresholds used range from 15 to 350. The results for this test are represented and discussed below.

The bar graph in Figure 4.6 represents the results of this experiment. The percentage of correct and incorrect splits is shown as red and green sections of each bar. The bargraph shows that the ideal threshold is between 75 and 250. Below 75, the number of incorrect splits increases, and the same is true of thresholds greater than 250. Dataset 10 appears to have more incorrect fits than the others.

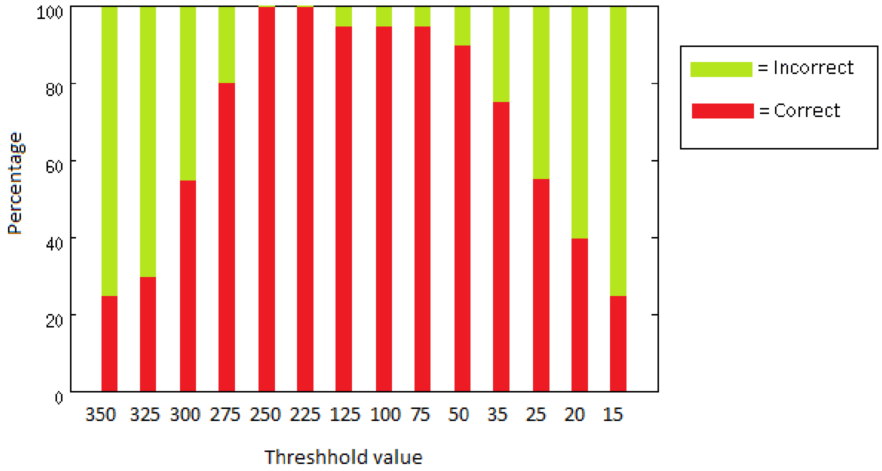

Figure 4.6: Bar graph showing the percentage of correct to incorrect splits using various different threshold values for the split and merge algorithm. All of the test datasets are represented - each dataset represents 5 percent in this graph.

In Figure 4.6, all "incorrect" splits recorded above a threshold of 250 are considered to be incorrect because too few lines have been fitted to the data. It is important to note that datasets for which 1 line is "correct" cannot have too few lines being fitted to the data. The "incorrect splits" for thresholds lower than 125 are incorrect because too many lines were fitted to the data.

These results show that the hypothesis that lower thresholds cause too many lines to be fitted to the data and thresholds which are too high cause too few lines to be fitted holds for this data. It also shows that the optimum values have a large range - between 75 and 250. Although dataset 10 seems to only have correct fits for thresholds between 125 and 250. This is because dataset 10 contains much more noisy data than the other datasets, and therefore the low thresholds which are fine for the other datasets are not suitable for more noisy data. This is an important result, as it shows that if data are more noisy, the threshold needs to be higher to account for this. This means that noisier datasets should be optimised differently.

The graph also shows that there is a very steep drop-off in correct fits above a threshold of 250 and below a threshold of 75. This means that if the threshold is slightly incorrect, or outside the optimum range, this will cause many incorrect fits to the data.

Therefore, the optimum threshold for the split-and-merge algorithm should be determined for each type of data. This includes data with more noise, as well as data which have different maximum and minimum distances, and data with a different scale (if the data were given in metres rather than millimetres, the optimum threshold would be 1000 times smaller). This optimisation should ensure that the number of correct splits is as high as possible for the particular type of data.

Effect of the threshold on the speed of the algorithm

The aim of this test is to determine the effect of the threshold on the speed of the algorithm. This test will use the results of the previous test to determine which thresholds should be used to determine this effect.

It is expected that the threshold value should have no effect on the efficiency of the algorithm. This is because the threshold for the split-and-merge is simply a number which is compared to another number using the ">" function. This function should not be affected by the size of either number being compared, unless the size of the number causes the number to require more bits of storage. However, even this decrease in efficiency should be negligible.

To achieve the aim of this test, eight threshold values were chosen from the results of the previous test. These thresholds range from 75 to 250 in increments of 25. The algorithm will be run for each of these thresholds for each different dataset used in the previous experiment - with the exception of dataset 10, as this dataset did not result in "correct fits" for all of the thresholds chosen for this experiment. The time taken to run the algorithm for each combination of threshold and dataset is recorded. It is important to note that the time taken may be affected by the number of points on the dataset (see the next experiment), and therefore the average time taken across every dataset will be determined to negate this effect.

The data collected for this experiment are shown in Appendix B. The average time taken for each threshold is also shown in the table. These average times are plotted against the threshold values in Figure 4.7. The graph shows that the time taken varies very little on the threshold. The approximate value for the time taken across the different thresholds is 0.0379 s. The standard deviation on this was about 0.016 s.

This result is exactly as expected from the theoretical understanding of how the threshold works given above. The time taken should be unaffected by the threshold, and the results seem to prove this hypothesis. The reason for the high uncertainty (shown by the high standard deviation) is that there were multiple number of different points in the sets used - and this affects the data greatly. It is important to note that though the result in the previous section would not hold for all types of data, the result in this section does. This is because the threshold should not cause a difference in the

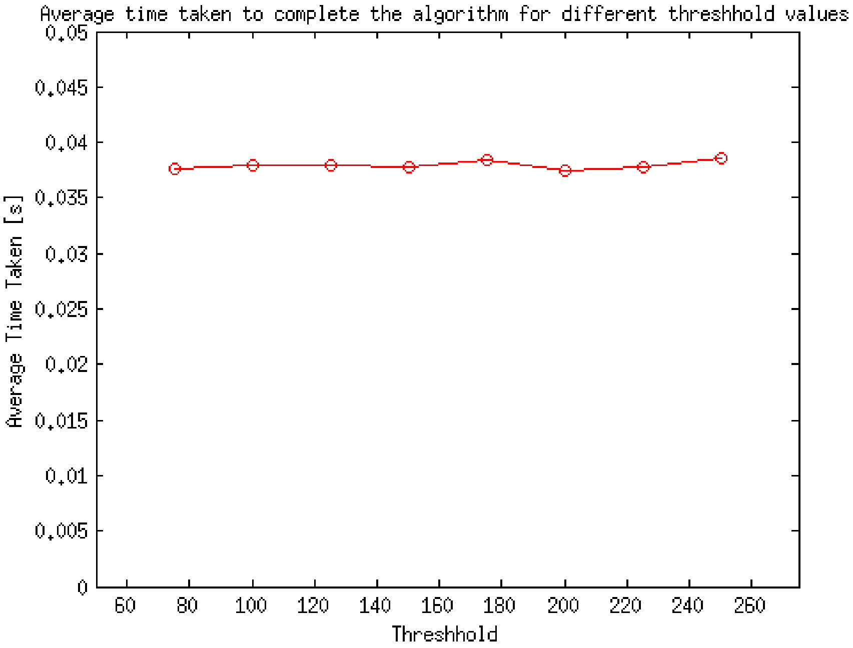

Figure 4.7: Graph showing the average time taken to complete the split-and-merge algorithm across all the data sets for each different threshold. The standard deviation was approximately 0.016 s.

efficiency of the algorithm, and this is independent of the type of data used.

4.2.2 Test of the effect of the number of points in the dataset and number of lines fit to the data on the time taken

There are two experiments in this section. The first experiment is to determine the effect of the number of points in the dataset on the time taken to complete the algorithm for that dataset. The second experiment is performed to determine the effect of the number of lines fit to the data on the speed of the algorithm. These tests are performed to determine whether the algorithm scales well with the number of points and the number of lines fit.

It is expected that as the number of points on the set increases, this will not cause the time taken to increase significantly. This is because, although the number of points being processed will increase the time taken slightly, the function in which the points are used is not a particularly processor-intensive process. Therefore, it is expected that the small increase due to the number of points will be negligible.

However, it is expected that an increase in the number of lines fit will result in a large increase in the time taken. This is because if no split is required, the algorithm only has to run once. If however, two lines need to be fit to the data, the algorithm will effectively have to run 3 times - once for the first iteration and twice after the set has been split (once for each “part” of the split). This means that the time taken to run the algorithm when two lines need to be fit to the data is expected to take 3 times as long as when the algorithm is run on data which only requires one line to be fit to the data.

The results for this experiment are the same as the data collected for the previous experiment. The number of points is recorded for each dataset. Each dataset is then processed by the algorithm with the same thresholds as used in the previous test. The average time taken for each dataset is then calculated. It is important to note that the threshold has been found to have very little effect on the time taken for the algorithm, and therefore the time taken is effectively found 8 times for each dataset.

The data collected for this experiment are given in Appendix B. The average time taken for each dataset is also shown in the table. These data are represented in Figure 4.8. The red line in this graph represents the average times for the datasets where 2 lines are fit, and the blue line represents the datasets where only one line is fit.

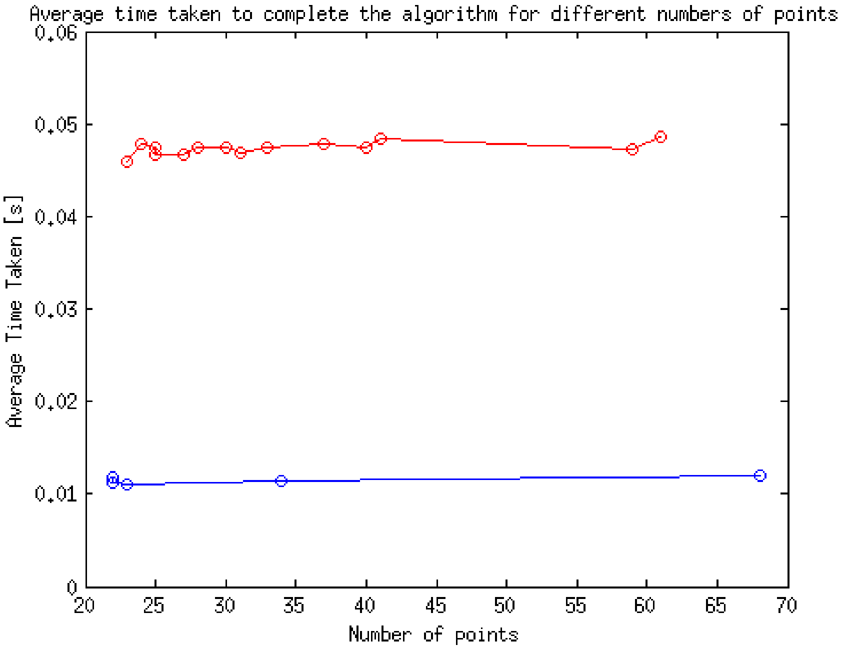

Figure 4.8: Graphs showing the average time taken to complete the split-and-merge algorithm for different numbers of points. The red line represents the datasets where 2 lines were fit and the blue line represents the data where only 1 line was fit.

The graph shows that the time taken is unaffected by the number of points in the dataset. Both lines are fairly straight, and there is little deviation from the average times for each of the different cases. However, the difference in the time taken for the sets where one line is fit and where two lines are fit is large. The average time taken for the sets where one line is fit to the data is 0.0115 s, whilst the average time for sets where two lines are fit is 0.0474 s. This means that fitting one extra line to the data causes the time taken to quadruple.

The first result (that the number of points in the set does not affect the processing time) is as expected. The number of points in the set may affect the processing time slightly for a greater increase in the number of points, but for these datasets the increase is negligible.

The second result is also similar to what was expected. It was expected that there would be an increase in the time taken by the algorithm of approximately 3 times the processing time for one line fit in comparison to two lines fit. This increase is, in fact, 4 times for the experiment given above. This is unexpected — although, this extra increase can be explained by looking at the process of splitting the data. This process is fairly processor-intensive — the point in the set where the split is to be made has to be found, and then the data has to be split into two new sets. This function would never run when only one line is fit to the data, however it clearly has a negative effect on the processing time, as it results in an extra increase in the time taken when two lines are fit to the data.

4.3 Hough Transform

In this section, the Hough transform is tested in a similar manner to the Split-and Merge algorithm. Note that for this section, "grid number" refers to the number of grid blocks used. i.e. a grid number of 500 means that there are 500 distinct ranges of r, and these will range from the minimum r to the maximum r.

4.3.1 Test to determine the effect of the discretisation of the angle θ on the efficiency of the algorithm

The aim of the test in this section is to determine the effect of the discretisation of the angle parameter on the efficiency of the algorithm. The angle in the implementation used in this dissertation determines the value of r for each different value of θ given by a discrete set of values within a given range. For example, a discretisation value of 1 degree means that the θ set is given by values from 1° to 180° is steps of 1°.

It is expected that as the angle discretisation gets smaller, the time taken to run the algorithm will increase. This is because decreasing the discretisation value has the effect of increasing the number of grid blocks in the θ direction. This means that each of the points in the initial point set has to be evaluated more times as the discretisation decreases. The value chosen should also be as small as possible, because this means that the angle of the line found by the algorithm will be closer to the true angle.

The method for this test is to vary the value of the angle discretisation to the following values (0.25, 0.5, 1, 2, 3 and 4). The algorithm will then be run for each of these values on a point cloud generated by a laser range finder. The algorithm will find two lines for this set for all of the angles, which should negate the effect of fitting more lines to the data. The grid number will be set at 80 for each of the runs. The algorithm will be run for each value of the discretisation of the angle three times, so the an average for the processing time can be determined.

These data are shown in a table in Appendix B. The average values are plotted against the angle discretisation in Figure 4.9. The graph shows that as the angle discretisation decreases, the time taken increases quickly. This relationship appears to be hyperbolic (this can easily be seen by plotting the time taken against the inverse of the angle of discretisation). The processing time is fairly consistent between 1 and 4.

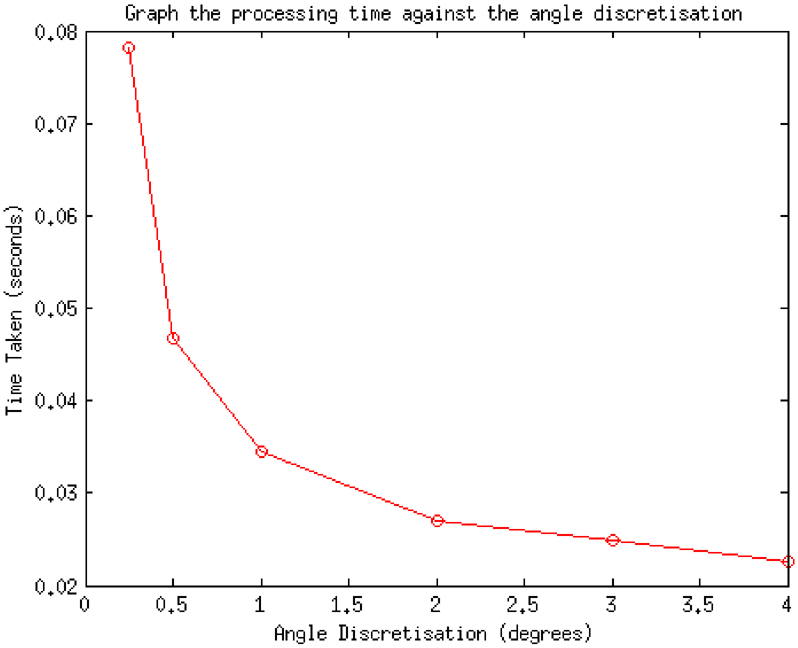

Figure 4.9: Figure showing the relationship between the time taken to run the Hough transform and the angle discretisation.

This result is as expected from the theory. The angle discretisation has a large effect on the time taken for the Hough transform. This effect is very noticeable below one - the time taken increases very quickly below 1. Values of 1 and above have fairly similar processing times. We know from the theory that a smaller value is better, as this means that the angle found by the algorithm is closer to the actual angle of the line. Therefore, an angle discretisation of 1 degree is chosen for the purposes of this dissertation.

4.3.2 Test to determine the effect of the grid number on the accuracy of the algorithm

This test is used to determine the effect of the grid number on the accuracy of the lines generated as the output of the Hough transform algorithm. The effect of the grid number on the speed and efficacy of the algorithm will be tested in a later section. This test will simply test whether increasing the grid number has a positive or negative affect on the accuracy of the line found by the algorithm.

It is expected that and increase in the grid number will have a positive effect on the accuracy of the lines generated by the algorithm. This is because the grid number defines the discretisation of the parameter r, and therefore the higher the gridnumber, the closer the r value found by the algorithm will be to the actual value of r for the data. This is because the ranges of r for each grid block will be smaller.

To perform this test, a set of points in a straight line at 45° is generated, and then the algorithm is run on this set for 6 different values of the grid number - from 10 to 60 inclusive, in increments of 10. These lines are the plotted along with the "true" line, and then the effect of the grid number on whether the algorithm find a line which is closer to the true line or further away can be determined easily. Note that for this test, and all future tests, an angle discretisation of 1° is used.

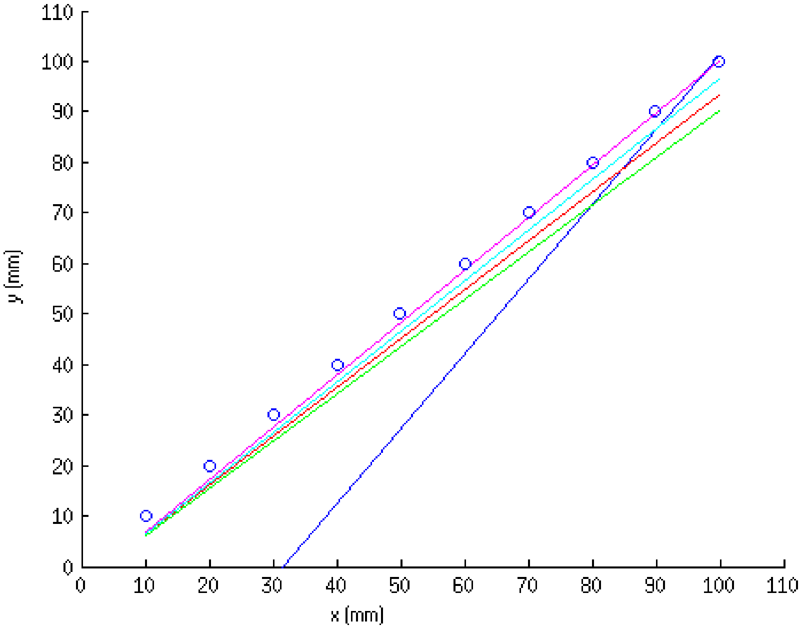

Figure 4.10: Figure showing the results of the Hough transform algorithm for multiple grid numbers

The results for this test are represented in Figure 4.10. Each of the different grid numbers is represented by a different coloured line, as given in the table below:

Table 4.1: Table showing the line colours for each of the grid numbers

Grid number	**Line Colour**
10	Blue
20	Green
30	Red
40	Cyan
50	Yellow
60	Magenta

From this figure, we can see that the line which is furthest from the “true line” (shown by the blue dots) is the blue line - representing the output with the grid number at 10. This is as expected from the theory – a lower grid number results in a lower degree of accuracy. The closest line is the magenta line, which falls directly on top of the line that would connect the blue dots. The magenta line represents a grid number of 60.

This result shows that an increase in the grid number has a positive effect on the accuracy of the algorithm, in terms of how close the line it finds is to the “true” line for the dataset. This result will be further tested in the next test, to see whether increasing the grid number too much causes the algorithm to find more lines than it should.

4.3.3 Determination of the optimal grid number for the Hough transform, based on the efficacy of the algorithm

The aim of the test in this section is to determine the effect of the grid number on the efficacy of the algorithm in finding the correct lines to fit to the data. The grid numbers used in this section are not specific to the type of data used, as they simply define the number of grid blocks in the r direction - the actual range for each grid block is determined by the maximum and minimum values for the specific dataset.

If the grid number is too large, too many lines will be fit to the data. This is because the range of each grid block is much smaller, and therefore point which should pass through the same block now pass through different blocks.

Two situations can arise if the grid number is too small. The first is that the algorithm may not find all the lines that should be found in the data, because sine waves representing the points may pass through the same block, where they should pass through different blocks. This is because the range of r value in the block will be greatly increased. The second effect is that the line may not be a good representation of the line. This can occur for the same reasons as detailed in the previous test - that the r value is discrete, and therefore it is possible that if the range for each block is too wide, the r value found will not be a good representation of the data.

The experiment used to determine the effect is as follows. Each of the datasets used in the test for the split-and-merge algorithm is run through the algorithm with multiple different grid numbers. It is determined whether the number of lines found by the Hough transform is too many, too few, or simply not close enough to the data to be considered a correct fit. The number of incorrect fits (defined as too many lines, too few lines or not close enough) for each grid number is then determined. The grid numbers used for this experiment range from 10 to 150. The actual values of these grid numbers are shown in the results table for this test in Appendix B.

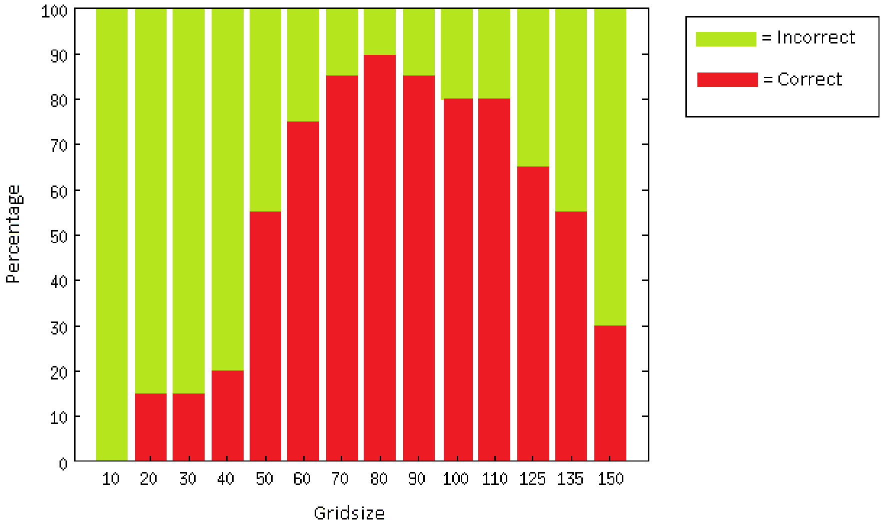

Figure 4.11: Figure showing the results of the grid number test for the Hough transform

All the data collected for this experiment are shown in Appendix B. The results for this experiment are represented in Figure 4.11. This graph shows that the number of correct fits increases up to a point (a grid number of 80) and then decreases. The increase between 40 and 80 is very steep, but the downward slope that occurs after a grid number of 80 is much shallower.

This result is exactly as expected from the theory. The best grid numbers seem to be between 60 and 110. These values result in a fairly high number of correct fits in comparison to the other values. The data in the table in Appendix B shows that the grid number generates too many lines in the data for values above 110 and too few for lower values, although the bounds of "correct" fits are very different for the different sets of data. Incorrect fits are all found for values below 60 - which means that any value below this does not result in a true representation of the data.

The range of values found in this test will be used in the next test to determine the effect of the grid number on the efficiency of the algorithm.

4.3.4 Determination of the effect of the grid number on the speed of the algorithm

This test will determine the effect of the grid number on the efficiency of the algorithm. This will determine whether the grid number needs to be minimised, in order to prevent a loss in efficiency.

The grid number is expected to have a negative effect on the speed of the algorithm. This is because, firstly, when the grid is filled, there are more blocks that now need to be filled, and therefore this process will take longer. Secondly, the process of running through the blocks to determine the highest number of votes will also take longer for the same reason.

For this test, as the algorithm did not find the correct lines for every grid number between 60 and 110 for every dataset which was used in the previous test, datasets need to be generated. These datasets contain 11 different numbers of points between 10 and 210. The data should only have one line fitted to it. This was tested with the Hough transform, and found to be correct. Each set is then processed by the Hough transform with 7 different grid numbers - ranging from 50 to 110. A grid number of 50 is included to give a better range to determine the effect of the grid number on the efficiency.

Each dataset is processed by the Hough transform with the different grid numbers and the time taken to run the algorithm is determined. The average for each grid number is also determined - this should negate any effect that the number of points has on the efficiency.

The results of the test are shown in Appendix B. They are represented in the graph in Figure 4.12. This graph shows that as the grid number increases, so does the time taken. This trend seem to be linear within this range - the least squares gradient of the line is approximately 0.0005 s/increase of 1 in grid number. In other words, if the grid number is increased by 10, the time taken increases by 5 ms.

This result is as expected. The time taken does increase due to the grid number - because of the reasons outlined above. However, the increase is not large enough that

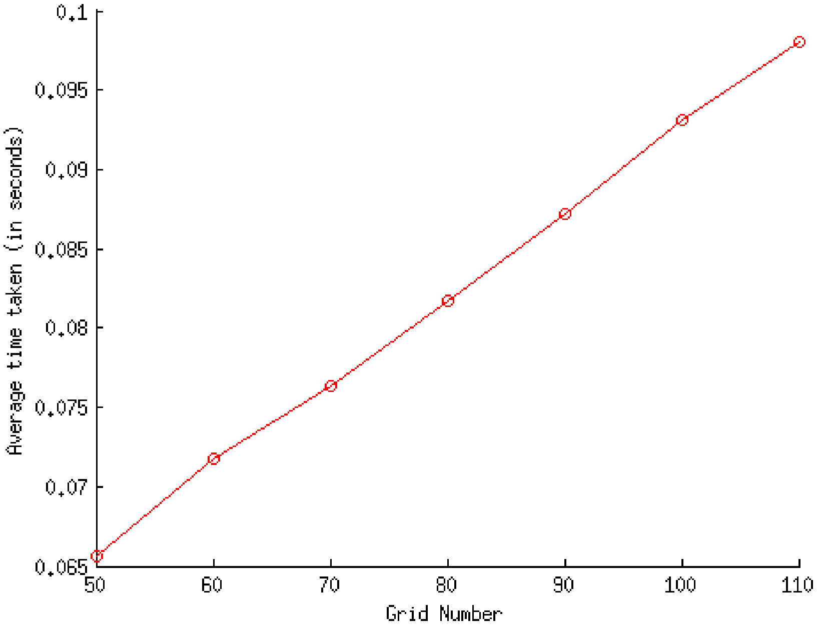

Figure 4.12: Figure showing the average time taken against the grid number. This shows that the time taken increases as the grid number increases. Therefore, the grid number should be kept as low as possible.

the grid number should be optimised for efficiency. The grid number has to be between 60 and 110 for the algorithm to be accurate, and the increase in time taken over that range is only 25 ms, and considering that the sensor only takes one measurement every 100 ms, it is not necessary to optimise this speed.

4.3.5 Test to determine the effect of the number of points in the set on the efficiency

The aim of this test is to determine the effect of an increase in the number of points in the set on the time taken to run the algorithm. This is tested to determine whether the set needs to be pre-processed to reduce the number of points in the set.

There is expected to be an increase in the time taken when the number of points in the set is increased. However, this change is expected to be negligible in terms of the time taken to run the algorithm, as well as in terms of the time taken for the sensor to take a scan. The increase in speed should be caused by the fact that there are now more points that need to be converted and placed in the grid.

This experiment uses the same data as used in the previous experiment. However, now the average is found across the different grid numbers - so as to negate the effect of the grid number on the increase in speed of the algorithm.

The results are shown in a table in Appendix B. These data are represented in the graph in Figure 4.13. This graph shows that the time taken to run the algorithm does increase with the number of points in the set. Once again, this relationship appears to be linear over this range. The least squares gradient of the line is 0.0004 s/point. This means that for an increase of 100 points, the time taken increases by 40 ms.

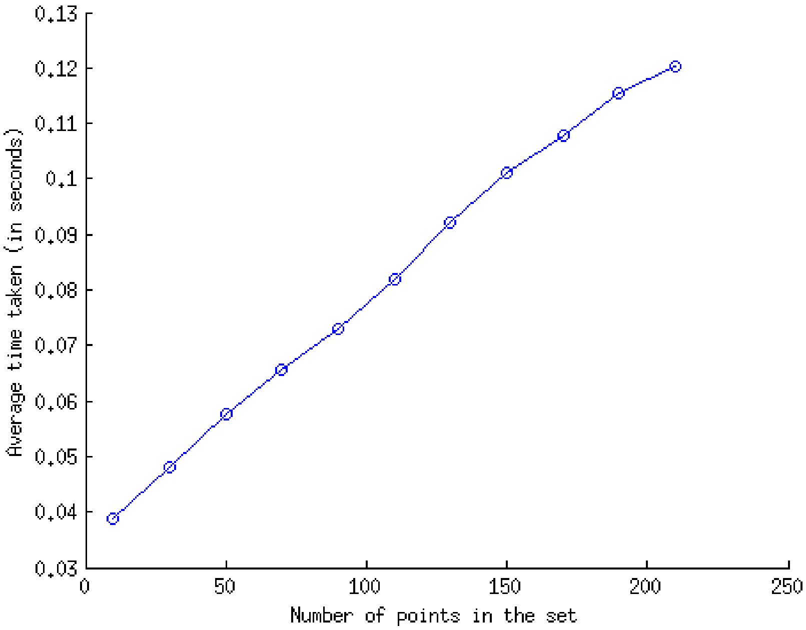

Figure 4.13: Figure showing the average time taken against the number of points in the dataset

This result is as expected - the number of points does affect the time taken. This increase is larger than expected. An increase of 40 ms for an increase of 100 points is fairly large, in comparison to the increase due to the grid number in the previous section. This means that if the sensor takes 100 more samples, half of the time between that sample and the next sample will be taken in processing the extra points.

Having said that, the Laser Range Finder has a maximum number of points in a sample of 270. Therefore, even if the maximum number of samples is taken, the time taken to process the algorithm is only very slightly more than the minimum sampling time (0.1 s per sample). The solution to this problem would be to reduce the sampling frequency slightly, and to ensure that every sample can be correctly processed by the algorithm before the next sample is output by the sensor.

Another conclusion that can be drawn from this result is that the process of filling the grid uses a lot of the time taken to run the algorithm. If a sensor with a higher number of points in a sample, or a higher sampling frequency is used, this process would need to be optimised to ensure that the Hough transform is able to process the data at a sufficient rate.

4.3.6 Test to determine the effect of the number of lines fit to the data on the efficiency

The aim of this test is to determine what the effect of fitting more lines to data is. This will determine whether the complexity of the data input into the Hough transform is an important factor.

It is expected that an increase in the number of lines fit to the data will increase the time taken to run the algorithm, although this increase is expected to be fairly small. The increase will be due to the fact that the maximum number of votes needs to be found more than once. The increase will be fairly small, as the process of looking for the maximum number of votes in the voting matrix is a fairly efficient process, and should therefore not have a large effect on the efficiency of the algorithm.

The method used to determine the effect of the of the number of lines fitted is as follows. Various datasets are created, each containing 500 points, but different numbers of horizontal lines that will be fitted to the data. This should mean that the number of points will have no effect on the time taken to run the algorithm. The various numbers of lines used are 1, 2, 4, 5, 6, 8 and 10. The algorithm will then be run three times for each of these sets of data, so that an average can be found for the time taken to run the Hough transform. The grid size chosen for this test is 100.

The data collected from this test are recorded in Appendix B. The average across the three runs is also shown in the table, and these average are plotted against the number of lines fit to the data in Figure 4.14. The figure shows that there is a clear upward trend in the time taken to run the algorithm as the number of lines fit increases. The increase is fairly small - only increasing by approximately one third between one line and ten, which is a fairly slow increase. There is a slight dip in the data at 6 lines found. This is probably caused by a slight inaccuracy in the data used - and the dip is very small, so its effect is considered to be negligible.

This result is as we expected from the theory. The total change between 1 line and 10 lines being fit is only about one third of the time it takes to run the algorithm on one line. Therefore, it is considered not to be an important factor for the purposes of this dissertation, as this study deals with the sensors in sparse environments.

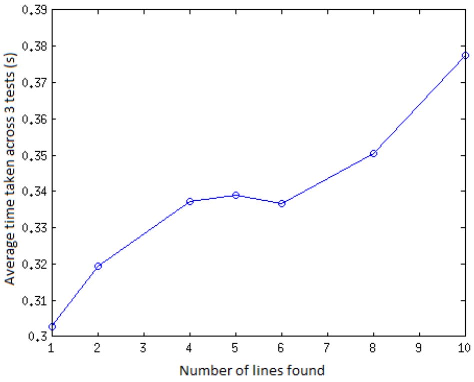

Figure 4.14: Figure showing the average time taken against the number of lines fitted. This shows that as more lines are fitted by the Hough transform, the time taken also increases.

4.4 Discussion and comparison

In terms of how well the two algorithms find lines in Laser Range Finder data, the two algorithms are fairly similar. They both extract the right number of lines, assuming that the correct threshold or grid number is chosen. For both algorithms, the threshold (or grid size) requires the same process of optimisation, as they are equally important in both algorithms. This means that in terms of how well it finds the lines, the two algorithms are equally good.

In terms of accuracy, the Split-and-Merge algorithm appears to be more accurate than the Hough transform. This is because it uses a Least Squares line fit, while the Hough transform uses a discrete r and θ value to describe the line. The Hough transform is very susceptible to the grid size in this sense (another reason for the grid size and angle parameterisation to be carefully optimised). The Hough transform does, however (if the grid size is correct), get a “close enough” line most of the time - even if the scale change (at least it seems so), however, the Split-and-Merge would need to be re-calibrated for every different scale.

In terms of the speed of the algorithm, the Hough transform is susceptible to many different parameters (number of points, number of lines fitted and grid size). The Split-and-Merge, however, is far more sensitive to the number of lines that are fitted to the data than the Hough transform (the processing time quadruples for two lines in comparison to one line). The same is not true of the Hough transform, although there is an increase, it is not significant enough to cause a problem. The Split-and-Merge, however, is very susceptible, and although the sensor would be used in a sparse environment (less than 4 lines fit), the Split-and-Merge is considered to be a worse candidate due to this major increase in the time taken.

The final category for comparison is the ease of extension into a 3-dimensional algorithm. The Hough transform has previously been extended into 3-dimensions [42]. The extension is relatively straight forward, as it simply requires an extension of the parameterisation into 3D, and a 3-dimensional grid, instead of a 2-dimensional grid.

The Split-and-Merge algorithm, on the other hand, has not been extended into 3-D before. It would be much more complicated to extend it into an algorithm for line extraction, as it would require splits in two directions, and a least squares fit to a plane, rather than a line. This makes it much more difficult to do, and is therefore not considered to be a good candidate for extension.

In conclusion, the Hough transform performs well in comparison to the Split-and-Merge algorithm in 2-dimensions. There are some factors for which is performs worse - such as the accuracy, however this is expected. Due the fact that it is not susceptible to scale (as the split-and-merge is), it is considered to be a good algorithm for mobile robotics in 2-dimensions as robots will not always work in the same environment. It is also relatively easy to extend into 3-dimensions. For this reason, it is considered to be a good candidate for extension into 3D to determine its usefulness as a 3D algorithm for mobile robotics. This will be demonstrated in the next chapter, along with an investigation into the use of the Hough transform with the Kinect sensor in mobile robotics.

Chapter 5

Extension of Hough Transform into 3-Dimensions

In this section, the development of the Hough Transform from the 2-dimensional algorithm for line extraction shown in the previous chapter, to an algorithm that can be used for plane extraction on 3-dimensional Kinect data is investigated. The pre-processing required (such as edge-detectors and smoothing algorithms) will also be presented and tested. The efficacy of the Hough Transform will be investigated for use with SLAM for Mobile Robotics.

5.1 Development of the 3-dimensional Hough Transform

For the extension of the Hough Transform into 3-D, the parameterisation needs to be extended into 3-D, and the grid needs to be a 3D grid (with cubes rather than squares). The extension into 3D is presented in [42], [31]. This extension uses the Hesse parameterisation of a plane, which is:

$$p_x \cos\theta \sin\phi + p_y \sin\theta \sin\phi + p_z cos\phi = \rho \tag{5.1}$$

where (p_x, p_y, p_z) is a point in the (x, y, z) plane, the normal vector to the plane on which this point makes an angle θ with the z-axis, and its projection into the $x - y$ plane makes an angle ϕ with the x-axis, and the normal vector has length ρ to the plane, as shown in figure 5.1. This is the extension of the parameterisation shown in the previous chapter, which is used for the 2D Hough Transform. The theory behind how this works is the same as for the 2D Hough Transform - each point in the (x, y, z) plane is transformed into a 3D “sine wave” in the parameter space. A voting scheme is then used to determine the plane that best fits the data.

Using this new transformation, this is the algorithm for the 3D Hough Transform:

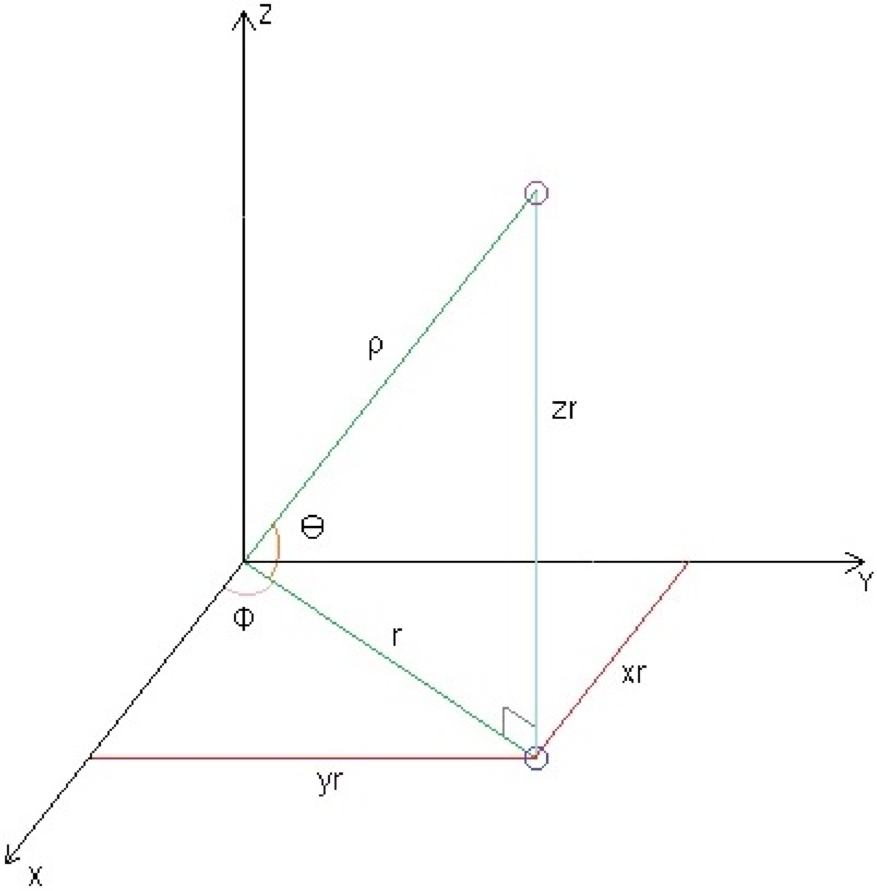

Figure 5.1: Figure showing the meaning of the parameters θ, ϕ and ρ in the Hesse parameterisation. The vector labelled ρ is the normal vector to the plane. The plane is not shown in this illustration.

Step 1: A set (matrix) $\mathbf{s} = (\mathbf{p_x}, \mathbf{p_y}, \mathbf{p_z})$ is initialised, where each row in the matrix represents one point in the (x, y, z) plane and each column represents the x, y or z coordinate for every point in the set (this step is similar to that for the 2D Hough Transform

Step 2: All of the points in the set $\mathbf{s}$ $(p(i)_x, p(i)_y, p(i)_z)$ are transformed into 3D sine waves in the Hough Space, each of which is described by $(\rho_i, \theta_i, \phi_i)$ (where θ and ϕ range from 1:180 degrees - this range is because angle "0" is equivalent to an angle of "360" and therefore is not used. If increments of 1 are used, there are exactly 180 values between 1 and 180 inclusive). This is achieved using the following formula:

$$\rho_i = p(i)_x \times \cos\theta_i \times \sin\phi_i + p(i)_y \times \sin\theta_i \times \sin\phi_i + p(i)_z \times cos\phi_i \qquad (5.2)$$

For the purposes of this project, this formula will be evaluated at discrete intervals of 1 degree for each of θ and ϕ.

Step 3: A 3D grid is then initialised. The grid will be determined in the ρ direction only, as both ϕ and θ are discrete in this formulation of the algorithm. The grid in the ρ direction is initialised in the same way as for the 2D Hough Transform.

Step 4: In this step, the voting grid is filled. The matrix is now a 3D matrix. The x direction in the matrix defines the different values for ρ, the y direction defines the values of θ and the z direction defines the values for ϕ. The voting matrix is filled in the same way as for the Hough Transform in 2D - by determining which cubes each of the sine waves found in step 2 pass through. The voting matrix in this implementation is, in fact, a 4-dimensional matrix (i.e. there is one 3D matrix per point) - it is filled with 1s and 0s in the same way as the one for the 2D is filled.

Step5: In this step, the total number of votes for each cube in the matrix is determined by adding all of the ones in the 4D matrix described in step 4. The maximum number of votes is determined and recorded, and the waves that pass through that particular cube are removed from the data. This process is repeated until all of the blocks are empty.

Step 6: Finally, each of the recorded blocks is converted back into the (x, y, z) space as a plane.

This process is used in the code implementation used for this dissertation. The code, along with a block diagram of how it works is shown in the Appendices. The implementation of this algorithm is tested later in this section.

5.2 Tests without pre-processing

The aim of the test in this section is to determine the effect of the number of points on the efficiency of the algorithm. This is performed to determine whether a pre-processing procedure needs to be applied to the data before it is run through the Hough transform. The data that comes from the Kinect contains over 300 000 points, and therefore if the algorithm does not scale well with the number of points in the set, that number of points will not be processable by the algorithm.

The number of points is expected to affect the efficiency of the algorithm negatively. The algorithm is expected to run quite slowly for even small numbers of points (for example 50 points). This is because, as seen in the previous section, the Hough 2D is not particularly efficient, and the Hough 3D adds an extra dimension and increases the size of the grid matrix. Therefore, it is expected to scale even more poorly with the number of points in the set.

The data used for this test is “created” by the author for the purpose - a matrix of (x, y, z) coordinates is created - each of the points in the dataset representing a point cloud. All the points in the dataset will be on a single plane by design. Each set will contain a different number of points - 9, 16, 25, 36, 49, 64, 81, 100 and 121. Each of the sets is then used as an input to the Hough 3D algorithm. The time taken to run the algorithm on those sets is then recorded.

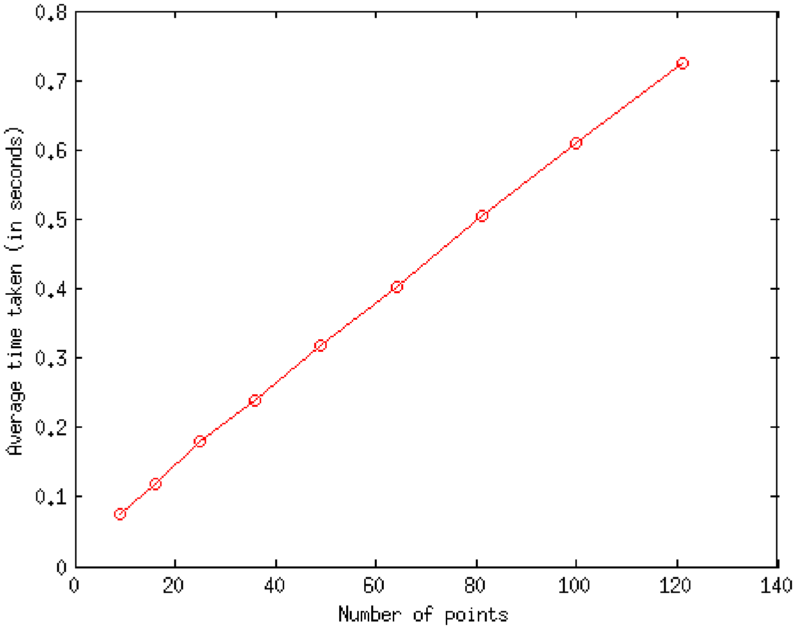

Figure 5.2: Figure showing the average time taken against the number of points in the set. This shows that the time taken by the 3D Hough transform increases linearly with the number of points in the set.

The result of this experiment is shown in Figure 5.2. The figure shows that the time taken increases approximately linearly against the number of points (for the range shown). The gradient of the line is found by Least Squares to be 0.0058 s/point. This means that a 100 point increase results in more than half a second of increase in the time taken. This is as we expected from the theory and prior knowledge - that the time taken would be negatively affected by the number of points in the set.

Based on the results above, some method of pre-processing the data is required before the data are input in to the Hough transform. Methods for doing this are discussed in the next section.

5.3 Pre-processing algorithms

Due to the nature of the environments that the Kinect is being used in, an edge-detector is a good choice of pre-processing algorithm to use on the data before it is run through the 3D Hough Transform. MATLAB has six different built-in edge detectors. These are Sobel, Prewitt, Roberts, Laplacian-of-Gaussian, Zero-Cross and Canny. The Laplacian-of-Gaussian Edge Detector is a special case of the zero-crossing edge detector. The three that will be tested to determine which is the best for this project are Sobel, Laplacian-of-Gaussian and Canny - due to the fact that these three cover the different types of edge detectors. Each of these edge detectors is described briefly, and then tested below.

5.3.1 Theory of the Edge Detectors

This section will describe how each of the three edge detectors (Sobel, Laplacian-of-Gaussian and Canny) work, and will list the advantages and shortcomings of each of the algorithms.

Sobel Edge Detector

The standard Sobel formulation computes an approximation of the gradient of the intensity function of the image. To achieve this, it uses two convolutions. The first is the convolution of a 3x3 matrix $\mathbf{C_x}$ with the image intensity function, the second is the convolution of a matrix $\mathbf{C_y}$ (also a 3x3 matrix) with the image function. The result is two separate images - $\mathbf{R_x}$ and $\mathbf{R_y}$ which are the gradient approximations in the x and y directions, respectively. The matrices $\mathbf{C_x}$ and $\mathbf{C_y}$ are both combinations of a differentiation kernel and an averaging kernel. They are formed as follows:

$$\mathbf{C_x} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{5.3}$$

$$\mathbf{C_y} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{5.4}$$

where $\mathbf{IM}$ is the Image Intensity matrix, $\mathbf{R_x}$ and $\mathbf{R_y}$ are formed as follows:

$$\mathbf{R_x} = \mathbf{C_x} * \mathbf{IM} \tag{5.5}$$

$$\mathbf{R_y} = \mathbf{C_y} * \mathbf{IM} \tag{5.6}$$

Now, these two matrices are combined to find the approximation of the gradient magnitude at each point in the image:

$$\mathbf{R} = \sqrt{\mathbf{R_x}^2 + \mathbf{R_y}^2} \tag{5.7}$$

For the Sobel Edge Detector, the maximum of the matrix $\mathbf{R}$ is found, and these points are returned as an edge.

The Sobel edge detector has a few drawbacks. The first is that it is very susceptible to noise, due to the small window size used. It is also rather inaccurate, although it is still used in many image processing applications. Its great strength is that, because of the small kernel size, it is a very fast edge detector.

The Laplacian-of-Gaussian edge detector

The Laplacian-of-Gaussian (LoG) Edge detector uses a similar method to that of the Sobel Edge Detector, but instead of finding the first derivative of the image intensity, it computes the second derivative. It uses an approximation to a Gaussian filter to do this. The kernels can become rather large, because the larger the kernel is, the more accurate the approximation to a Gaussian filter. Smoothing with a very small Gaussian on a discrete grid can have no effect. The effect of increasing the size of the Gaussian kernel is to change the variance σ^2 of the Gaussian kernel. In order to compute a LoG kernel, A Gaussian kernel is convolved with the Laplacian kernel **L**. **L** is given as:

$$\mathbf{L} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.8}$$

The edge detection is performed by finding “zero crossings” in the second derivative. A “zero-crossing” is defined as a point in the second derivative where a positive value passes through zero to become a negative value. These places define edges because at an edge, the gradient will be very high, which means the change in the gradient should be high around the point - and it should go from very positive to very negative. Areas of constant intensity should be zero in the second-derivative.

The method for finding the zero-crossings in the image is to apply a threshold to the LoG output - with the threshold equal to zero. This will produce a binary image. The boundaries in the binary image could be easily detected - by finding every “1” in the image that has at least one “0” neighbour. This will, however, bias the location of the boundary to either the foreground or the background.

One drawback of this algorithm is that the edge will also be biased to the foreground or the background, as the zero-crossing is likely to always fall between two pixels. Another drawback is that it is also fairly susceptible to noise. It could be more accurate than the Sobel detector due to the larger kernel size. It may, however be slower than the Sobel detector for the same reason.

The Canny Edge Detector

The Canny edge detector consists of four stages. The first is noise reduction, the second is to find the intensity gradient of the image, the third is a non-maximum suppression stage, and the final stage is hysteresis thresholding.

The first stage (Noise Reduction) is used to prevent the edge detector from being too susceptible to noise. It uses a Gaussian filter, similar to that used for the LoG algorithm. It convolves a matrix that represents the Gaussian with the raw image. This has the effect of filtering the image. This should prevent the algorithm from being affected by any noisy pixels in the data.

The second stage is the same as that for the Sobel Edge detector. It uses an approximation of the gradient in the x and y direction to find the gradient magnitude for each pixel. Because of the addition of step one, this should be less susceptible to noise than the Sobel algorithm.

Step three is used for edge-thinning. This prevents the same edge from being found more than once in the same image. It achieves this by conducting a search of each image gradient (at each point). The algorithm then decides whether that pixel is a local maximum. It does this by using not only the magnitude of the gradient, but also the direction of the gradient. To determine whether the pixel is a local maximum, it uses the direction of the change in gradient to determine how it compares to the surrounding pixels (i.e. if the gradient direction is vertical, the point is considered to be an edge if its gradient is greater than those of the two pixels around it in the horizontal direction - the pixel immediately to its left, and the pixel immediately to its right). It does not matter what the sign of the gradient is - i.e. it does not matter whether the gradient is left-to-right or right-to-left.

The final step allows “strong” and “weak” edges to be found. This is done because high intensity gradients are more likely to correspond to edge than low intensity gradients. It is very difficult to specify the gradient intensity when a the edge changes from “strong” to “weak”. Therefore, a high and a low threshold are used. An assumption is also made that edge are likely to be in a continuous line with other edges. This allows single noisy points to be discarded. To start with, the high threshold is used. This results in points that are likely candidates for being edges. Then, the lower threshold is used while moving along a continuous line. This allows weak edges to remain, assuming that they are in a continuous line with a strong edge. Other weak edges will be removed from the data.

One problem with this algorithm is that it is slower than the other two, due to the higher number of operations that are performed. It should, however, be less susceptible to noise. It should also result in more accurate edges, due to the final two step in the algorithm. The final step, in particular, should allow non-edge pixels to be removed from the final output.

5.3.2 Testing of the edge detectors on the raw data

This section will test each of the algorithms described above for speed, accuracy and whether they find the edges that are relevant in the image (or find edges which are there, but are not relevant).

Comparison of the accuracy of the algorithms

In this experiment, the accuracy of each of the three edge detectors described above is evaluated. This is performed to determine which of the algorithms would be best for and input into the Hough transform. This input algorithm needs to be consistent in the edge that it finds. For example, if there is a box in the scene, and the edges of the box are the "targets" for the edge detector - any features detected other than the edge of the box are essentially noise.

Due to the specific requirements of this dissertation, the Canny edge detector is expected to perform poorly, as it is very good at finding "weak" edges. This is undesirable in this case, as only the strong edges of the edge of the box are desired.

The process for running this test is as follows. A depth image is taken using the Kinect sensor. The scene is an empty room containing a 500 mm x 500 mm box placed 1.5 m in front of the camera. This image is shown in Figure 5.3. Each of the edge detectors is then performed on this image, and the number of "correct" points is recorded (correct point being those points which are on or near the edge of the box in the scene).

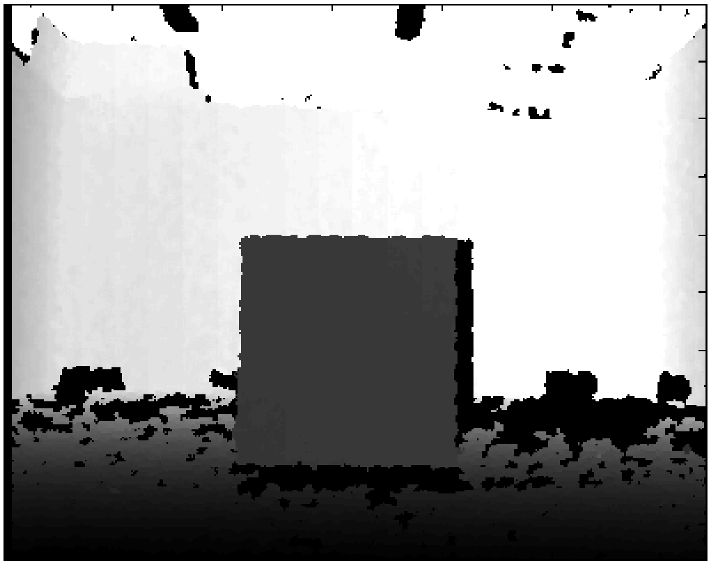

Figure 5.3: The depth image used for the edge detection

This table contains the correct number of edge points for each of the detectors, as well as the total number of edge points found by the detector. The difference between the total and the correct number gives the number of incorrect pixels.

Table 5.1: Table showing the number of correct and incorrect edge points found in the image by each of the edge detectors

Detector	Correct	Total	Incorrect
Sobel	940	5484	4544
LoG	894	7602	6708
Canny	913	13901	12988

From these data, we can see that each of the algorithms finds a similar number of correct points. This is expected, as the edge of the box has a constant number of pixels that can be found. However, the total number of "edges" found by each of the algorithm varies a lot – the Canny edge detector finds nearly double the number of edges in comparison to the other two. This is once again, as expected, as the Canny detector connects "weak" edge to "strong" edges, which means that some weak edge are not discarded by the algorithm.

The first result (that all three algorithms find a similar number of correct points) means that they all perform fairly well in finding the edge of the box. The disparity comes in when the number of "noise" points is evaluated. The Sobel and LoG edge detectors perform fairly similarly, but the Canny performs much worse.

Having said this, none of the algorithms perform particularly well - they all find large number of incorrect points in comparison to the number of correct points. This would indicate that some kind of pre-processing of these data may be necessary to reduce the noisy edges. This is discussed in Section 5.3.3.

Comparison of the speed of the algorithms

This test will determine which of the edge detectors is the fastest. Speed is important, as when SLAM is performed, it is imperative that the processing is finished for the previous frame before the next frame is captured by the sensor.

It is expected that the Canny will take longer than the other two algorithms, due to the nature of the way it finds edges - there are more operations that need to take place. The LoG is also expected to perform slower than the Sobel detector for the same reason.

For this test, the same tests are used as for the previous test. For each test, the time taken to run the algorithm is recorded.

The average time taken for each algorithm is recorded in the table below. The Sobel performs best, followed by the LoG and then the Canny, which takes nearly double the time of the Sobel detector.

Table 5.2: Table showing the average time taken for each of the edge detectors

Detector	Average Time Taken
Sobel	0.1524
LoG	0.1942
Canny	0.2853

Both of the results above are, to some extent, expected. The time taken of the Canny algorithm is expected to be higher than that of the other two, due to the extra operations that are performed. The LoG is also expected to take longer than the Sobel, due to the increased size of the convolution matrix.

Based on the two experiments above, the Sobel detector is considered to the be the best of the three, however, due to the number of "noisy" points in the output of all the detectors, extra pre-processing is necessary to remove these points. This pre-processing is discussed in the next section.

5.3.3 Reducing the noisy points in the output

This section will investigate how the raw data can be modified to reduce the number of noisy (incorrect) edge points that are found by the algorithm. Many of the incorrect edge points in the previous section were found because, when the Kinect gets no return from the scan, it automatically makes than point a zero. This means that all of the Kinect errors cause the gradient detector to find a large gradient. These are therefore reported as errors. This will be rectified by changing every zero point in the original image to a "NaN".

The other problem is that the background causes noisy points to appear. These can easily be removed with a threshold. Everything that has a depth of more than 4000 mm will also be given a "NaN" value.

In Figure 5.4, the difference between the output of the Sobel detector when the above changes are not made (in the left hand picture) and when they are made (the right hand picture) is shown. It is clear that including the processing described above, all the noisy points have been removed. However, the right side and bottom of the box have also been removed. This is because if we look at the original image (Figure 5.3), the right hand side of the box is entirely bordered by errors (black in the image), as is the bottom of the box. These are now "NaN", and are therefore not considered for the edge

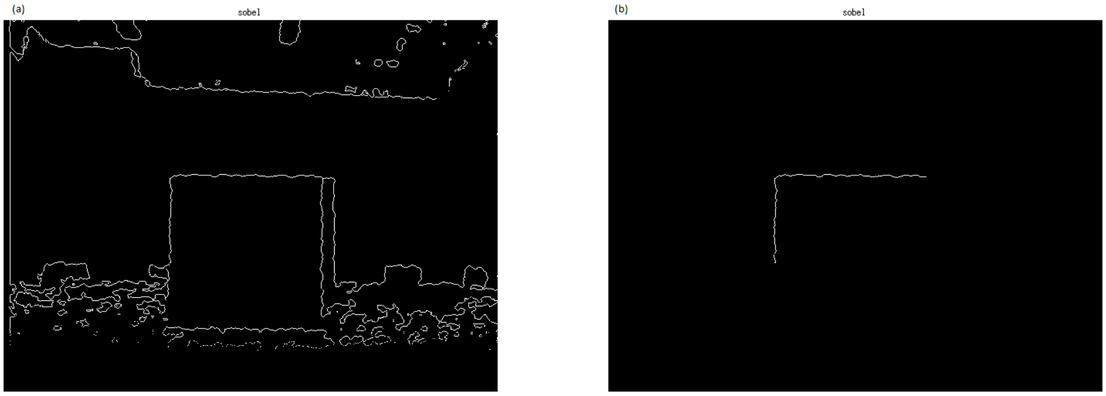

Figure 5.4: Images showing the output of the Sobel detector without the alteration of the data (a), and with the processed data (b).

detector. The bottom of the left hand side of the box is also cut off, for the same reason.

This is a problem - it would be better to leave a few noisy points, but get at least three sides of the box. Therefore, a slightly different approach is attempted. For this, the matrix of numbers is run through, from the bottom of the image to the top, and along the columns. The zero points are found, and if the point is zero, it is given the value of the pixel directly below it in the original image. This has the effect of filtering out all the zero values, and giving an approximation to what those pixels should have been, if the Kinect had been able to read them. The output of this processing for the Sobel detector is shown in Figure 5.5.

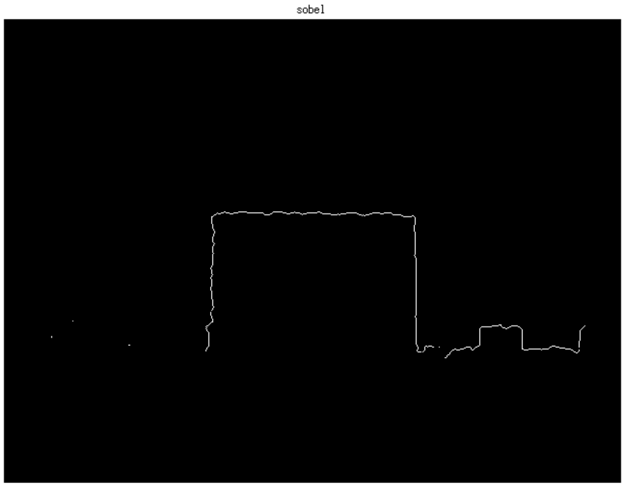

Figure 5.5: The final output of the Sobel detector, showing the significant reduction in the number of noisy points in the output.

This figure shows that three of the four sides of the box have been correctly identified. Unfortunately, the line where the floor connects to the wall is also found. This output is better than the previous outputs. Although there is more noise than for the first attempt at removing the noise, now three sides of the box are identified. This allows the box to be fully characterised in the Hough Transform. Therefore, this is the final processing for the Sobel detector.

This same processing is now tested for the LoG and Canny edge detectors. The outputs are shown in Figure 5.6. The LoG algorithm (output shown on the left) results in much more noisy points than the Sobel detector, especially on the line where the floor meets the wall. The Canny results in even more noisy points than that.

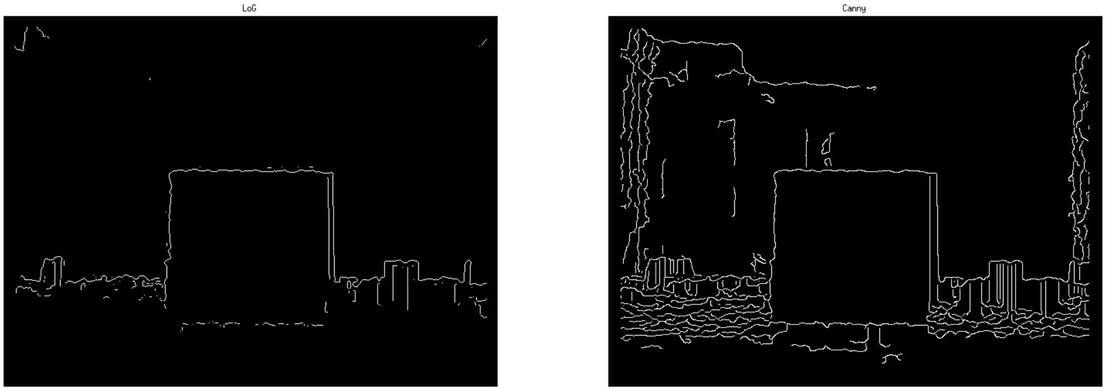

Figure 5.6: Images showing the output of the LoG (left) and Canny (right) detectors with the preprocessing

From the above tests, it is decided that the Sobel edge detector will be used, in conjunction with the pre-processing of the data as described above, as the pre-processing for the Hough Transform algorithm. The results of this final processing are given in the section below. It is important to note that the pixels with value "1" in the edge detector output are the edges of the box (at the depth of the box). The number of points is reduced to 500 - which should allow the Hough Transform to work correctly.

5.4 Final 3D Feature Extraction Algorithm

For this section, the pre-processing and edge detection that is described above is used before processing with the Hough Transform. The points that are output in the edge detector are 1s and 0s. The points at which the edge detector outputs a 1 are put into 3 vectors - one for the x value, one for the y value and one for the z value. These points are the inputs to the Hough Transform.

To test the Hough Transform, the algorithm is run on multiple different sets. These are all described below.

The first set of data is a simple cardboard box. It is placed 1.5 m and 1 m away from the Kinect sensor, at different angles. The ideal threshold for the 3D Hough will be determined. Each set has a "correct" number of planes that should be fitted. The correct number is determined by the author. For example, if only one side of the box is visible to the Kinect, the correct number of planes is one. However, if two sides are clearly demarcated in the image, this will have a correct number of planes of 2. This will be tested, along with the processing time for the data. The threshold is varied from 3 to 17, in increments of 2. The results are shown in tables in Appendix C.

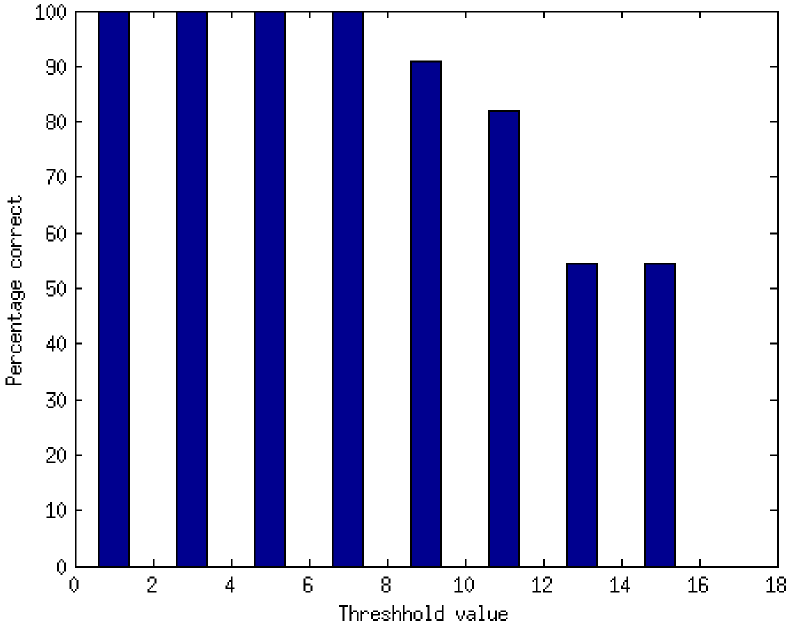

Figure 5.7: Bar graph showing the number of correct fits for each of the thresholds

The data for the number of correct fits to the data for each of the thresholds is shown in Figure 5.7. This figure shows that, as expected, when the threshold gets too high, the algorithm starts to fit more planes to the data than it should (as for the lines fitted in the 2D version). The threshold values are much lower for the 3D Hough than for the 2D Hough. This is for two reasons. The first is that when the threshold becomes too high, the algorithm simply requires too much computing power. The second is that, it is unlikely that more than two planes will need to be segmented, because of the way that the data is pre-processed. Therefore, a low threshold is desired. However, using a low threshold means that the accuracy may be compromised. Therefore, a higher threshold is desired - up to a point, if the algorithm becomes too slow, this is also a problem.

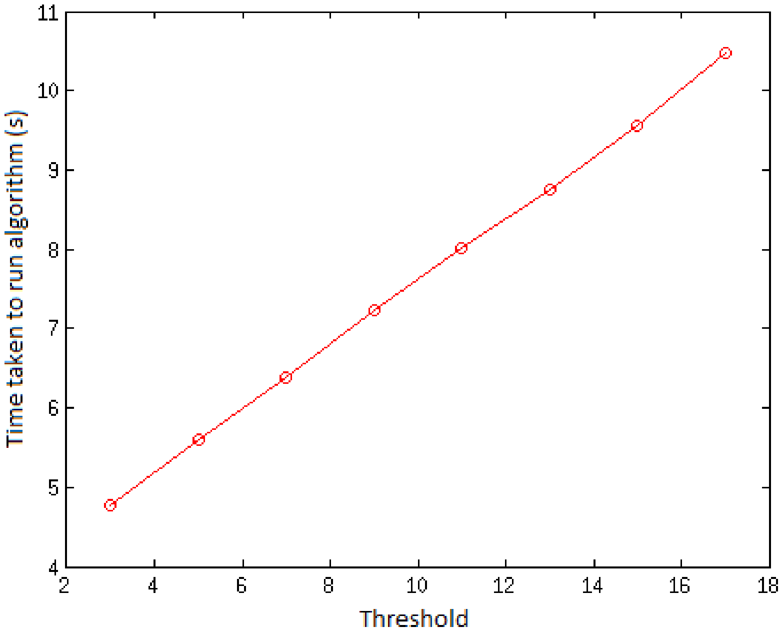

Figure 5.8: Graph showing the speed of the Hough transform depending on the threshold

Figure 5.8 shows the effect of the threshold on the speed of the algorithm for dataset 2. This graph shows that the threshold does have a great effect on the speed of the algorithm. The algorithm is already rather slow (taking approximately 8 seconds to run with the threshold at 11). The relationship between the speed and the threshold appears to be linear, with a gradient of 0.402 seconds per increase of 1 in the threshold. This is very bad. The threshold should therefore be high enough that the accuracy is not compromised, but not so high that the algorithm does not fit the correct planes, or the speed is too slow. Therefore, a threshold of 9 is considered to be ideal.

The final test is to determine the effect of the number of points on the speed of the algorithm. The speeds (for a threshold of 9) for each of the different sets are plotted against the number of points in the set. The result is shown in Figure 5.9. This graph shows that there is an increase in time taken as the number of points increases (as expected from the tests earlier in this section). The speed change is not large, however. The difference between the algorithm running on 360 points to 590 points is only 4 seconds. This is not too serious. Therefore, the edge detector is suitable for use

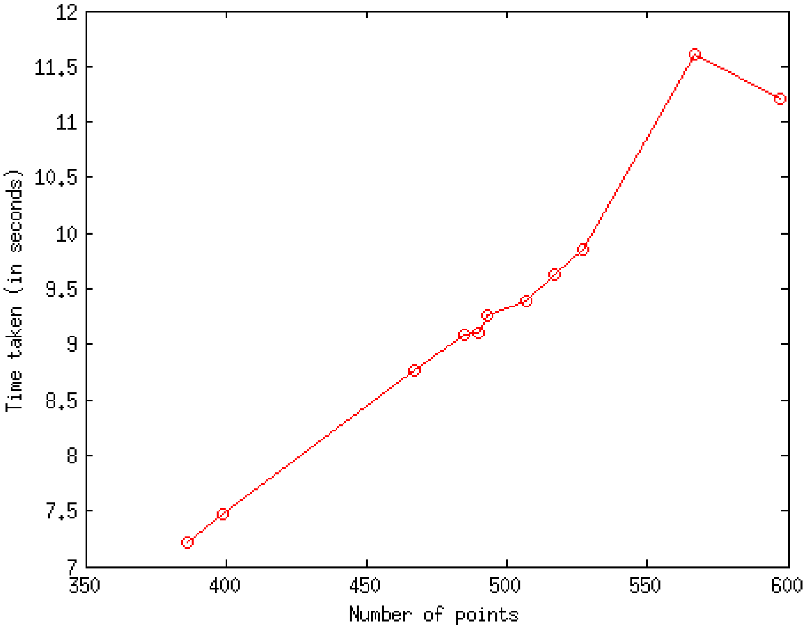

Figure 5.9: Graph showing the effect of the number of points on the speed of the algorithm

with the Hough Transform, as it reduces the number of points for all the test data to a reasonable level. Note that there is a small decrease in the time taken between the point at 567 points and 597 points (this increaase is less than 0.5 s). This is assumed to be because of noise, as for the other thresholds tested, this time taken continues to increase between these two sets.

The Hough Transform in 3D also achieves the task for reducing the number of points that need to be input into the SLAM algorithm. The output of the algorithm can be used to reduce the number of points to 4 per plane - one in each of the corners. More points in each plane could be used, to increase the accuracy of the feature tracking. This is a large reduction in comparison to the edge detectors (which reduce it from 300 000 to 500). This means that the Hough Transform does achieve the objective.

However, the Hough Transform is very slow. This may mean that it is unsuitable for use with SLAM, as it requires in the order of seconds to complete the segmentation of the planes from the data. This means that the Kinect could only take an image approximately once every 10 seconds - much slower than the sampling frequency. It also means that the robot would have to be very slow moving, or stationary until the Hough has completed the segmentation. If this is not the case, the robot may move too far, and therefore there may be no common features between two frames taken by the Kinect. Therefore, although the reduction in the number of features is useful, the speed which it takes to complete the segmentation is simply too slow. Therefore, the Hough Transform is not considered suitable at this time, with the current processing speeds available.

5.5 Investigation into the suitability of the Hough Transform for Mobile Robotics

This section will investigate whether the process described in the previous sections is useful for Mobile Robotics. First, it will be investigated for its usefulness as an aid for obstacle avoidance. This is very important for Mobile Robotics. In order to test this, we will determine whether the algorithm can extract planes from the raw data accurately enough that the robot could avoid crashing into these obstacles. This will be determined by calculating whether the planes extracted by the algorithm are “close enough” to the actual obstacles in the data - and also whether the size of the plane in the data is accurately extracted.

The second investigation will involve determining whether the algorithm is suitable for use as an algorithm used in SLAM to reduce the number of features input into the algorithm. Different types of environments will be used for this - sparse and cluttered - to determine whether features could be tracked between frames, and whether planes could be extracted accurately from the frames, so that they could be input into a SLAM process.

5.5.1 Suitability of the algorithm for obstacle avoidance purposes

For this section, tests will be run to determine the suitability of the Hough transform process for obstacle avoidance in robotics.

The first test will be to determine whether the extracted plane is accurate in terms of the parameters of the plane - i.e. if the obstacle is 2 metres from the robot directly ahead, does the algorithm extract a plane which is 2 metres from the robot, directly ahead. This is important, as, if the robot thinks the object is located at 3 metres, but it is in fact 2 metres away, the robot may crash into the object.

The second test will be to determine whether the *location* of the plane in the x and y directions is accurate. This means that if the object is directly in front of the robot, and is 50 cm square, will the algorithm be able to say exactly where the object is. This is important as if the robot thinks the object is 25 cm across, it may try and go around it and end up crashing in to it.

Test of the accuracy in extracting the correct plane from the data

This experiment will determine whether the Hough transform and the pre-processing described in this dissertation can correctly extract planes from simple planar data. The aim of this is to determine whether the algorithm is useful for obstacle avoidance in robotics.

The algorithm is expected to perform fairly well in both extracting the correct parameters of the plane and the correct corners of the planes. The grid number used is 9 and the angle parameterisation is 1 degree in both directions. The algorithm may, however, struggle to find more than one plane, and may also perform poorly in terms of the processing time needed.

In order to test this, the Hough transform will be run on 10 images from an online dataset [1]. The dataset used is the "rgbd_dataset_freiburg3_large_cabinet" dataset. An example of an image from this set is shown in Figure 5.10. This is used as it is the most useful one for simulating the movement of a robot with a Kinect sensor mounted on it moving around a room with a single large cabinet in it. This cabinet is a planar object, and is therefore suitable for this algorithm.

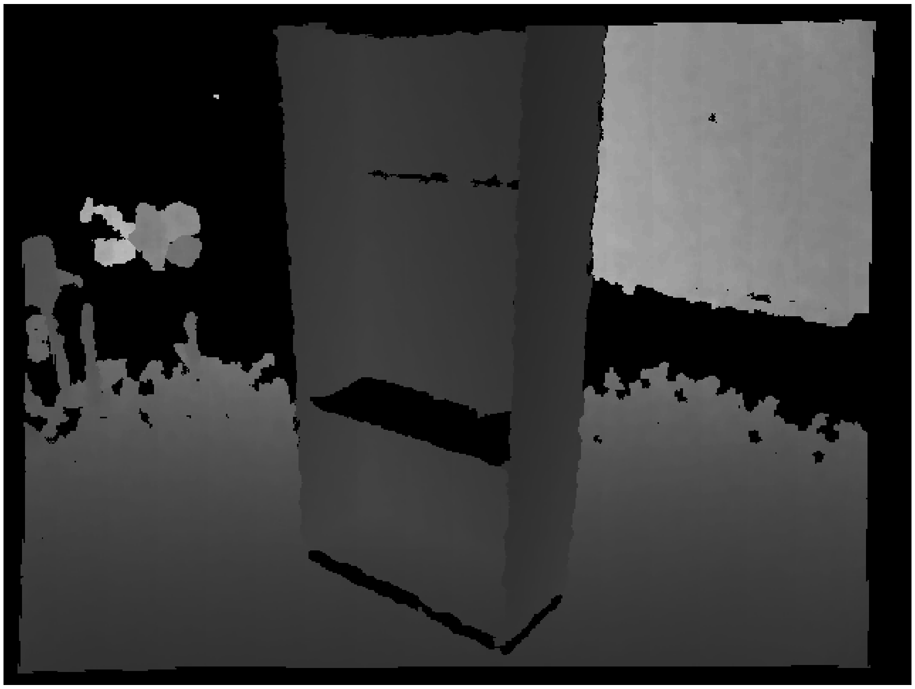

Figure 5.10: One of the depth images from the "rgbd_dataset_freiburg3_large_cabinet" dataset.[1]

For each of these 10 images, the number of correct planes is determined - for images of one side of the cabinet, this number will be 1, and where the Kinect is pointing at the corner of the cabinet, this number will be 2. The algorithm is then run on each of these images - the edge points detected are plotted, as well as the 4 corner points for each plane found by the Hough transform. Note that in Figure 5.10, there are 7 planes visible to us. The reason for the smaller number of planes being expected is that the walls are removed from the image by the edge detector, as are the less significant edges within the cabinet. Therefore, only the front and one side of the cabinet are expected to be found.

From this plot, each of the planes in the image is defined as either “correctly extracted”, “parameters extracted”, “corners extracted” or “incorrectly extracted”. “Correctly extracted” means that the plane has been correctly extracted for both the parameters and the corners of the plane. “Parameters extracted” and “corners extracted” mean that either the parameters or corners (respectively) have been correctly extracted, and finally “incorrectly extracted” means that neither the parameters nor the corners are correctly extracted by the algorithm. The time taken to run the algorithm on the image is also given.

The results for this experiment are given in Appendix C. There were only 3 correct fits - and only one of those (in Image 5), was the correct number of plane found. This is a very poor performance. In most cases 2 planes were found - this is problematic, as for most of those cases, only 1 plane was actually meant to be found. This means that there were a lot of “false” planes found in the data.

Where the correct plane was found, the plane was considered to be correct in both parameters and corners - there were no planes which were only partially found. There were 2 cases in which the algorithm ran out of memory and therefore could not complete - images 7 and 10. This is, once again, very problematic as then there are no planes at all for the robot to use.

Where planes were found, the algorithm took a very long time to find them - in the region of half a minute per image. This is also problematic for mobile robotics as the robot will probably have hit the object before the algorithm can process the image.

Therefore, based on the results above, this version of the Hough transform is not considered to be useful for obstacle avoidance in mobile robotics, mainly because the algorithm does not find the correct planes often enough. This is also a problem for SLAM, however, the next section will investigate whether, if the planes were correct, they would be trackable - which is the second requirement for SLAM.

5.5.2 Suitability of the algorithm for SLAM pre-processing

In this section, the suitability of the edge detector and the Hough transform is investigated for use as the feature extraction process for SLAM.

The first requirement (over and above those described in the previous section) is that the processing should extract the same objects in all the frames used - this means that the box must be extracted from all the frames. This has already been proved in the previous tests.

The second requirement for SLAM is that the objects can be tracked between frames. In order to prove this, we need to see whether there is enough detail in the output to allow distinct tracking between frames. For this test, two different environments will be investigated - first the environment described for the tests above - a simple cuboid in a planar environment, and then a more complex environment.

Test of object tracking with planar data

The aim of this experiment is to determine whether the process described above can extract suitable features for SLAM - that are distinct enough that they could be tracked through multiple frames.

It is expected that the features extracted by the above process will not be suitable, as they will not be distinct enough to track clearly between frames. This is because the images used are very sparse, and one side of a simple box is fairly indistinct from another side of the same box. Therefore, it will be difficult to track the features between frames.

The method used is the following. Ten images are chosen from the Sturm "rgbd_dataset _freiburg3_cabinet" dataset [1]. These will be processed using the edge detector (note: the Hough transform simply fits planes to the features output by the edge detector, and therefore has no impact on the features extracted, only on the location of those features). Then, the outputs will be compared to determine whether there are "distinct" features in the frames.

For each of the 10 images, the edges detected are shown in Figure 5.11. These images show that some edges of the cabinet are extracted in every image. However, there are no images in which all the edges are extracted correctly. There are also some images where more than just the cabinet is extracted - these are effectively noisy points.

Based on the edges extracted, these data are not suitable for SLAM as there are not enough features extracted in each image. This means that tracking the features through the images would be very difficult, because the cabinet is non-unique and has no points of reference. Non-unique means that two sides of the cabinet are indistinguishable from one another. Having no points of reference means that there are no other features in the scene which show where in the scene the camera is.

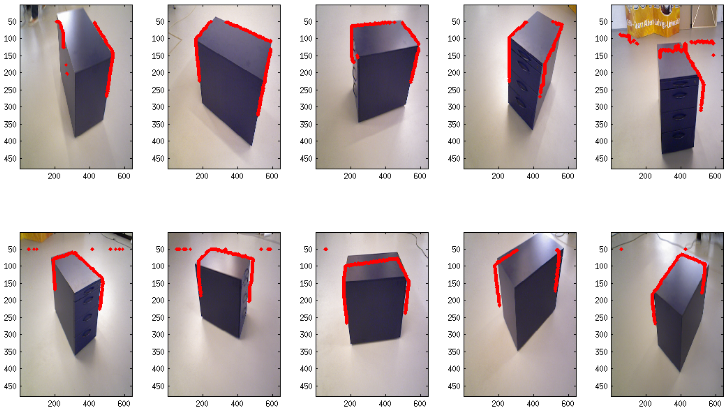

Figure 5.11: Images plotted with the extracted edges shown in red. Each of the images has some edges extracted, however there are also some images with noisy points.

The next test will investigate whether more complex data is suitable for SLAM with the Kinect.

Tests of object tracking with complex data

The experiment in this sections will determine whether the same features are segmented in successive frames in the dataset. It is very important that the same features are segmented, otherwise feature tracking is not possible.

It is expected that the process will be able to correctly segment the major features within a frame. It may not be able to segment the smaller features in the frames. The major features are the large features which stand out as being separate from the background. For example, the computer screen would be a major feature, whilst the keyboard would not (as - in a depth sense - is blends in with the background).

The images used for this experiment are images from the same online dataset as the data used above [1]. This dataset is a set of images of a desk. One of the images from the dataset is shown in Figure 5.12. On the desk are multiple objects - a computer screen, a plant, a teddy bear, and many others. There are over 3000 images in the dataset, but only 40 of them are used for the experiment in this section - the first image and then every 40th image after that image, until there are 40 images. This will gives a wide range of images.

For each of these images, the edge detection process is run on the image. The result of which features are correctly identified is recorded. The features are defined as either “segmented”, “not-segmented”, “partially segmented” or “not in image”. The reason for the last group is that most of the frames do not contain all the features in

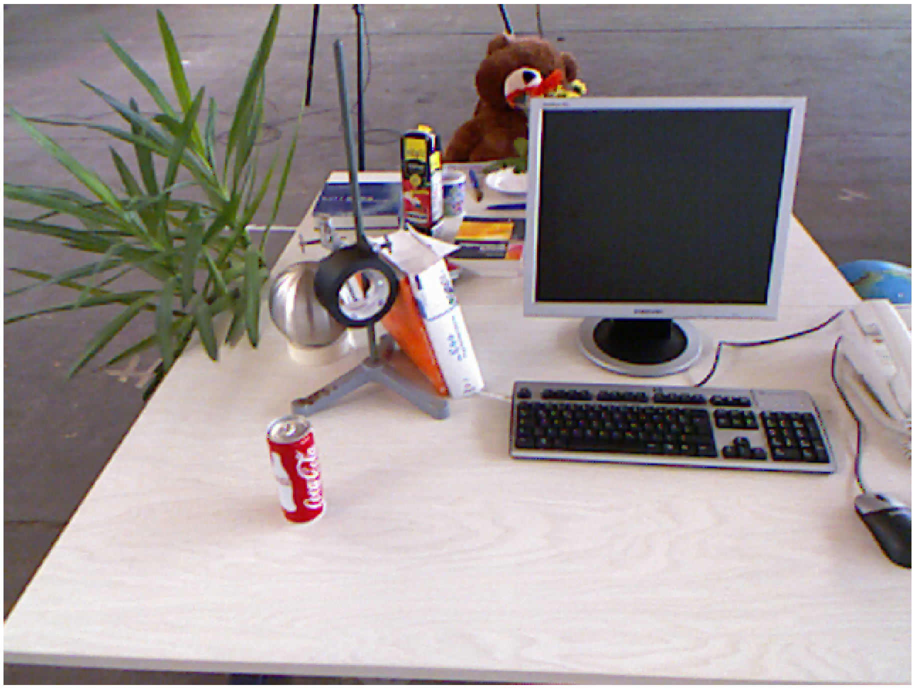

Figure 5.12: One of the RGB images from the desk dataset. Only the RGB images were used in this dissertation.

the scene (which is defined as all the features in all frames). The first 3 groups are represented in Figure 5.13. Any features which are not segmented in any frames are not in the table (for example, the keyboard).

The results of this experiment are shown in Appendix C. The results are represented in the bargraph in Figure 5.13. This graph shows that some features are very well extracted from the data, and could therefore be used for tracking the frames. These are the computer screen and the table (desk) on which everything sits. The teddy is also extracted in many frames, and could also be used for tracking, although it is not planar, and so therefore would not be useful for use with the Hough transform.

Some features are partially extracted in a number of frames - for example, the plant and the carton are partially extracted in a number of frames. Some objects are not extracted in any frames - for example the mug and the can. There is a high proportion of features not being extracted in subsequent of frames - for example, the book and lamp are not found more times than they are extracted. The globe seems to be easily segment-able, when it is in the frame (it was never not extracted when present in the frame).

The reason that the table and the screen are so easily extracted by the edge detec-

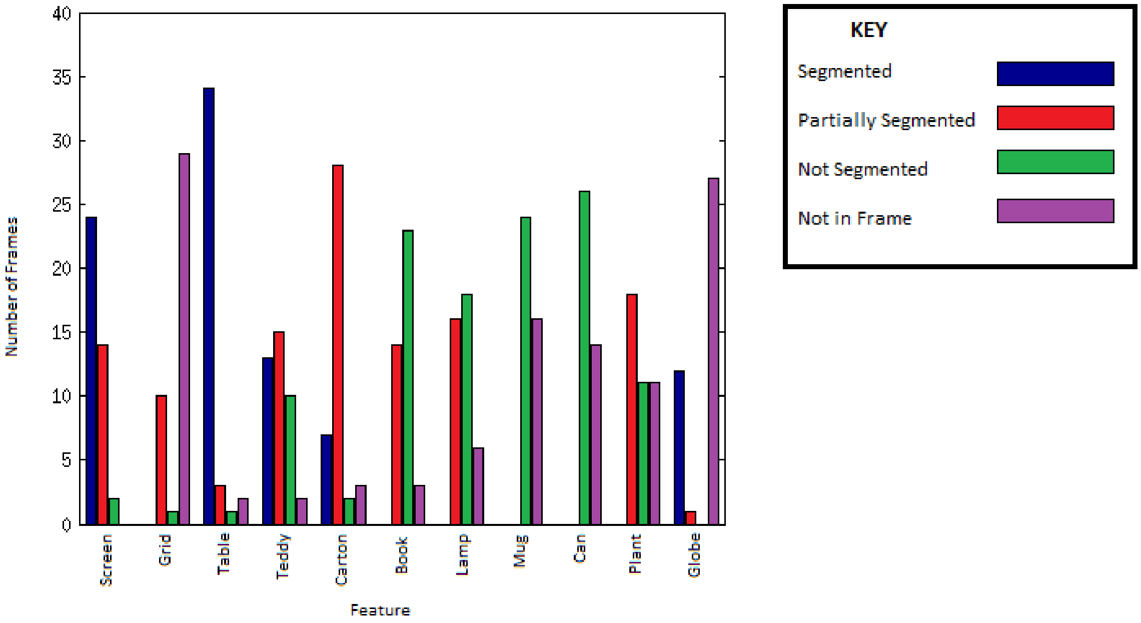

Figure 5.13: Bargraph showing the number of frames in which features are segmented, not segmented, partially segmented or not in the screen. The screen and table have high proportions of being fully segmented in frames, whilst the mug and the can are not segmented in any frames.

tor is that they are fairly large objects (in relation to the scene), and they generally do not have a close background (in other words, there is nothing behind them). This means that they are easily extracted because there is a large difference between the depth of the object (the screen or table) and the background. The same is true of the globe - although it is obstructed by the table in many frames, and therefore would not be as useful as the screen for tracking purposes.

The teddy is well extracted for the same reason as the table and the screen are well extracted - the background is fairly far away from it. It is only partially extracted most of the time, because it is a fairly complex object, and therefore is quite difficult for the algorithm to handle it. This is not a serious problem, as the teddy is not useful in this instance because it is not a planar object. It could, however, be useful for tracking algorithms.

The mug and the can are never extracted from the scene for the opposite reason that the others are extracted so often - they are small objects, and are “close” to the background. This means that the edge detector cannot detect an edge where the edge of the object (can or mug) meets to background (in this case the table).

The plant is partially extracted a fair amount - this is because it is a complex object, which is sometimes very distinct from its background, and sometimes not. It could be useful as a tracking object, although would not be useful for application to the Hough transform. The carton, on the other hand, is partially extracted a lot because sometimes it is easily distinguishable from its background (when its background is the wall), and sometimes not (when its background is the table). Most of the times it is somewhere between the two, and is therefore generally only partially extracted from the scene.

The above result is as expected - we expected that large object would be easily extractable, because they would show up as distinct object from the background. Small objects were not segmented well - this is also as expected.

The objects which were successfully extracted from the scene (the screen in particular) would be sufficient to allow tracking - especially of those objects which were segmented but are not planar (such as the teddy and the plant) are used as well). Therefore, the segmentation is considered to be acceptable for use with a tracking algorithm for SLAM.

The algorithm could, however, be improved so that it segments more of the objects correctly - this could be achieved by using the RGB data as part of the segmentation procedure.

Due to the above results, these data are not considered to be suitable for this type of algorithm, and therefore no further tests will be run for these data. Planar data which is more complex than a single box would be more suitable - as then objects could (potentially) be tracked between frames. however, due to the problems found with the algorithm, this will not be tested.

Chapter 6

Conclusions

This chapter will use the results and analysis in the previous chapters to draw conclusions about the research questions. This section will be split into multiple sections, one for each of the preceding chapters.

6.1 Conclusions about Calibration

The Calibration of the Kinect showed the following things:

1. That the calibration of the Kinect sensor using the MATLAB calibration toolbox does give different results to those given for the focal length and centre point, compared to those given by Microsoft and OpenNI. The calibration results in differences in the disparity-to-depth mapping, particularly in the disparity offset, which has a large effect on the depth value.

2. The lens distortion, although there, can be ignored. This is because, as seen in Figure 3.18, the change in the points from original to reprojected is very small, and therefore the lens distortion (which is applied during reprojection) is negligible.

3. The colour of the features in the scene do not affect the ability of the Kinect to find the correct depth.

4. The reflectiveness of the surface affects the Kinect's ability to identify the surface. For non-reflective surfaces there are no error points (points where there is no return recorded for a particular pixel). For reflective surfaces there are many error points.

The above findings mean that every Kinect needs to be individually calibrated. Although this may take time, it is very important, as the calibration values affect the output of the disparity-to-depth mapping significantly, in comparison to the mapping using the standard values given by Microsoft and OpenNI. The baseline and the disparity offset change according to the calibration parameters. However, the lens distortion can be discounted, as it does not have a large effect on the image. This means that the calibration needs only be done to determine the focal length and the centre point.

The environment in which the Kinect is operated can be in varying colours, as this does not affect the output of the Kinect. The disparity value found is unaffected by the colour of the object in view. However, care should be taken to ensure that there are no reflective objects in the scene, as this causes an increase in the error points in the Kinect output, in comparison to a non-reflective surface. This is because, with a reflective surface, the IR-light will be reflected away from the projection. Sometimes, this cannot be avoided, and in these situations, the colour camera could be used as an aid to the depth camera. The Kinect is, therefore, suitable for sparse indoor environments. If this could not be ensured, the RGB data could be used for error-checking in scenes with reflective objects.

6.2 Conclusions from the Feature Extraction

The feature extraction for 2-dimensional Laser Range Finder data showed the following:
1. Both the Split-and-Merge algorithm worked effectively on Laser Range Finder and "perfect" data.
2. The Split-and-Merge algorithm is more accurate for 2D data than the Hough Transform, as that it uses a least squares fit.
3. The Hough Transform runs slower than the Split-and-Merge algorithm.
4. It is less critical to find the correct threshold value for the Hough Transform than for the Split-and-Merge algorithm. The threshold for the split-and-merge needs to be carefully optimised, so that the correct number of lines is found, and the speed is optimised. For the Hough Transform, the threshold where the correct number of lines is found is wider, and therefore does not need to be optimised as carefully.

This means that for data which are collected using the Laser Range Finder, the Split-and-Merge algorithm generally out-performs the others, and would therefore be the better choice for use on these data. However, care needs to be taken in finding the threshold for the algorithm, as it is very susceptible to an incorrect threshold, particularly if the data are of a different scale to previous data used with the algorithm (as the threshold would then need to be changed). The Hough Transform is easier to set-up as it does not require a different threshold for different scales of data. It is also much easier to extend into 3-D, as there are existing extensions of the algorithm in the literature, which there are not for the Split-and-Merge.

6.3 Conclusions from the Hough 3D

The Hough Transform was successfully extended into 3D. Please note that all conclusions drawn here refer specifically to the Hough Transform as presented in this dissertation. These conclusions are not general to all versions of the Hough transform. Initial testing showed the following:

1. The Hough Transform is a rather slow algorithm - it takes 58 ms more per 10 point increase.

2. The Hough Transform is very susceptible to the number of points in the set. The time taken to run the algorithm is negatively affected by an increase in the number of points.

Based on the above findings, the Hough Transform is not practically suitable for use in SLAM on its own. It would simply take too long for it to run on the raw data from the Kinect sensor. Therefore, pre-processing is required to reduce the number of points that are input into the Hough Transform.

To do this, three edge detectors were investigated. The first is the Sobel edge detector, the second is the Laplacian-of-Gaussian edge detector, and the last is the Canny edge detector. These edge detectors were compared to each other on speed and susceptibility to noise. The following results were found:

1. The Sobel edge detector performs the best in terms of susceptibility to noise and speed for Kinect-type depth data.

2. Even with the Sobel detector, there is still a lot of noise present in the output. Many “edges” are detected due to the general features in the scene, rather than because true “edges” are found.

To reduce this noise, the data required pre-processing. One issue encountered was that the Kinect outputs a zero when there is an error return. This results in many edges being found, where in fact, the Kinect had returned an error. This problem was solved using a nearest-neighbour filtering technique. This worked effectively, and the Sobel Edge detector was chosen for use on the pre-processed data. The edge detector was able to reduce the number of points to around 500 as opposed to 300 000.

The Hough Transform was then tested in conjunction with the edge detector. The edge detector allowed the Hough Transform to perform well. The algorithm was able to successfully find planes in the data, and generally found the correct number of planes. The threshold was, once again, fairly important, especially because the accuracy of the output is higher for a higher threshold. Therefore, the threshold was tested. Increasing the threshold caused a significant increase in the time taken for the algorithm.

The Hough Transform was found to be very slow (taking of the order of 10 seconds to complete a segmentation). This is too slow for use on board a mobile robot, as the robot would have to be moving very slowly, or stationary for the duration of the computing of the Hough transform, otherwise Feature Tracking would become extremely difficult. However, the Hough Transform and the Edge Detector can successfully reduce the number of points that would need to be input into the SLAM algorithm from 300 000 to 4 per feature (one for each corner of each plane). The edge detector by itself reduces it to 500 points.

The reduction by the edge detector took approximately 0.15 s for a full Kinect image. The Hough Transform, reducing from 500 to 4 took 5 seconds. This means that the reduction by the Hough Transform is very slow in comparison to the reduction by the edge detector. Therefore, the Hough Transform is too inefficient to be useful, due to the processing speed. Therefore, until processing speeds increase, the Hough Transform was found not to be suitable. However, this may require further study as, in conjunction with SLAM, it may be better to reduce the number of features as much as possible.

The accuracy of the planes extracted by the Hough transform was found to be very poor. The algorithm did not find the correct plane in most cases, and did not correctly segment all planes in the scene in all but one case. In many cases, the algorithm extracted more planes than there were present in the data. This may have been caused by one of two things - an angle parameterisation which is too low, or too many noisy points. In this case, the second is the reason. The edge detector found many noisy points in the data used for this experiment - and therefore, the Hough transform found many false planes. This meant that, in terms of accuracy, the algorithm was not suitable for obstacle avoidance or SLAM. These noisy points could be reduced by looking at pre-processing algorithms other than edge detectors.

In terms of the accuracy of the plane parameters extracted, the algorithm also performed poorly. This was probably due to the low threshold. This caused the planes to be extracted in the "correct" area for the Hough transform - but due to the discrete nature of the grid, the grid block was too wide, and therefore the extracted plane was incorrect. The threshold could not be increased as this caused the memory to run out in MATLAB.

Finally, the ability of the algorithm to track objects between frames was tested. First, this was tested on simple planar data. For this, it was able to consistently extract the edges of the cabinet, however, due to the nature of the scene (sparse and simple), the object could not be tracked as the object could not be identified uniquely across frames. Then, more complex data were used. Here, the larger, more distinct object could be used for tracking. However, due to the fact that the scene was non-planar, the data was not suitable for use with the Hough transform.

Chapter 7

Recommendations

The following recommendations for future work are made, based on the results of this study:

1. Due to the small number of Kinect sensors available for this study, and the variation in the parameters found for the two Kinects, the calibration should be tested on more Kinect sensors, to get a wide range of values, to determine the variation in parameters of Kinect sensors. This would give a greater understanding of the Kinect sensor.

2. Due to the lack of studies into the extension of the Split-and-Merge algorithm into 3 dimensions, it was not used on Kinect data in this study. However, because it was found to be more accurate and faster than the Hough Transform in 2D, it should be investigated for extension into 3D. This could result in a very fast and efficient feature extractor that could be used with SLAM. The method of extension would involve using a 3-dimensional least squares algorithm.

3. Due to the fact that the Hough Transform was found to be inefficient (taking around 58ms to run on 10 points), the 3D Hough Transform should be optimised, to possibly make it more suitable for use with SLAM. This, once again, may result in a faster plane extraction that could be used with SLAM.

4. The Hough transform should also be optimised in terms of the angle parameterisation in both the ϕ and θ directions to make it more accurate.

5. Feature tracking was not investigated in this dissertation, and therefore the suitability of the Hough Transform for use with Feature Tracking for SLAM is unknown. Therefore, feature tracking should be implemented, for use with the Hough Transform, to determine further the suitability of the Hough Transform for use with SLAM.

6. The edge detectors used in this study were suitable for reducing the number of points input into the Hough Transform, but there may be other, more suitable,

methods for achieving this. Therefore, other methods for reducing the number of points input into the Hough Transform should be investigated.

7. Other Hough Transform variants should be investigated for use with SLAM. In particular, more optimised algorithms such as the Randomised and Probabilistic Hough transform should be investigated.

8. All the algorithms should be used in real-time on data collected from the Kinect (and, in the case of the 2D algorithms, Laser Range Finder), as this was not part of the scope of this study. The sensors should be placed on-board mobile robots, and the algorithms should be used for SLAM in real-time. This would complete the study of the Hough Transform for use in SLAM.

Appendix A

Calibration Tables

A.1 Tables of the calibration data used in section 3.2.2

The following tables show the data used to plot the graphs shown in section 3.2.2. The discussion of the data in this section is shown in section 3.2.2. This is the data used for the calibration discussion of Kinect 1:

Ex. No.	**f(x)**	**f(y)**	**c(x)**	**c(y)**	**K1**	**K2**	**P1**	**P2**
1	586.04062	588.61077	347.18929	274.63528	-0.1256	0.32845	0.01104	0.00942
2	580.35654	581.74003	307.29755	234.60084	-0.1523	0.44276	-0.00581	-0.01058
3	591.08345	594.11913	351.13297	280.59514	-0.11984	0.30925	0.01246	0.01064
4	574.26383	577.98529	312.22428	249.29456	-0.12737	0.44038	-0.00406	-0.00527
5	586.04077	588.6109	347.19026	274.63789	-0.1256	0.32845	0.01104	0.00942
6	589.72008	592.21311	346.69073	263.6263	-0.11594	0.30271	0.00668	0.00783
7	584.29879	586.94946	343.5802	271.32472	-0.12317	0.32238	0.01009	0.008
8	586.04082	588.61086	347.18964	274.63475	-0.12561	0.32844	0.01104	0.00942
9	586.04261	588.61277	347.19101	274.63676	-0.1256	0.32846	0.01104	0.00942
10	586.04328	588.61351	347.19116	274.6387	-0.1256	0.32846	0.01104	0.00942
11	591.70095	594.22345	347.63114	273.81487	-0.1137	0.3058	0.01105	0.01019
12	585.67866	588.84668	341.5077	269.26527	-0.12755	0.33934	0.00931	0.00576
13	586.03974	588.61011	347.19043	274.63562	-0.1256	0.32844	0.01104	0.00942
14	587.21698	589.95695	348.25934	275.76825	-0.12849	0.33126	0.01132	0.00919
15	586.04215	588.61237	347.18999	274.6383	-0.12559	0.32842	0.01104	0.00942
16	586.25206	589.35645	305.01279	242.08601	-0.14112	0.44004	-0.00723	-0.01013
17	588.81379	592.51087	332.34978	256.87289	-0.12685	0.51359	0.00197	0.00227
18	581.01682	584.69299	340.17377	282.42488	-0.15063	0.42259	0.01541	0.00921
19	591.31662	593.54194	350.68008	278.73688	-0.13235	0.35348	0.01274	0.01206
20	580.20848	584.38829	341.52624	264.11806	-0.13038	0.23902	0.00609	0.006
21	585.40346	587.86506	352.92261	286.82467	-0.11239	0.3009	0.01528	0.01455
22	578.13612	580.88494	336.2151	269.078	-0.12835	0.34018	0.00837	0.00469
23	576.19339	579.74035	345.08101	282.02773	-0.1001	0.26145	0.0097	0.00946
24	576.91532	577.68644	352.3241	268.55414	-0.17446	0.39516	0.00544	0.00663
25	586.97497	589.7856	346.35726	274.15356	-0.13039	0.34083	0.01121	0.00932
26	584.30365	586.95822	337.42387	266.3556	-0.15178	0.43633	0.00524	0.00377
27	586.91017	589.76718	345.7446	272.8214	-0.11825	0.31866	0.01002	0.00878
28	584.78003	586.21157	350.5111	265.325	-0.14584	0.28583	0.00918	0.00776
29	595.85789	597.71524	350.26248	268.50792	-0.13362	0.34194	0.00936	0.00768
30	588.91044	591.33298	349.22536	277.98125	-0.12437	0.33556	0.01148	0.0109

This table shows the average and standard deviation of the data in the table above:

Parameter	Mean [px]	Std. Dev.
Focal Length x (f(x))	585.77396	4.18750
Focal Length y (f(y))	588.42104	4.13190
Principle Point x (c(x))	341.91038	13.25127
Principle Point y (c(y))	269.38315	12.04043
Radial Distortion 1 (K1)	-0.12584	0.00846
Radial Distortion 2 (K2)	0.33953	0.04264
Tangential Distortion 1 (P1)	0.00855	0.04264
Tangential Distortion 2 (P2)	0.00678	0.00563

This table shows the data collected for the second Kinect camera:

Ex. No.	**f(x)**	**f(y)**	**c(x)**	**c(y)**	**K1**	**K2**	**P1**	**P2**
1	601.14818	603.29438	348.25047	260.35587	-0.1299	0.47597	0.01121	0.0153
2	598.36925	600.34541	342.39417	263.58544	-0.13885	0.55521	0.01305	0.01237
3	589.30884	591.65213	346.64214	258.91832	-0.10705	0.34473	0.01454	0.01413
4	608.03787	610.13935	338.86447	274.15517	-0.16591	0.42494	0.00834	0.01188
5	601.74392	603.21244	353.4863	261.38766	-0.1421	0.5686	0.01116	0.01422
6	607.35712	605.72096	287.31798	238.83865	-0.07784	0.3878	-0.00355	0.01407
7	589.14784	594.8801	360.82353	295.72639	-0.07922	0.18813	0.02063	0.02052
8	599.81426	601.69883	352.12917	261.72948	-0.12943	0.45387	0.01258	0.01751
9	598.90658	600.56504	350.14196	260.36625	-0.1414	0.49395	0.01193	0.0152
10	633.33498	630.38919	380.42537	292.0621	0.00447	0.00638	0.02619	0.03485
11	599.61263	601.5785	350.03992	261.02494	-0.13028	0.45278	0.01194	0.01543
12	596.39598	598.24233	347.14404	258.93603	-0.17191	0.66538	0.01051	0.01556
13	599.61272	601.57856	350.04008	261.02531	-0.13028	0.45277	0.01194	0.01543
14	594.26147	596.12096	349.60796	261.52336	-0.13118	0.44512	0.01009	0.01503
15	597.80258	599.69584	348.55042	262.56294	-0.18398	0.82022	0.0105	0.01482
16	591.77163	595.05329	337.12899	272.94597	-0.12321	0.42744	0.00803	0.0108
17	593.3828	594.95345	354.23431	265.37417	-0.09875	0.25935	0.01112	0.01671
18	594.3305	595.22489	356.40908	256.15737	-0.13594	0.42624	0.01198	0.01671
19	596.30302	598.82103	339.59746	261.80398	-0.14247	0.53827	0.00879	0.01089
20	604.64901	605.55793	360.47988	255.87851	-0.12578	0.43102	0.01271	0.01799
21	584.15486	595.25208	353.58019	274.62642	-0.14147	0.32144	0.00816	0.02285
22	599.61248	601.57832	350.04009	261.02528	-0.13028	0.45278	0.01194	0.01543
23	599.61312	601.57896	350.03979	261.02504	-0.13028	0.45278	0.01194	0.01543
24	591.62061	594.25271	338.58092	265.3255	-0.15238	0.53156	0.0092	0.01137
25	594.14361	595.76394	353.554	268.9726	-0.1425	0.60442	0.01421	0.01982
26	603.25945	604.10249	356.84834	251.39067	-0.1651	0.58853	0.00757	0.01589
27	599.61352	601.57944	350.03925	261.02525	-0.13028	0.45276	0.01194	0.01543
28	604.54681	606.16204	365.66167	254.9122	-0.19936	0.2279	0.0119	0.0145
29	606.83571	609.03962	347.6542	263.73787	-0.12312	0.52774	0.01188	0.01537
30	606.05485	608.12048	347.73704	264.08277	-0.11848	0.40623	0.01097	0.01384

This table shows the mean and standard deviation of these data:

Parameter	**Mean**	**Stan. Dev.**
Focal Length x (f(x))	599.49154	8.63673
Focal Length y (f(y))	601.20516	7.75155
Principle Point x (c(x))	348.91477	14.45711
Principle Point y (c(y))	263.68272	10.61077
Radial Distortion 1 (K1)	-0.13048	0.0.03642
Radial Distortion 2 (K2)	0.44597	0.15121
Tangential Distortion 1 (P1)	0.01145	0.0046
Tangential Distortion 2 (P2)	0.015968	0.004441

A.2 Tables of the data used for the comparison of parameters in section 3.2.3

The following table shows the Kinect disparity and the corresponding depth value for each of the $b * f$ and d_{off} pairs:

Kinect Disparity	**Calculated**	**Microsoft**	**OpenNI**
600	729.3469	710.2041	704.0816
650	811.6659	790.9091	784.0909
700	914.9313	892.3077	884.6154
750	1048.3	1023.5	1014.7
800	1227.2	1200	1189.7
850	1479.7	1450	1437.5
900	1863	1831.6	1815.8
950	2514.5	2485.7	2464.3
975	3047.2	3026.1	3000
1000	3866.3	3866.7	3833.3

A.3 Tables showing the data used for the comparison of black and white surfaces

The following table shows the maximum and minimum distances recorded by the Kinect for the black and white screens at each of the distances.

Actual Distance [mm]	**Maximum - Black [mm]**	**Minimum - Black [mm]**	**Maximum - White [mm]**	**Minimum - White [mm]**
800mm (1)	804	761	786	742
800mm (2)	806	761	788	743
1000mm (1)	1014	966	1020	953
1000mm (2)	1014	969	1020	948
1500mm (1)	1519	1455	1567	1431
1500mm (2)	1525	1443	1581	1413
2000mm (1)	2039	1947	2063	1915
2000mm (2)	2015	1947	2051	1915
2500mm (1)	2570	2443	2669	2426
2500mm (2)	2570	2443	2669	2409
3000mm (1)	3216	2845	3129	2893
3000mm (2)	3186	2869	3129	2893
3500mm (1)	3952	3216	3738	3341
3500mm (2)	3952	3277	3779	3277

A.4 Tables showing the data used for the reflective and non-reflective surface tests in section 3.3

This table shows the number of errors recorded for the reflective and non-reflective surfaces under high light conditions and low light conditions for each of the distances.

Distance	Reflective - High Light	Non-reflective - High Light	Reflective - Low Light	Non-reflective - Low Light
800mm (1)	19391	0	23925	0
800mm (2)	21946	0	26475	0
900mm (1)	17534	0	23571	0
900mm (2)	18182	0	22439	0
1000mm (1)	17304	0	21938	0
1000mm (2)	19227	0	22271	0
1200mm (1)	21068	0	24259	0
1200mm (2)	22755	0	23009	0
1400mm (1)	22728	0	24163	0
1400mm (2)	25838	0	24502	0
1600mm (1)	31845	0	28214	0
1600mm (2)	31855	0	28271	0
1800mm (1)	39474	0	33106	0
1800mm (2)	40423	0	33380	0
2000mm (1)	48033	0	39857	0
2000mm (2)	49439	0	39178	0
2200mm (1)	53338	0	43576	0
2200mm (2)	53873	0	42996	0
2400mm (1)	60739	0	47916	0
2400mm (2)	59324	0	48477	0
2600mm (1)	65319	0	54807	0
2600mm (2)	64842	0	52145	0
2800mm (1)	70565	0	61218	0
2800mm (2)	68612	0	58994	0
3000mm (1)	74450	0	66404	0
3000mm (2)	77151	0	66585	0
3200mm (1)	80749	0	65607	0
3200mm (2)	80548	0	66427	0
3400mm (1)	80629	0	66900	0
3400mm (2)	76911	0	69274	210
3600mm (1)	80667	1	71374	258
3600mm (2)	78995	0	72254	382

A.5 Distortion Models

The following figure shows the distortion model for Kinect 1:

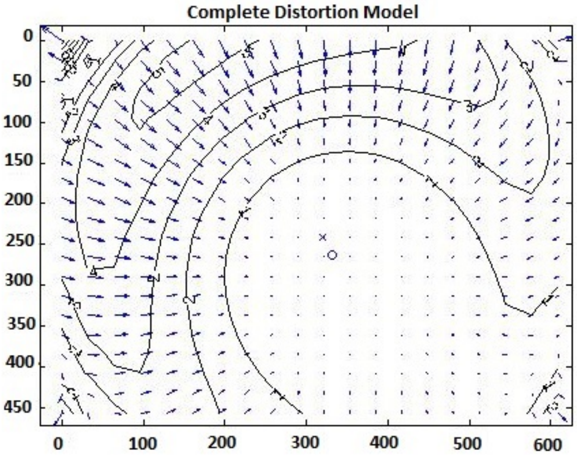

Figure A.1: Image showing the distortion model of the first Kinect IR sensor.

Appendix B

Tables of data used in chapter 4

B.1 Data for the threshhold determination in Split-and-Merge

The correct number of lines for each of the datasets is shown in the table below. The X location and Y location of the corner applies to when the box is placed at some (x, y) location in relation to the Laser Range Finder (datasets 4 - 20). Otherwise, the X location is not applicable, as the box is placed directly in front of the sensor.

Data set number	Correct number of lines	X location	Y location
1	1	N/A	1
2	1	N/A	2
3	1	N/A	3
4	2	1	1
5	2	1	3
6	2	2	1
7	2	2	2
8	2	2	3
9	2	3	1
10	2	3	2
11	1	3	3
12	2	-1	1
13	2	-1	2
14	2	-1	3
15	2	-2	1
16	2	-2	2
17	2	-2	3
18	2	-3	1
19	2	-3	2
20	1	-3	3

The table below shows whether the the value of the threshhold results in the correct number of lines, or greater or fewer. The total number of correct(=), greater (>) or fewer (<) lines is given in the last two lines of the table (either classified as the correct number of lines or an incorrect number of lines).

Threshhold value/ Dataset Number	350	325	300	275	250	225	125	100	75	50	35	25	20	15
1	=	=	=	=	=	=	=	=	=	=	=	=	=	>
2	=	=	=	=	=	=	=	=	=	=	=	=	=	=
3	=	=	=	=	=	=	=	=	=	=	=	=	=	=
4	<	<	<	=	=	=	=	=	=	=	=	=	>	>
5	<	<	<	<	=	=	=	=	=	=	=	=	>	>
6	<	<	=	=	=	=	=	=	=	=	>	>	>	>
7	<	<	<	=	=	=	=	=	=	=	=	>	>	>
8	<	<	<	<	=	=	=	=	=	=	>	>	>	>
9	<	<	=	=	=	=	=	=	=	>	>	>	>	>
10	<	<	<	=	=	=	>	>	>	>	>	>	>	>
11	=	=	=	=	=	=	=	=	=	=	=	=	=	=
12	<	<	<	<	=	=	=	=	=	=	=	=	=	>
13	<	<	=	=	=	=	=	=	=	=	=	=	=	=
14	<	<	<	=	=	=	=	=	=	=	=	=	=	>
15	<	<	=	=	=	=	=	=	=	=	=	>	>	>
16	<	<	<	=	=	=	=	=	=	=	>	>	>	>
17	<	<	<	<	=	=	=	=	=	=	=	>	>	>
18	<	=	=	=	=	=	=	=	=	=	=	=	>	>
19	<	<	=	=	=	=	=	=	=	=	=	>	>	>
20	=	=	=	=	=	=	=	=	=	=	=	=	=	=
No. Correct	5	6	11	16	20	20	19	19	19	18	15	11	8	5
No. Incorrect	15	14	9	4	0	0	1	1	1	2	5	9	12	15

B.2 Data for speed test of Split-and-merge algorithm

This section shows all the data used for the speed test of the split-and-merge algorithm.

This table shows the number of points for each of the datasets. It also shows the time taken to run the algorithm for each dataset, for each of the different threshhold values. These test were performed in MATLAB 2011b (for Linux) on a Sony Vaio Laptop with the following specifications:
Operating System: Ubuntu 12.04 64-bit

Processor: Intel Core i7-2640M CPU 2.80GHz x 4
RAM: 3.8 GiB

Set no.	T/h No.pts	250	225	200	175	150	125	100	75	Ave	S.Dev
1	68	0.0113	0.0121	0.0122	0.0118	0.0119	0.0116	0.0126	0.0115	0.0119	0.0004
2	34	0.0115	0.0112	0.0110	0.0116	0.0113	0.0127	0.0113	0.0112	0.0115	0.0005
3	23	0.0114	0.0110	0.0107	0.0115	0.0107	0.0108	0.0108	0.0108	0.0110	0.0003
4	59	0.0481	0.0474	0.0465	0.0464	0.0483	0.0461	0.0467	0.0476	0.0472	0.0008
5	23	0.0459	0.0456	0.0464	0.0468	0.0448	0.0459	0.0457	0.04612	0.0459	0.0005
6	40	0.0463	0.0467	0.0468	0.0552	0.0460	0.0468	0.0459	0.0458	0.0475	0.0003
7	31	0.0463	0.0464	0.0457	0.0471	0.0467	0.0472	0.0473	0.0475	0.0468	0.0006
8	25	0.0465	0.0461	0.0459	0.0463	0.0450	0.0469	0.0503	0.0461	0.0467	0.0015
9	28	0.0466	0.0467	0.0459	0.0488	0.0517	0.0463	0.0461	0.0464	0.0474	0.0019
11	22	0.0114	0.0116	0.0114	0.0108	0.0107	0.0110	0.0109	0.0114	0.0112	0.0003
12	61	0.0547	0.0475	0.0474	0.0476	0.0478	0.0467	0.0485	0.0472	0.0485	0.0025
13	37	0.0476	0.0488	0.0465	0.0474	0.0498	0.0475	0.0480	0.0467	0.0478	0.0011
14	25	0.0523	0.0461	0.0467	0.0465	0.0462	0.0461	0.0470	0.0478	0.0474	0.0020
15	41	0.0469	0.0495	0.0469	0.0470	0.0471	0.0538	0.0481	0.0473	0.0484	0.0023
16	33	0.0488	0.0456	0.0462	0.0500	0.0467	0.0466	0.0478	0.0471	0.0474	0.0014
17	24	0.0491	0.0480	0.0490	0.0472	0.0484	0.0468	0.0465	0.0467	0.0478	0.0010
18	30	0.0474	0.0483	0.0481	0.0470	0.0460	0.0477	0.0462	0.0489	0.0475	0.0010
19	27	0.0477	0.0450	0.0472	0.0469	0.0461	0.0461	0.0471	0.0461	0.0466	0.0008
20	22	0.0115	0.0114	0.0111	0.0119	0.0120	0.0133	0.0113	0.0112	0.0118	0.0007
Ave		0.0385	0.0377	0.0375	0.0384	0.0378	0.0379	0.0379	0.0376		
S. D.		0.017	0.016	0.016	0.016	0.016	0.016	0.016	0.016		

The following table shows the data used for the tests of the effect of the number of data points on the time taken to run the algorithm. Each of the test is run 5 times (to obtain an average time elapsed).

Number pts in set	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
50	0.010928	0.011300	0.011939	0.010463	0.011734	0.0113
250	0.017402	0.022857	0.017412	0.018252	0.019956	0.0192
500	0.021076	0.018051	0.025184	0.024006	0.024488	0.0226
750	0.033693	0.028373	0.029393	0.031743	0.034325	0.0315
1000	0.037967	0.043282	0.034394	0.039287	0.039773	0.0389
1500	0.054541	0.049696	0.054049	0.050404	0.048996	0.0515
2000	0.074112	0.070112	0.092851	0.083258	0.072628	0.0786

B.3 Data for the angle discretisation test for the Hough Transform

The table below shows the time taken to run the Hough Transform algorithm for various different values of the angle discretisation.

Angle discretisation	**Time 1**	**Time 2**	**Time 3**	**Ave Time**
0.25	0.07724	0.07705	0.79979	0.0781
0.5	0.04591	0.04695	0.04722	0.0467
1	0.03202	0.03541	0.03617	0.0345
2	0.02519	0.02519	0.03052	0.0270
3	0.02632	0.02329	0.02519	0.0249
4	0.02162	0.02173	0.02424	0.0225

B.4 Data for grid size optimisation in Hough Transform

The table below shows whether the size of the grid results in the correct number of lines, or greater or fewer. The total number of correct(=), greater (>) or fewer (<) lines is given in the last two lines of the table (either classified as the correct number of lines or an incorrect number of lines). There is also a category for ”incorrect fit” - a line that is simply not representative of the data. This is given by ”i”.

Grid Size Dataset Number	10	20	30	40	50	60	70	80	90	100	110	125	135	150
1	i	=	=	>	>	>	>	>	>	>	>	>	>	>
2	i	=	=	=	=	=	=	=	=	=	=	>	>	>
3	i	=	=	=	=	=	>	>	>	>	>	>	>	>
4	i	<	<	<	=	=	=	=	>	>	>	>	>	>
5	<	<	<	<	=	=	=	=	=	=	=	=	=	>
6	<	<	i	i	=	=	=	=	=	=	=	=	>	>
7	<	i	i	i	=	=	=	=	=	=	=	=	>	>
8	<	<	i	i	=	=	=	=	=	=	=	=	=	>
9	<	i	i	i	i	=	=	=	=	=	=	=	=	>
10	<	<	i	i	i	i	=	=	=	=	=	=	=	>
11	i	i	i	i	i	=	=	=	=	=	=	=	=	=
12	i	i	i	i	=	=	=	=	=	>	>	>	>	>
13	<	i	i	=	=	=	=	=	=	=	=	=	=	=
14	<	i	i	i	=	=	=	=	=	=	=	=	=	=
15	<	i	i	i	i	i	=	=	=	=	=	>	>	>
16	<	i	i	i	i	=	=	=	=	=	=	>	>	>
17	<	<	i	=	=	=	=	=	=	=	=	=	=	>
18	<	i	i	i	i	=	=	=	=	=	=	=	=	=
19	<	<	i	i	i	i	i	=	=	=	=	=	=	=
20	i	i	i	i	i	i	=	=	=	=	=	=	=	=
No. Correct	0	3	3	4	11	15	17	18	17	16	16	13	11	6
No. Incorrect	20	17	17	16	9	5	3	2	3	4	4	7	9	14

The following tables show the data used for the speed analysis of the Hough Transform. The data is generated is in a striaght line at 45 degrees. The number of points is varied from 10 to 410 in 20 point increments. The average across each dataset and across the grid number is shown in the last column and last row (respectively).

Data-	G. No	50	60	70	80	90	100	110	Ave
set	No.points								
1	10	0.03592	0.03774	0.03724	0.04089	0.03847	0.04040	0.04003	0.0387
2	30	0.04287	0.04482	0.04752	0.04717	0.04936	0.05179	0.05229	0.0480
3	50	0.04927	0.05169	0.05354	0.06019	0.05998	0.06359	0.06434	0.0575
4	70	0.05419	0.06269	0.06195	0.06513	0.06796	0.07195	0.07572	0.0657
5	90	0.05796	0.06341	0.06709	0.07614	0.07755	0.08246	0.08682	0.0731
6	110	0.06389	0.07161	0.07646	0.08333	0.08767	0.09327	0.09796	0.0820
7	130	0.07242	0.07832	0.08622	0.09154	0.09952	0.10696	0.11028	0.0922
8	150	0.07839	0.08716	0.09248	0.09921	0.11282	0.11449	0.12359	0.1012
9	170	0.08347	0.09241	0.10167	0.10454	0.11279	0.12689	0.13285	0.1078
10	190	0.08943	0.09923	0.10562	0.11015	0.12174	0.13637	0.14530	0.1154
11	210	0.09378	0.10027	0.11061	0.12067	0.13172	0.13638	0.14919	0.1204
Ave		0.0656	0.0718	0.0764	0.0817	0.0872	0.0931	0.0980	

The following table shows the data collected for the speed tests across datasets where more than one line is fitted. The number of lines fitted is varied to the following numbers (by adding more horizontal lines to the data, but not changing the total number of points - which is 500) 1, 2, 4, 6, 8 and 10. Each test is run 3 times, to obtain an average. Note that the gridsize used is 100.

Number of points	Test 1	Test 2	Test 3	Average
1	0.30454	0.29912	0.30487	0.3028
2	0.32351	0.31552	0.31917	0.3194
4	0.32625	0.35315	0.33235	0.3372
5	0.33917	0.34115	0.33638	0.3389
6	0.32874	0.33759	0.34372	0.3367
8	0.34101	0.35829	0.35207	0.3505
10	0.38131	0.37446	0.37679	0.3775

Appendix C

Tables of data used in Chapter 5

C.1 Data for the speed test of the Hough 3D algorithm without pre-processing

In this section, the data used for testing the Hough Transform without pre-processing is presented. The table below shows the data used for the speed testing on different numbers of points for the Hough Transform. Each set of points is contained on a single plane, but has a different number of points. The scale and orientation of the plane are kept constant. Each test is run 5 times, and the final column in the table gives the average value.

No. points	Test 1	Test 2	Test 3	Test 4	Test 5	Average
9	0.078244	0.074374	0.075106	0.085454	0.073411	0.0765
16	0.122961	0.120213	0.115418	0.119475	0.117022	0.1190
25	0.183623	0.186131	0.180537	0.178552	0.177216	0.1812
36	0.235600	0.234266	0.241445	0.242460	0.244232	0.2398
49	0.311320	0.324586	0.316869	0.326224	0.312841	0.3184
64	0.399005	0.408272	0.394926	0.402125	0.411927	0.4033
81	0.504059	0.484546	0.511512	0.506549	0.514516	0.5042
100	0.624370	0.614677	0.599386	0.597584	0.613164	0.6098
121	0.742740	0.714353	0.717057	0.732673	0.716942	0.7248

C.2 Data for the testing of the edge detectors

The following table contains the speed and accuracy data for the Sobel, LoG and Canny edge detectors. Each test is run 10 times. The speed at which the algorithm ran is recorded in seconds.

Test no.	Sobel	LoG	Canny
1	0.153060	0.183715	0.292605
2	0.148045	0.191071	0.284494
3	0.151892	0.190805	0.281146
4	0.150328	0.187574	0.291359
5	0.147206	0.199866	0.289759
6	0.156228	0.191144	0.278600
7	0.153543	0.191304	0.290804
8	0.158614	0.198658	0.279902
9	0.155451	0.199178	0.285761
10	0.147959	0.208837	0.278112

The following tables show the results of the tests of the 3-D Hough Transform. The first table shows whether the Hough Trasnform gives the correct result for each of the different threshholds. The threshholds used are 3, 5, 7, 9, 11, 13, 15 and 17. Each set should result in only one plane being output

Set no./Threshhold	3	5	7	9	11	13	15	17
1	=	=	=	=	=	=	=	=
2	=	=	=	=	=	=	=	=
3	=	=	=	=	=	=	=	=
4	=	=	=	=	=	=	=	=
5	=	=	=	=	=	=	>	>
6	=	=	=	=	=	=	>	>
7	=	=	=	=	=	=	>	>
8	=	=	=	=	=	=	=	=
9	=	=	=	=	=	=	=	=
10	=	=	=	=	>	>	>	>
11	=	=	=	=	=	>	>	>

The following table shows the speed of the algorithm depending on the threshhold. The time taken is given in seconds. The number of points in the set is also given. Note that the time taken is only given for the threshholds where the correct number of planes is found.

T/H	**No.**	3	5	7	9	11	13	15	17
Set no.	**pts**								
1	399	5.0011	5.8879	6.6115	7.4638	8.3126	9.1233	9.9444	10.7598
2	386	4.7766	5.6046	6.3920	7.2228	8.0232	8.7592	9.5556	10.4724
3	493	6.0465	7.1444	8.1782	9.2612	10.2478	11.2418	12.3019	13.2234
4	527	6.5613	7.6802	8.8018	9.8487	10.9269	11.9829	13.1688	14.2508
5	485	5.9412	6.9448	8.0398	9.0797	10.0206	10.9297	N/A	N/A
6	467	5.5920	6.7459	7.7256	8.7629	9.7069	10.8354	N/A	N/A
7	490	6.0213	7.1189	8.1081	9.0932	10.2133	11.1001	N/A	N/A
8	517	6.3666	7.5104	8.5952	9.6259	10.7788	11.6912	12.7925	13.8157
9	507	6.1629	7.3845	8.3614	9.3824	10.6415	11.5292	12.7092	13.7189
10	597	7.2663	8.6850	9.9711	11.2104	N/A	N/A	N/A	N/A
11	567	6.8653	8.2802	9.4901	11.6019	11.8245	N/A	N/A	N/A

C.3 Data collected for the obstacle avoidance tests

The following table shows the correct number of planes for each image, as well as the number of "correct", "parameters correct", "corners correct" and "incorrect" planes. a "correct" plane means that both the parameters and the corners are close enough to the actual data to be useful for obstacle avoidance. A plane is defined as "parameters correct" when the parameters of the plane are correct, but the corners extracted are not. Finally, "corners correct" means that the corners are correctly extracted but the parameters are not. Where N/A is recorded as the time, the memory limit in MATLAB was reached.

Image	Actual	Correct	Parameters	Corners	Incorrect	Time taken
1	2	1	0	0	1	28.37
2	1	0	0	0	2	24.59
3	1	0	0	0	2	19.56
4	2	1	0	0	1	37.19
5	1	1	0	0	1	25.99
6	2	0	0	0	2	17.22
7	1	0	0	0	1	N/A
8	1	0	0	0	2	33.25
9	1	0	0	0	2	19.27
10	2	0	0	0	2	N/A

C.4 Data collected for the tests of the Hough Transform on Real World Data

The following table shows the features which are segmented from each of the 92 frames. Each feature is "Fully segmented" (F), "Not segmented" (N), "Partially segmented" (P), or ""Not in the fram" (-).

Feature	Screen	Grid	Table	Teddy	Carton	Book	Lamp	Mug	Can	Plant	Globe
Frame 1	F	P	F	N	P	P	N	N	-	-	-
Frame 2	F	P	F	F	P	P	N	N	N	-	-
Frame 3	F	-	F	N	P	P	N	N	N	P	-
Frame 4	F	-	N	N	P	P	N	-	N	P	-
Frame 5	F	-	F	P	P	N	P	-	N	P	-
Frame 6	F	-	P	P	F	N	P	-	N	P	-
Frame 7	P	-	P	F	P	N	P	-	N	P	-
Frame 8	P	-	F	F	N	N	P	-	N	P	-
Frame 9	P	-	F	F	N	N	P	-	N	P	-
Frame 10	P	-	F	F	P	N	P	-	N	P	-
Frame 11	P	-	F	F	F	N	P	-	N	P	-
Frame 12	F	-	F	F	P	N	P	-	N	P	-
Frame 13	F	-	P	P	P	N	P	-	N	P	-
Frame 14	F	-	F	-	F	P	N	N	N	P	-
Frame 15	F	N	-	P	P	P	N	N	N	N	-
Frame 16	F	P	F	F	P	P	N	N	-	-	-
Frame 17	F	P	F	P	P	N	-	N	-	-	-
Frame 18	F	P	F	P	-	-	-	N	-	-	-
Frame 19	F	P	F	N	-	-	-	N	-	-	-
Frame 20	F	P	F	N	-	-	-	N	-	-	-
Frame 21	F	P	F	N	F	N	-	N	-	-	-
Frame 22	F	P	F	N	F	P	-	N	-	-	-
Frame 23	F	P	F	N	F	P	N	N	-	-	-
Frame 24	F	-	F	N	P	P	N	N	N	-	-
Frame 25	F	-	F	N	P	P	P	N	N	P	-
Frame 26	F	-	-	P	P	N	P	N	N	P	-
Frame 27	P	-	F	F	P	N	P	N	N	N	F
Frame 28	P	-	F	F	P	N	P	N	N	N	F
Frame 29	P	-	F	P	P	N	N	-	N	N	F
Frame 30	P	-	F	P	P	N	N	-	-	N	F
Frame 31	P	-	F	P	P	N	N	-	-	N	F
Frame 32	P	-	F	P	P	N	N	-	-	N	F
Frame 33	N	-	F	P	P	N	N	-	-	N	F
Frame 34	N	-	F	P	P	N	N	-	-	N	F
Frame 35	P	-	F	P	P	N	N	N	N	N	F
Frame 36	P	-	F	P	P	N	N	N	N	N	F
Frame 37	P	-	F	F	P	N	N	N	N	P	F
Frame 38	F	-	F	F	P	P	P	N	N	P	F
Frame 39	F	-	F	F	P	P	P	N	N	P	P
Frame 40	F	-	F	-	F	P	P	N	N	P	-
Total F	24	0	34	13	7	0	0	0	0	0	12
Total P	14	10	3	15	28	14	16	0	0	18	1
Total N	2	1	1	10	2	23	18	24	26	11	0
Total -	0	29	2	2	3	3	6	16	14	11	27

Appendix D

Code

D.1 Code for the 2-D Hough Transform

D.2 Code for the 3-D Hough Transform

Bibliography

[1] J. Sturm, N Engelhard, F. Endres, W. Burgard and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems”, in *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2012.

[2] B. Lange, C. Chang, E. Suma, B. Newman, A. Rizzo and M. Bolas, “Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor”, in *Proceedings of the 33rd Annual International Conference of the IEEE EMBS*, Boston, Massachusetts, USA, Aug-Sept 2011, pp. 1831 - 1834.

[3] A. Da Gama, T. Chaves, L. Figueiredo and V. Teichrieb, “Poster: Improving motor rehabilitation process through a natural interaction based system using Kinect sensor”, *2012 IEEE Symposium on 3D User Interfaces (3DUI)*, pp.145 - 146, 2012.

[4] C. Chang, B. Lange, M. Zhang, S. Koenig, P. Requejo, N. Somboon, A. Sawchuk and A. Rizzo, “Towards pervasive physical rehabilitation using the Microsoft Kinect”, in *2012 6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth) and Workshops*, pp. 159 - 162, 2012.

[5] Y. Chang, S. Chen and J. Huang, “A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities”, *Research in Developmental Disabilities*, vol. 32, pp. 2566 - 2570, 2011.

[6] T. Dutta, “Evaluation of the Kinect sensor for 3-D kinematic measurement in the workplace”, *Applied Ergonomics*, vol. 43, no. 4, pp. 645 - 649, 2012.

[7] H. M. Hondori, M. Khademi and C. V. Lopes, “Monitoring intake gestures using sensor fusion (Microsoft Kinect and inertial sensors for smart home tele-rehab setting”, in *1st Annual IEEE Healthcare Innovation Conference of the IEEE EMBS*, pp. 36 - 39, Houston, Texas, USA, November 2012.

[8] I. Oikonomidis, N. Kyriazis and A. A. Argyros, “Efficient model-based 3D tracking of hand articulations using Kinect”, in *BMVC*, vol. 1, no. 2, pp. 3, 2011.

[9] Z. Ren, J. Meng, J. Yuan and Z. Zhang, “Robust hand gesture recognition with Kinect sensor”, in *Proceedings fo the 19th ACM International conference on Multimedia*, pp. 759 - 760, Nov. 2011

[10] N. Villaroman, D. Rowe and B. Swan, "Teaching natural user interaction using OpenNI and the Microsoft Kinect sensor", in *Proceedings of the 2011 ACM conference on Information technology education*, pp. 227 - 232, Oct. 2011.

[11] V. Frati and D. Prattichizzo, "Using Kinect for hand tracking and rendering in wearable haptics", In *2011 IEEE World Haptics Conference (WHC)*, pp. 317 - 321, June 2011.

[12] M. Tölgyessy and P. Hubinsky, "The Kinect sensor in robotics education", in *Proceedings of the 2nd International Conference on Robotics in Education*, pp. 143 - 146, 2010.

[13] H. J. Hsu, "The potential of Kinect in education", *International Journal of Information and Education Technology*, vol. 1, no. 5, pp. 365 - 370, Dec. 2011.

[14] P. Benavidez and M. Jamshidi, "Mobile robot navigation and target tracking system", in *Proceedings of the 2011 6th International Conference on System of Systems Engineering*, pp. 299-304, Albuquerque, New Mexico, June 2011.

[15] C. Park, S. Kim, D. Kim and S. Oh, "Comparison of plane extraction performance using laser scanner and Kinect", in *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pp. 153 - 155, Incheon, Korea, Nov. 2011.

[16] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation", in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, pp. 1697-1702, Saint Paul, Minnesota, May 2012.

[17] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers and W. Burgard, "An evaluation of the RGB-D SLAM system", in *Proceedings of the 2012 International Conference on Robotics and Automation*, pp. 1691 - 1696, Saint Paul, Minnesota, May 2012.

[18] J. Stowers, M. Hayes and A. Bainbridge-Smith, "Altitude control of a quadrotor helicopter using a depth map from Microsoft Kinect sensor", in *Proceedings of the 2011 IEEE International Conference on Mechatronics*, pp. 358 - 362, Instanbul, Turkey, April 2011.

[19] R. El-Laithy, J. Huang and M. Yeh, "Study on the use of Microsoft Kinect for robotics applications", in *2012 IEEE/ION Position Location and Navigation Syposium (PLANS)*, pp. 1280 - 1288, 2012.

[20] P. Khandelwal and P. Stone, "A low cost ground truth detection system for RoboCup using the Kinect", in *RoboCup 2011: Robot Soccer Wrold Cup XV*, pp. 515 - 527, Springer Berlin Heidelberg, 2012.

[21] L. Cheng, Q. Sun, Y. Cong and S. Zhao, "Design and implementation of human-robot interactive demonstration system based on Kinect", in *2012 24th Chinese Control and Decision Conference (CCDC)*, pp. 971 - 975, May 2012.

[22] N. Chen, Y. Hu, J. Zhang and J. Zhang, "Robot Learning of everyday object manipulation using Kinect", in *Foundations and Practical Applications of Cognitive Systems and Information Processing*, pp. 665 - 647, Springer Berlin Heidelberg, 2014.

[23] N. Ganganath and H. Leung, "Mobile robot localization using odometry and kinect sensor", in *2012 IEEEInternational Conference on Emerging Signal Processing Applications (ESPA)*, pp. 91 - 94, Jan. 2012.

[24] P. Henry, M. Krainin, E. Herbst, X. Ren and D. Fox, "RGB-D mapping: using Kinect-style depth cameras for dense 3D modeling of indoor environments", *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647 - 663, February 2012.

[25] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of Kinect depth data for indoor mapping applications", *Sensors*, vol. 12, no. 2, pp. 1437 - 1454, Feb. 2012.

[26] P. Rakprayoon, M. Ruchanurucks and A. Coundoul, "Kinect-based obstacle detection for manipulator", *IEEE/SICE International Symposium on System Intergration (SII)*, pp.68 - 73, 2011.

[27] Z. Zhang, "Microsoft Kinect sensor and its effect", *Multimedia*, IEEE, vol. 19, no. 2, pp. 4 - 10, 2012.

[28] J. Smisek, M. Jancosek and T. Pajdla, "3D with Kinect", *Consumer Depth Cameras for Computer Vision*, Springer London, pp. 3 - 25, 2013.

[29] R. Macknojia, A. Chávez-Aragón, P. Payeur and R. Laganière, "Experimental characterisation of two generations of Kinect's depth sensors", in *2012 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pp. 150 -155, 2012.

[30] H. Surmann, A. Nüchter, J. Hertzberg, "An autonomous mobile robot with a 3D laser range finder fo 3D exploration and digitization of indoor environments", *Robotics and Autonomous Systems*, vol. 45, pp. 181 - 198, 2003.

[31] A. Nasir, C. Hille, H. Roth, "Plane extraction and map building using a Kinect equipped mobile robot", in *Workshop on Robot Motion Panning, Online, Reactive and in Real-time, 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 7 - 12, Algarve, Portugal, October 2012.

[32] V. Nguyen, A. Martinelli, N. Tomatis and R. Siegwart, “A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics”, in *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1929 - 1934, Alberta, Canada, Aug. 2005.

[33] A. Rosenfeld, “Picture processing by computer”, *ACM Computing Surveys (CSUR)* vol. 1, no. 3, pp. 147 - 176, 1969.

[34] P.V.C. Hough, “Method and means for recognizing complex patterns”, U.S. *3 069 654*, Dec. 18, 1962.

[35] R.O. Duda and P.E. Hart, “Use of the Hough transformation to detect lines and curves in pictures”, *Communications of the ACM*, vol. 15, no. 1, pp. 11 - 15, Jan. 1972.

[36] C. Kimme, D. Ballard, J. Slansky, “Finding circles by an array of accumulators”, *Communications of the ACM*, vol. 18, no. 2, pp. 120 - 122, Feb 1975

[37] P. M. Merlin and D. J. Farber, “A parallel mechanism for detecting curves in pictures”, *IEEE Transactions on Computers*, vol. 24, no. 1, pp. 96 - 98, Jan 1975.

[38] D.H. Ballard, “Generalising the Hough Transform to detect arbitrary shapes”, *Pattern recognition*, vol. 13, issue 2, pg. 111 - 122, 1981.

[39] C.M. Brown, “Inherent bias and noise in the Hough Transform”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, pp. 493 - 505, Sept. 1983.

[40] J. Illingworth and J. Kittler, “A survey of the Hough Transform”, *Computer Vision, Graphics and Image Processing*, vol. 44, no. 1, pp. 87 - 116, 1988.

[41] L. Xu, E. Oja and P. Kultanen, “A new curve detection method: Randomised Hough Transform (RHT)”, *Pattern Recognition Letters*, vol. 11, no. 5, pp. 331 - 338, May 1990.

[42] D. Borrmann, J. Elsberg, K. Lingemann and A. Nüchter, “The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design”, *3D Research*, vol. 2, no. 2, pp. 1 - 13, 2011.

[43] N. Kiryati, Y. Eldar and A. M. Bruckstein, “A probabilistic Hough Transform”, *Pattern Recognition*, vol. 24, no. 4, pp. 303 - 316, 1991.

[44] A. Ylā-Jāāski, N. Kiryati, “Adaptive termination of voting in probabilistic circular Hough Transform”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 9, pp. 911 - 915, 1994.

[45] L.A.F Fernandes, M.M. Oliveira, “Real-time line detection through an improved Hough transform voting scheme”, *Pattern Recognition*, vol. 41, no. 1, pp. 299 - 314, 2008.

[46] J. Overby, L. Bodum, E. Kjems and P.M.Ilsøe, “Automatic 3D building reconstruction from airborne laser scanning and cadastral data using the Hough Transform”, *Int. Arch. of Photogrammetry and Remote Sensing*, vol. 35, no. B3, pp. 296 - 301, 2004.

[47] G. Vosselman and E. Dijkman, “3D building model reconstruction from point clouds and ground planes”, *Int. Arch. of Photogrammetry and Remote Sensing*, vol. 34, no. 3/W4, pp. 37 - 43, 2001.

[48] D. Dube and A. Zell, “Real-time plane extraction from depth images with the randomised Hough Transform”, *2011 IEEE International Conference on Computer Vision Workshops*, pp. 1084-1091, 2011.

[49] R. Hulik, M. Spanel, P. Smrz and Z. Materna, “Continuous plane detection in point-cloud data based on 3D Hough transform”, *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 86 - 97.

[50] M. Camurri, R. Vezzani and R. Cucchiara, “3D Hough tranform for sphere recognition on point clouds.” *Machine Vision and Applications*, vol. 25, no. 7, pp. 1877 - 1891, 2014.

[51] R.-C. Dumitru, D. Borrmann and A. Nüchter, “Interior reconstruction using the 3-D Hough Transform”, *Int. Arch. of the Photogrammetry, Remoted Sensing and Spatial Information Sciences*, vol. 40, Feb. 2013.

[52] O.J. Woodford, M.-T. Pham and A. Maki, “Demisting the Hough Transform for 3D shape recognition and registration”, *International Journal of Computer Vision*, vol. 106, no. 3, pp. 332 - 341, 2013.

[53] D.H. Ballard and J. Slansky, “A ladder-structured decision tree for recognising tumors in chest radiographs”, *IEEE Transactions on Computers*, vol. 100, no. 5, pp. 503 - 513, May 1976.

[54] H. Wechsler and J. Slansky, “Finding the rib cage in chest radiographs”, *Pattern Recognition*, vol. 9, no. 1, pp. 21 - 30, 1977.

[55] F. Zana and J.C. Klein. “A multimodal registration algorithm of eye fundus images using vessels detection and Hough Transform”, *IEEE Transactions on Medical Imaging*, vol. 18, no. 5, pp. 419 - 428, May 1999.

[56] M.E. Brummer, “Hough Transform detection of the longitudinal fissure in tomographic head images”, *IEEE Transactions on Medical Imaging*, vol. 10, no. 1, pp. 74 - 81, Mar. 1991.

[57] P. Jensfelt, H.I.Christensen, “Laser based position aquisition and tracking in an indoor environment”, *International Symposium on Robotics and Automation - ISRA*, vol. 98, 1998.

[58] S. T. Pfister, S.I Roumeliotis and J.W. Burdick, “Weighted line fitting algorithms for mobile robot map building and efficient data representation”, in *Proceedings of IEEE International Conference on Robotivs and Automation, ICRA 2003*, vol. 1, pp. 1304 - 1311, 2003.

[59] D. Ribas, P. Ridao, J. Neira and J. D. Tardós, “SLAM using an Imaging Sonar for partially structured underwater environments”, in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5040 - 5045, Beijing, China, Oct. 2006.

[60] Adept Mobile Robots, “Pioneer 3 - DX Datasheet”, available: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx. accessed: 12 October 2014, 18:05.

[61] D. Ribas, R. Ridao, J. Tardós and J. Neira, “Underwater SLAM in a marina environment”, in *Proceedings of the 2007 IEEE/RSJ International Conference in Intelligent Robots and Systems*, pp. 1455 - 1460, San Diego, CA, USA, Nov. 2007.

[62] R. Muniz, L. juanco and A. Otero, “A robust software barcode reader using the Hough Transform”, *International Conference on Information Intelligence and Systems*, pp. 313 - 319, 1999.

[63] T. Pavlidis and S.L. Horowitz, “Segmentation of plane curves” *IEEE Transaction on Computers*, vol. 23, no. 8, pp. 860 - 870, 1974.

[64] G. Borges and M. Aldon, “A split and merge segmentation algorithm for line extraction in 2-D range images”, in *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 1, pp. 441 - 444, Barcelona, Spain, September 2000.

[65] G. Borges, M. Aldon, “Line extraction in 2-D range images for mobile robotics”, *Journal of Intelligent and Robotic Systems*, vol. 40, no. 3, pp. 267-297, 2004.

[66] T. Einsele, “Real-time self-localisation in unknown indoor environments using a panorama laser range finder”, in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, vol. 2, pp.697 - 702, Sept. 1997.

[67] L. Zhang and B. K. Ghosh, “Line segment based map building and localisation using 2D laser range finder”, in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2538 - 2543, San Francisco, CA, Apr. 2000.

[68] N. Sunderhauf, S. Lange, P. Protzel, "Using the unscented Kalman filter in mono-SLAM with inverse depth parameterisation for autonomous airship control", *Proceedings of the 2007 IEEE International Workshop on Safety, Security and Rescue Robots*, pp. 1 - 6, Rome, Italy, Sept. 2007.

[69] A.J. Davison, I.D.Reid, N.D.Molton and O. Stasse, "Mono-SLAM: real-time single camera SLAM", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052 - 1067, June 2007.

[70] J. Tardif, Y. Pavlidis, K. Danilidis, "Monocular visual odometry in urban environments using an omnidirectional camera", *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2531 - 2538, Nice, France, Sept. 2008.

[71] R. Hartley and A. Zisserman, *Multiple view geometry*, Second Edition, Cambridge, UK: Cambridge University Press, 2003.

[72] D.C. Brown, "Decentering distortion of lenses", *Photometric Engineering*, vol. 32, No. 3, pp. 444 - 462, 1966.

[73] J. Bouget (2013, Oct. 10). *Camera Calibration Toolbox for Matlab*[Online] Available: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[74] K. Konolige, P. Mihelich, "Technical description of Kinect calibration", available: www.wiki.ros.org/kinect_calibration/technical. Accessed: 10:06AM (CAT) on 25 January 2014.

[75] Micron, "1/2-Inch megapixel CMOS digital image sensor", MT9M001 datasheet, 2004.

[76] Hokuyo Automatic Company, "Scanning laser range finder URG-04LX Specifications", URG-04LX Datasheet, 2005.

[77] M. Montemerlo, S. Thrun, D. Koller and B. Wegbreit, "FastSLAM: a factored solution to the simultaneous localization and mapping problem", *AAAI.IAAI*, pp. 593 - 598, 2002.

[78] H. Durrant-Whyte and T. Bailey, "Simultaneous localisation and mapping: part 1," *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99 - 110, June 2006.

[79] J. Guivant and E. Nebot, "Optimisation of the simultaneous localisation and map-building algorithm for real-time implementation", *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242 - 257, June 2001.

[80] L. Paz, J. Tardos and J. Neira, "Divide and conquer: EKF-SLAM in O(n)", *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107 - 1120, October 2008.