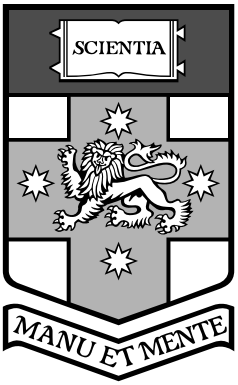
The University of New South Wales

School of Computer Science and Engineering



Simultaneous Localisation and Mapping on a Model Off-Road Vehicle

Sara Falamaki

Submitted in partial fulfilment of the requirements of the degree of Bachelor of Engineering (Computer Engineering)

June 2005

Supervisor: Dr. Waleed Kadous

Assessor: Prof. Claude Sammut

Abstract

Simultaneous Localisation and Mapping (SLAM) is often seen as the first step to achieving autonomous agents. This problem has been studied extensively in the last two decades but still remains a fairly elusive goal.

This thesis presents the design, and implementation of a system to perform SLAM on a Tamiya TXT-1 Monster Truck.

The results of an extensive literature survey is first presented, then the design and implementation of an algorithm to determine the trajectory of a moving robot, and simultaneously produce a volumetric map of its environment is detailed. A framework for testing this algorithm is designed and then used to obtain results to evaluate the solution.

Acknowledgements

I would like to thank all the people who have made completing this thesis possible. First, I'd like to thank my supervisor Waleed Kadous, for his guidance and support throughout the course of this thesis, and Raymond Sheh, for the invaluable advice, explanations and assistance he imparted about all things SLAM.

I would also like to thank my friends; Philip Derrin, for his encouragement and support, both technical and otherwise, Ada Lim for her guidance, advice and support throughout the course of my degree, and Sarah Webster for cheerfully proof-reading the manuscript for this thesis.

Last but not least, I would like to thank my parents for their patience and support throughout this endeavour. In particular, my lovely mum who made the long hours sitting behind a computer so much more bearable with her constant supply of fruit, nuts, chocolate and assorted goodies; all delivered to my desk.

Contents

List of Figures

1 Introduction

Rescue missions searching for survivors in the aftermath of natural disasters, wars, and urban rescue situations such as fires and storms are becoming increasingly common. It is often vastly preferable to send robots into hazardous situations rather than human rescuers.

Often the ability to utilise radio communication with such robots is limited; thus, autonomous and semi-autonomous robots are preferable to those requiring expert human control.

1.1 Motivation

This thesis aims to demonstrate the implementation of a system which allows Simultaneous Localisation and Mapping to be done on the TXT-1 Monster Truck. This solution aims to be useful in a Robocup Rescue arena. A photo of the truck is shown in 1.1.

The platform this solution is being deployed on has a number of interesting limitations. There is limited room on the truck, so large sensors, like a laser range finder would simply not fit on top of it. The usefulness of 2D SLAM solutions is limited; the truck will be climbing up platforms and dealing with floors which are not smooth or uniform.

This means that the system must be able to work with small, lightweight sensors which are rich enough in information so they can deal with 6-DoF motion. In the following sections, the background for this project, Robocup rescue, is discussed, as is why SLAM is important in both the context of this competition, as well as in broader search and rescue situations.

Figure 1.1: The TXT-1 Monster Truck

1.1.1 Robocup Rescue

Robocup Rescue is a competition designed to stimulate new research into the important problem of robot-aided disaster rescue at many levels, including [23, 25]:

- Multi-agent teamwork coordination
- Information Infrastructures
- Personal Digital Assistants
- Physical robotic agents for search and rescue
- Rescue strategies and decision support systems and
- Simulators and evaluation benchmarks.

The many research problems associated with this competition include dealing with noisy, dynamic environments, incomplete information, logistics, long-range planning, team collaboration between multiple agents, and designing agents that can survive extraneous conditions. By researching and developing agents that can perform autonomous search and rescue operations, survive in places that are difficult or dangerous for humans to venture into, and help rescuers by sensing important environmental measures, the job of recovering from natural and man-made disasters will be made more efficient, saving thousands of lives each year.

Robocup rescue is divided into two competitions and leagues, the Simulation League and the Robotics and Infrastructure league.

The competition focuses on the Urban Search and Rescue (USAR) tasks of identifying live victims, determining victim condition, providing accurate victim location and enabling victim recovery, without causing damage to the victim or the environment.

Accurate victim identification is encouraged, as is the generation of accurate maps of the environment. Autonomous behaviour is encouraged by penalising teams for each human operator used. The teams compete in missions lasting for twenty minutes each, in one of several arenas, rated based on their difficulty. The team with the highest cumulative score wins. The following performance metric is used for scoring:

$$\frac{\begin{matrix}map\\quality\end{matrix} + \begin{matrix}victim\\localisation\end{matrix} + \begin{matrix}victim\\tag\end{matrix} + \begin{matrix}victim\\situation\end{matrix} + \begin{matrix}victim\\safe\end{matrix} - \begin{matrix}arena\\bumping\end{matrix} - \begin{matrix}victim\\bumping\end{matrix}}{[1 + \begin{matrix}numberof\\operators\end{matrix}]^2} \tag{1.1}$$

1.1.2 Why Simultaneous Localisation and Mapping?

In order for an agent to navigate an environment and locate victims, it needs a map. If the environment is known *a* priori, the robot can be supplied with a map, which it will have to localise in. Occasionally, the environment is unknown and no map can be supplied; this is often the case with rescue and exploration missions, where a disaster could have altered the makeup of the environment dramatically.

In such missions an agent that can generate an accurate map is critical. Before such a map can be generated, the agent must be aware of where it is, in order to be able to plot what it sees; as such, the mapping and Localisation must be done simultaneously.

Once a robot can perform SLAM, it can plot various features of the environment onto the generated map, so that other robots, or human rescuers can locate and rescue survivors, or extinguish physical threats.

SLAM is an important part of the Robocup Rescue League, because it underpins the capability of any of the other tasks being performed. Once SLAM can be done on small, hardy and agile robots, they can precede other agents in a disaster site and help plan operations.

1.2 Goals and Contributions

The aim of this thesis is to design and implement SLAM on the TXT-1 Monster truck.

The goals of this work can be stated as:

- Build a map that allows for motion with 6-DoF.
- Accurately localise on the generated map, at each stage of the process.
- Use a sensor that is small in both weight and footprint.

Upon the completion of the project:

- A system for performing SLAM on the truck will be presented
- A testing framework for evaluating the performance of this system will be written
- The system will be evaluated using this framework

1.3 Outline of Thesis

Chapter 2 provides a literature review on current SLAM implementations. The prevailing methods of performing SLAM are discussed, and these solutions are put into context.

Chapter 3 discusses other, related, uses of the SLAM technique. This chapter focuses on 3D SLAM, SLAM using vision, and multi-sensory SLAM. The different implementations are evaluated in terms of this thesis' design goals.

Chapter 4 presents the overall design of the algorithm used in this thesis. A step by step analysis of decisions is presented, and a high-level view of the design is given.

Chapter 5 explains the algorithm used to implement SLAM on the monster truck. The different stages of the algorithm are presented, and some alternative solutions are discussed. Implementation decisions are justified.

Chapter 6 evaluates the performance of the algorithm, and presents results for different scenarios, both real, and simulated using Povray. Test results for individual parts of the algorithm, as well as the algorithm on the whole are presented in this chapter.

Chapter 7 inspects the results presented in the previous chapter and gives commentary and analysis on the outcomes obtained. Strengths and weaknesses of the algorithm are discussed in this chapter.

Chapter 9 talks about improvements that can be made to the solution, interesting ideas that can be tried in order to improve it, and future work that would allow the solution to be extended to perform other tasks, such as victim localisation and autonomous exploration.

Chapter 10 wraps up the thesis, and presents conclusions from previous chapters.

The appendix includes a user manual for the installation and deployment of the system.

2 Background

2.1 Simultaneous Localisation and Mapping (SLAM)

In order for a robot to achieve true autonomy, it must be able to localise itself in its environment. Simultaneous Localisation and Mapping (SLAM) addresses this problem.

Current implementations of SLAM employ statistical methods such as Extended Kalman Filters (EKF), Particle Filters, and other estimators to localise themselves with respect to certain landmarks. Landmarks are detected using either range finders (laser, sonar, infrared) or cameras (wide angle, CCD, stereo) [32].

There is a wide body of research addressing the SLAM problem. This chapter describes the SLAM problem, and different approaches to it.

2.1.1 The SLAM Problem

The main challenges in building a SLAM algorithm can be summarised as [32]:

- Errors arising from measurement noise. Noise would be easy to compensate for if the data measurements obtained were independent. Unfortunately all measurements will be statistically dependent, meaning that we cannot simply take more measurements to reduce error, and errors will accumulate.

- The high dimensionality of the data obtained. A detailed 2D floor plan requires the storage of thousands of numbers, and a detailed 3D image of a building requires the storage of millions of data points.

- The correspondence, or *data association* problem. This is the problem of determining whether two sensor readings taken at two different times correspond to the same object in the world.

- A changing environment. Most mapping algorithms assume a static world where all changes to the world are interpreted as noise. If the mapping takes place over a longer period of time, changes in the world become more significant (even closing an open door will alter the world in such a way to confuse the robot).

- The Robotic Exploration problem. In order to be able to map it, a robot must

explore its world autonomously. It must be able to know where to explore, and at what point to retreat, being able to recover from unforeseen circumstances. This is a hard problem.

As the Robocup Rescue (see section 1.1.1 on page 10 competition is in a static environment, this thesis will not be dealing with the challenges of a dynamic environment. The main focus will be on localising with respect to points observed, and plotting these points onto a 3D grid.

SLAM is a problem, not a solution, and as such, there are many different approaches to solving it presented in a multitude of papers, in the last 20 years. These approaches can be roughly classified into three major groups:

- Kalman Filter and Extended Kalman Filter based techniques using both bearing and range measures.
- Bearing-only SLAM, using EKF together with other techniques, such as particle filters.
- Scan-Matching techniques, such as those using the Iterative Closest Point (ICP) algorithm.

In the following section, the above techniques are introduced, and some examples of their application is given. Most implementations use a combination of the above approaches, and these implementations will be discussed in greater depth in the next chapter, Related Work.

2.1.2 Approaches to SLAM

Kalman Filters and Extended Kalman Filters

In order to perform SLAM, information about nearby landmarks is obtained, and the robot localises itself with respect to these. This information may then be augmented with odometry data, and data from other sensors to create a map.

These landmarks can be distinctive contours, visual points of interest, or explicitly laid out beacons.

Most algorithms for performing SLAM are probabilistic. This is because the problem of robotic mapping is characterised by uncertainty and sensor noise [32]. Probabilistic algorithms approach this problem by explicitly modelling both the sensors and the different sources of noise. Bayes rule is the principle underlying these systems, giving the probability of observing a particular measurement d, under a hypothesis x,

$$p(x|d) = \eta p(d|x) \cdot p(x)$$

A Bayes filter extends Bayes rule to temporal estimation problems, by recursively estimating posterior probability distributions over quantities that are not directly observable. This recursiveness means the time per update is constant, enabling a Bayes filter to integrate information indefinitely. Kalman filters, hidden Markov models, dynamic Bayes networks, and partially observable Markov decision processes are closely related to Bayes filters. A Bayes filter allows simultaneous estimation of the map model, and the robots pose in the environment.

Kalman filters are Bayes filters that represent posteriors with Gaussian distributions. Gaussian distributions can be represented compactly with a small number of parameters. In robotic mapping, these are the robots pose and the map.

Three basic assumptions underly the Kalman filter model:

- The next state function must be linear with added Gaussian noise.
- The same characteristics must apply to the perceptual model.
- The initial uncertainty must be Gaussian.

Theoretically, if the above conditions hold, a Kalman filter based approach will always converges. Additionally, as it stores the uncertainty of the entire map, it aids navigation and exploration.

An Extended Kalman Filter (EKF) is a modification of the Kalman filter which can handle nonlinear dynamics and nonlinear measurement equations. The EKF combines the model of the system dynamics with noisy sensor data.

EKF SLAM requires each landmark to be observed infinitely many times, and the resulting map only shows the landmarks observed. Any additional information (such as the placement of walls) must be obtained by other means, usually with the help of a laser range finder, or sonar sensors.

This approach is usually applied to robots moving on flat terrain, in other words, 2D SLAM. Odometry information is used to estimate the robots position, and this allows for only planar motion, with 3 degrees of freedom (translation and rotation in a 2D plane). There has been approaches using Kalman filters (together with stereo vision) to do SLAM in 3D. The implementation in [1] extends the Kalman filter by embedding the standard deviation of the terrains curvature, which it obtains using additional sensors.

There is a vast body of literature dealing with performing SLAM using Laser range finders (such as SICK), discussing optimisations that deal with the problems mentioned above and in the previous section.

Montemerlo *et* al have developed a technique called FastSLAM [19] which addresses the issue of complexity, as well as the data association problem. This approach uses a particle filter (PF) to factor the full SLAM posterior into a product of the robot path posterior. The time taken to factor the observations into FastSLAM scales logarithmically to the number

of landmarks, furthermore, FastSLAM samples potential data associations as well, enabling it to be used in environments with highly ambiguous landmark identities.

Building on this technique, Nieto *e*t al use multiple hypothesis tracking (MHT) [22] along with a Nearest Neighbour Filter (NNF) to further handle uncertainty in the data association problem.

The data association problem is discussed in a paper by Hähnel *e*t al [12]. This paper seeks minimise data association uncertainty by establishing data associations lazily instead of using a more naive maximum likelihood (ML) heuristic. Once a wrong choice has been made using incremental ML, there is no chance of recovery. By using a tree search to consider alternative data associations, not just for the current increment, but previous ones too, this brittleness is circumvented. Instead of maintaining just a single path in the data association tree, this approach maintains an entire frontier. This approach seems to be successful in resolving ambiguities in the map.

Most solutions which use the canonical Kalman filter only map the world in two dimensions. These solutions can map corridors and smooth passageways, but are usually unsuited to undulated terrain.

Bearing-only SLAM

Traditionally SLAM has been performed using sensors providing both range and bearing. These sensors were traditionally sonar, but more recently laser range finders are increasingly employed.

It's desirable to be able to perform SLAM using just bearings, as this allows for the use of much cheaper, and more compact vision sensors. Using a camera, one can guess bearings, and judge relative depths of objects, however, in order to get a realistic scale, the algorithm must be initialised with known parameters. EKF SLAM cannot deal with feature initialisation unaided.

The critical problem with initialisation is, as a single measurement does not constrain a feature location [2], at least two measurements are required. If these two measurements are taken sequentially, there is insufficient baseline to make a good location estimate, unless more constraints are placed on the measurements (for example, if a stereo camera is used, the distance and angle between these measurements is fixed).

A possible solution is using non-linear batch initialisation methods, observing a feature multiple times, until its position exceeds a certainty threshold. The problem here becomes evaluating this certainty: when do we know we've seen it enough times? The object may have been observed multiple times from the same position, giving away little in the way of certainty. A technique which uses non-linear batch adjustment by tying it into a conventional EKF framework is *bundle adjustment*. Bundle adjustment places a condition to measure when to do feature initialisation.

Bailey [2] presents a method for initialising features by retaining past vehicle pose estimates in the SLAM state vector until the estimates for them become well-conditioned. He defers feature initialisation until the probability density function (PDF) of its location closely resembles the Gaussian approximation obtained from a Jacobian-based linearised transform. These two distributions are compared using the *Kullback-Leibler distance* measure, or relative entropy.

EKF SLAM assumes that the estimates have small errors, and are near-Gaussian. This method meets this requirement, as new features which do not have Gaussian PDFs will not be initialised. The sequence the feature estimates are incorporated does not matter, and this method retains consistency despite integrating features out of order. What this paper does not address however is the data association problem, which it defers to future work.

Another approach to this problem is presented by Davison [9, 8] with the use of a single camera. A state vector is used to represent each feature detected and a covariance matrix is used to represent uncertainties for all the quantities in the state vector. Feature estimates in the state vector can be freely added or removed. The state vector, and the covariance matrix are updated in two steps:

1. During motion, a prediction step uses a *motion model* to calculate how the camera moves and how its position uncertainty increases.

2. When feature measurements are obtained, a *measurement model* describes how map and robot uncertainty can be reduced.

The key idea here is that clusters of close features are highly correlated to one another, their relative positions are well known, but their position in the world reference frame is uncertain. By deferring their localisation until more information is gathered about their actual position in the world, they can eventually be plotted with relative accuracy. Like Bailey's approach, Davison *et* al estimate the exact position of a point by comparing Gaussian uncertainties, however, instead of using point features, they use elliptical regions several pixels wide.

Scan-Matching Based Techniques

Although most SLAM implementations employ a probabilistic approach to localise individual features on some sort of map, there are a class of SLAM algorithms which merge different world views by estimating the rotation and translation of the robot. This is done by iteratively moving points, such that the distance between all points in the different views is minimised, thus geometrically aligning the views.

The Iterative Closest Point algorithm (ICP), first proposed by Besl & Mc Kay [3] is the most common way of finding this transformation.

The ICP algorithm can be divided into 6 basic steps [28]

1. **Selection** of some set of points in one or both meshes.

2. **Matching** these points to samples in the other mesh.

3. **Weighting** the corresponding pairs appropriately.

4. **Rejecting** certain pairs based on looking at each pair individually or considering the entire set of pairs.

5. Assigning an **error metric** based on the point pairs.

6. **Minimising** the error metric.

Although EKF is a tried and tested technique for localising in two dimensions, it becomes intractable with a very large set of features, and has to be heavily modified so it can deal with the 6 degrees-of-freedom when mapping in 3D.

In the previous section the use of particle filters and bundle adjustment was discussed, these techniques localise features (points, or small patches) in 3D space probabilistically. Increasingly, solutions to SLAM are being published where instead of using the probabilistic EKF model, which relies on error being both stochastic and Gaussian, attempt to match scans of the world using ICP.

Each of these solutions requires the following steps:

1. Point acquisition: Points are usually acquired using a stereo camera [17] or using 3D laser scans [6]

2. Feature detection: Often done using corner or edge detection algorithms

3. Feature pairing: Using Scale Invariant Feature Transform (SIFT) descriptors [29], KD-Trees [20] or cross-correlation maximisation [17]

4. Finding Homogeneous Transform: Found using some variant of the ICP algorithms registration phase

5. Integrating each frame into the 3D world map.

An ICP-based algorithm is implemented to do SLAM in this thesis.

2.2 Hardware

The visual sensor this thesis relies on is the stereo megapixel camera(MDCS) from Videre. The STH-MDCS is a compact, low-power, synchronised digital stereo head with an IEEE 1394 digital interface. It consists of two 1.3 megapixel, progressive scan CMOS imagers mounted in a rigid body, and a 1394 peripheral interface module, joined in an integral unit. [34]

It can mounted on a pan-tilt mechanism to facilitate 6 degrees of motion.

The implementation presented does not require any additional specialist hardware, assuming a fairly fast processor, and plenty of memory are available on-board for real-time processing, or off-line for processing at a later stage.

2.3 Software and Libraries

2.3.1 Small Vision System

In order to utilise the capabilities of the stereo camera, the images it produces need to be rectified, and their depth determined. The Small Vision System [15] (SVS) software does real-time stereo analysis on consumer hardware.

In order to produce a 3D point cloud, SVS needs to first find matching points in the two images, and then apply image geometry to calculate disparity. The two main ways of doing this are feature extraction, and small area correlation. With feature extraction, a number of distinctive features are pinpointed, and then matched between the images; these are often corners or edges, as such features remain *somewhat* invariant at different angles. The major fault with using feature extraction is that finding features can be time consuming to compute, and algorithms using features detection tend to produce sparse results. SVS instead, uses area correlation. In this method, a small area is selected, and then compared to other small areas in a search window in the alternate image. The extreme value of the correlation at each pixel is selected, and used to determine the disparity of the pixel. Post filters then clean up noise in the resulting disparity image.

Because of the way SVS calculates disparities, smooth, uniformly coloured, un-textured surfaces are impossible to correlate, and SVS can not produce meaningful disparities for points on such surfaces. As a result, if the camera sees a smooth surface, it will be able to calculate the disparities and 3D point values at the edges, but the object will appear hollow. Figure2.1 illustrates this.

Another consequence of this method is that shadows on smooth surfaces get picked up, whereas the surface itself does not. It's important to realise this when designing a system that relies on SVS for points, as it can not be used to determine the presence of surfaces. A robot doing SLAM with only these points has a good chance of running into walls.

A stereo camera set using SVS needs to be calibrated. The calibration process calculates intrinsic camera parameters, so it can compensate for things like lens distortion when doing rectification. Parameters to the algorithm also need to be set each time a camera is used, catering for different environmental situations, such as the presence of multiple light sources, and the distance from the objects seen. Correctly calibrating SVS for a camera rig is quite a tedious process.

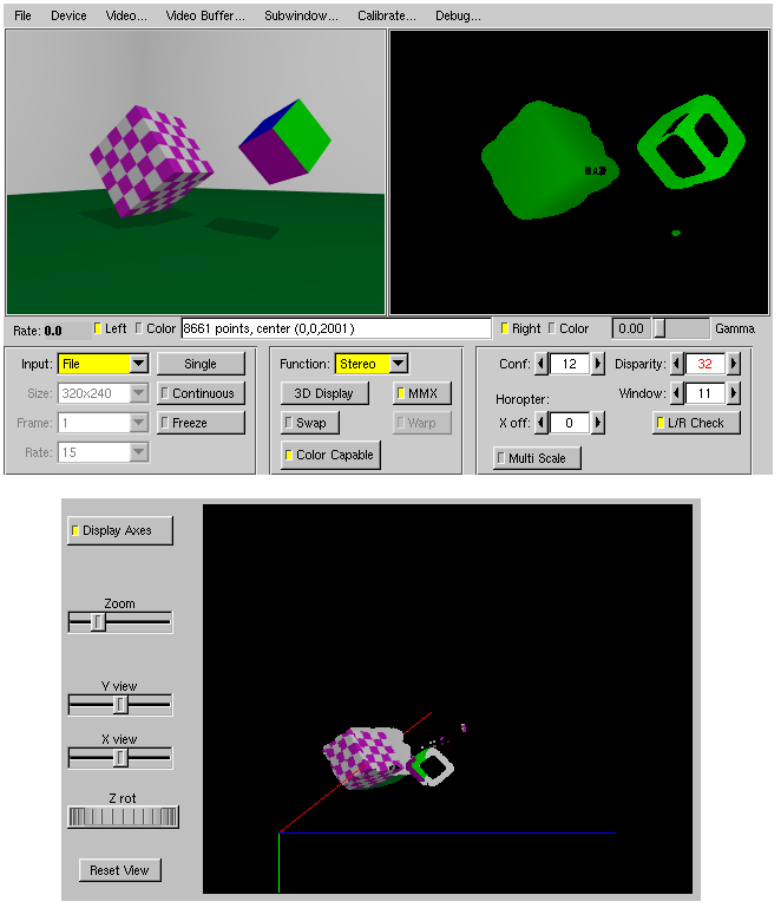


Figure 2.1: SVS window and 3D display showing a cube. The smooth surfaces of the cube are invisible.

2.3.2 Newmat

Many of the transforms in image algorithms require the use of matrices. The somewhat deceptively named Newmat [7] library provides the matrix transforms needed, in particular, it can be used to find Single Value Decompositions (SVDs) in the ICP algorithm. Being fairly old, it is tried and tested, however, it doesn't utilise many of the language features in C++.

2.3.3 VTK

To test results, and present the algorithm, the points need to be visualised somehow. For Visualisation Toolkit (VTK) was used to facilitate this. VTK has bindings to many languages, including C++ and Python. For this project, the Python bindings were used because this avoids having to use Cmake (a make-like tool VTK uses to compile C++ code), and being able to change the visualiser without needing to recompile it.

2.3.4 Vigra

To do the feature extraction stage of the implementation, an imaging library that does Harris corner detection and Canny edge detection was sought. Vigra [5] is a generic computer vision tool library which can be used for both tasks.

3 Related Work

Various research groups have been working on the SLAM problem since the 1980s, and there is a constant stream of papers being published on the topic.

This thesis focuses on doing SLAM in 3D, and as such, the related work presented here focuses mainly on solutions to the 3D SLAM problem, as well as those using vision based techniques, especially the solutions published recently, in 2004 and 2005.

As in the background chapter, the works presented here are divided into three categories: those using EKF techniques and variants thereof, those using bearing only measurements, and those utilising the scan-matching techniques, such as the ICP algorithm. It should be emphasised that this is not an exhaustive literature survey — the wealth of information out there makes one impossible — nor are the categories presented here the only way one can divide the various solutions to this problem. This chapter will however highlight some of the more relevant theory, and current practice, and be used to justify design and implementation choices made in subsequent chapters.

3.1 Kalman Filters and Extended Kalman Filters

SLAM with Omni-directional Stereo Vision Sensor

The paper by Kim *et* al [13] combines Structure From Motion and stereo vision to implement an EKF based SLAM algorithm. The stereo vision is obtained using an omni-directional camera.

This paper tries to address the correspondence problem in all vision based SLAM algorithms by replacing feature detection with SFM, which doesn't have a correspondence problem. SFM however, cannot resolve scale, and the stereo vision complements it by providing depth information. This is similar to the proposal made by Dillart et-al [10] which talks about structure from motion without correspondence.

This implementation seems to have been tested in very limited environments, particularly flat ones.

SLAM- Loop Closing with Visually Salient Features

An extremely challenging problem in performing SLAM is finding loop closures, finding if the robot is in an area that it has mapped previously.

This problem ties in with the problem of using odometry to calculate pose, as soon as the robots position becomes erroneous, it is extremely hard to recover, and detect loop closure. This is precisely the time that loop closure would be useful.

Most SLAM solutions take advantage of the accuracy of laser sensors. Laser based sensors however, have a significant disadvantage. They are very information poor. In contrast, vision based solutions, using cameras are very information rich, however, calculating exact range and bearing measures from these sensors is still an exceedingly difficult task.

The implementation proposed in a paper by Newman and Ho [21] finds visually salient features with wide baseline stability (they can be observed from different viewpoints while still being recognisable) and encodes them using SIFT descriptors. Before a new feature is added to the map, the algorithm queries this database of encoded visual features, and seeks a match. If a match is found the loop is closed. Any SLAM algorithm can be used for the localisation part of the problem, this particular solution uses a simple non-linear Kalman filter.

This paper illustrates the advantage of combining traditional range-bearing laser sensors with vision in order to obtain more information.

An Autonomous Robotic System for Mapping Abandoned Mines

In this paper by Ferguson *e*t al [11] SLAM with lazy data association is used in combination with A* search to facilitate corridor following and mapping in abandoned mines not reachable by people. Two laser range finders are used, one for 2D and the other for 3D scanning. The 3D maps are projected on a 2.5D terrain map.

By combining these techniques robust system has been developed to search abandoned mines. The authors have deployed their system in extreme conditions, some inaccessible by people; and have obtained surprisingly good results.

The major reason that such an approach is impractical on the TXT-1 is that even a single Laser range finder is far too big to fit on its chassis, two is just impossible.

3.2 Bearing-only SLAM

Up until recently, most of the SLAM algorithms implemented relied on the availability of either laser, stereo vision , or sonar range finders [11, 19, 12, 16, 33, 32]. However, such

devices are large and expensive and unsuitable for many applications. Being able to employ SLAM techniques using only cameras is appealing, as cameras are cheap, compact and flexible.

Active Vision

In their paper Davison *e*t al [9] describe a technique for implementing SLAM using only a 30Hz camera waved around in the hand. The algorithm picks out points of high interest and uses these features to localise. These features are often corners, or distinctive, well localised small objects. Initially the depth of the feature is unknown.

A semi-infinite 3D ray into the map with an endpoint at the camera's optical centre is initialised, and it's known that the feature lies somewhere along this line. By repeatedly testing, many hypotheses are gathered about the point's exact depth, and probabilities evolve. The point is eventually plotted on a 3D map representing the search space.

The four corners of an A4 piece of paper are enough to bootstrap the system (they are used to provide information on the scale of the map and the camera motion) and map-management heuristics are used to decide when to initialise new features. The aim is to keep a pre-defined number of features visible at all times. Features not reliably matched repeatedly are deleted.

Although fairly successful, the Davidson(2003) system was limited by the camera they employed. Because of the narrow field of view, the features that could be seen simultaneously had to be close together, leading to uncertainty in the camera position estimates.

A subsequent paper [8] describes using a wide-angle camera instead, providing a far greater field of view. By using a wide-angled lens, they are able to achieve better camera motion estimation, have an increased range of movement and have an increased range of tractable movement accelerations. At the moment, the technique is limited in that the camera must stay roughly horizontal, as feature matching takes place through 2D template matching.

The only problem with using active vision based techniques in this context is that they can't be used for mapping, on their own. This is because the only things that appear on the map are landmarks, which can be used for localisation, but provide very little in the way of mapping an unknown environment.

3.3 Scan-Matching Based Techniques

3D Simultaneous Localisation and Modelling from Stereo Vision

Garcia and Solanas [17] present a 3D SLAM scheme that avoids statistically modelling either the sensors or the map using a single stereo vision system. This system determines the 6-

DoF self-motion without using odometry, and does not require any initial calibration.

The algorithm proposed can be outlined as follows:

1. Acquire 3D point Cloud

2. Find corresponding pairs of points, in adjacent images

3. Compute the rotation matrix and translation vector between the two images using the registration phase of the ICP algorithm

4. Perform a 3D transform based on the pinhole camera model in order to integrate each new acquired frame

Much of this thesis uses ideas presented in this paper, and as such, it will be discussed in far greater depth in subsequent chapters.

Using Naturally Salient Regions for SLAM with 3D Laser Data

Cole *et* al [6] present an approach very similar to that found in [17], this time using a laser range finder mounted on a motor driven reciprocating cradle. This setup allows the collection of 3D laser range scans.

In this technique which is optimised for outdoor environments — where odometry doesn't yield useful information, and there is a significant absence of crisp walls, corners and geometric surfaces — allows their robot "Marge" to move in terrain with motion with 6 DoF. Like [17], this technique isn't strictly feature based, it instead registers segments of 3D data producing an egomotion transform. Where it differs from [17] is that instead of using geometric feature detection on a 2D image obtained by a camera, it locates salient patches in the laser scan.

Laser scans are segmented based on regions of local interest. The saliency of a region is determined by looking at the changes in the distribution of normal directions in that area. Multiple scans of the same region are required to compensate for the unevenness of laser range data. A camera is used to "colourise" the points, aiding the pairing process, as well as providing more informative maps. Correspondences are found using an approximate KD-Tree, and the transform matrix is calculated using an algorithm similar to ICP.

This is a very interesting paper, as it uses a traditional sensor, a laser range finder, but instead of using EKF, it uses a scan matching based technique to achieve very impressive matches. The only problem with this implementation in the context of this thesis is that there is no room for a laser range finder on the truck.

4 Design

In the background and related work section, the various solutions to the SLAM problem were divided into three categories. Those who use a combination of odometry data and range-bearing laser sensors to build a probabilistic, 2D map of the world; those which use bearing-only sensors such as cameras, in combination with the probabilistic Kalman Filter approach to map features onto a 3D world map; and finally, those which use scan matching or ICP to find transform which results in the greatest overlap between successive frames, and use this transform to unify all acquired points into the one map.

The major drawbacks of the first set of solutions are that they map the world in only two dimensions and rely on odometry. This makes them unsuitable for deploying in a non-planar rescue environment, where the robot's wheels will slip, and the robot will climb up obstacles, rendering odometry information useless. A 2D map is also unsuitable for a robot that needs to explore autonomously, how would it know whether it can climb up an obstacle, or if has to go around it?

The second set of solutions doesn't rely on odometry, and produces a 3D map. These solutions employ small, lightweight, vision based sensors, which make them suitable for use on the truck. The major problem with these solutions is that they only plot *features* onto the map. In other words, they are good for localising, but will not produce a world map suitable for navigation. Such solutions could potentially be complemented with range sensors to produce some sort of grid, but on their own will not produce a 3D map.

This leaves the final set of solutions. Instead of mapping features and the uncertainties associated with them, these solutions take a 3D scan of the world, and then use curve matching, point matching or Structure From Motion(SFM) techniques to *unify* different views of the world into on coherent world map. These solutions need both range and bearing data, so if vision is to be used, a stereo camera is required.

This thesis uses ideas from this final set of solutions. The goals of this thesis are to:

- Build a map that allows for motion with 6-DoF.
- Accurately localise on the generated map, at each stage of the process.
- Use a sensor that is small in both weight and footprint.

A stereo vision with point-matching solution will meet the first criterion, as it will allow the construction of a 3D map. Localisation can be performed by finding an homogeneous transform for moving points found in one frame's field of view to another's; this localisation

matrix represents the current position of the agent with respect to the origin; thus meeting the second criterion. A stereo camera rig will easily fit on the truck, thus satisfying all three criteria.

This thesis is modelled on the paper titled "3D Simultaneous Localisation and Modelling from Stereo Vision" [17]. As mentioned in 3, this solution maps the world using a six stage algorithm. Basically, at each iteration an image, and a corresponding 3D point cloud need to be obtained, the salient features from the current frame need to be matched with those of the preceding one and a transform function that moves each point in the current from to the position of its matching point in the previous frame needs to be computed.

Once a transform is computed, it is multiplied by the transform in the preceding image, and then applied to all the points in the current image. This recursive definition insures that every point in the current frame is placed in the coordinate system of the initial frame in the series. Finally, in the integration step of the algorithm the points in the current frame undergo a 3D transformation that brings them closer to the points in a few of the previous frames. This transform helps unify the points into a coherent image, and helps compensate for small disparities between the transform obtained from ICP and the true transform.

The following design is based on the paper by Garcia *et al*, but modifications have been made where their design has proved insufficient. This chapter will discuss some of the design aspects associated with the algorithm proposed by Garcia and Solanas, and propose some possible implementations. The following chapter will realise this design, and talk about the practical issues associated with it. A thorough evaluation of this algorithm, and modifications to it will be presented in the Results and Evaluation chapters.

4.1 Defining a World Model

The aim of this thesis is to generate a map of the environment surrounding the agent. To do this, the agent's world needs to be modelled. The world can be divided into a simple object hierarchy:

- World: Consists of a map, the camera frames used to obtain it and the agents current pose.

- Frame: Consists of the 2D bitmap images obtained from the camera, the corresponding point cloud, and information about the characteristics of the images, for example, their width and height.

- Point: Consists of a row and column coordinate representing its position on the 2D bitmap image, a field representing its colour and optionally, X, Y and Z coordinates representing its position as projected in 3D space.

Once the data structure is in place, the algorithm over it can be defined as in the following sections.

4.2 Obtaining point clouds

In order to be able to construct a map, information about the depth and the bearing of the points observed is needed. Since space on the TXT-1 truck is quite limited, using a laser range finder to find depth was not feasible. Since visual sensors are compact and information rich, a camera was seen as a viable option.

Once decided on using a camera, the options were stereo vision and active vision. Active vision works with a single lens by delaying the time a point is entered into the map until its location is known above a certainty threshold. While the active vision technique is useful for plotting landmarks in 3D space, the number of "features" it can track at once is quite limited, making it unsuitable for constructing an occupancy grid without assistance from other sensors.

Deducing distance information from a pair of cameras whose parameters are known is a simpler task, which can be done independently of the rest of the algorithm. This means that far larger sets of points can be dealt with because once a transform is calculated from a few points in the set, it can be applied to all the 3D points calculated.

The main drawback of using stereo vision for this task is that deducing the distance from smooth, un-textured objects is impossible. Further discussion about this problem can be found in the results section.

In this implementation the a stereo camera from Videre Design is used, and the stereo processing is done using the SVS software package from SRI International.

4.3 Filtering and selecting points

Once a set of 3D points have been obtained, they will need to undergo some pre-processing before they are useful for mapping. This is because the stereo processing software is not, and can not be perfect. It is quite prone to making mistakes where little texture is available, at very close and very far distances, and where lighting and shadows make for erroneous disparity calculations. At the very least, points that are at infinity, or are too close need to be filtered out and discarded.

SVS samples points by using correlation matching in texture patches. This means that all the points obtained are in areas with lots of texture. It also means that SVS cannot get depth information for background areas around foreground areas. This issue is illustrated in Figure 4.1 on the facing page.

In order to obtain accurate transforms, only stable points should be used, this often means using only edge points, or corner points. Once the transform is calculated, all the 3D points obtained can be transformed using it, but only using a subset of points to determine the transform will make finding it both more accurate, and faster.

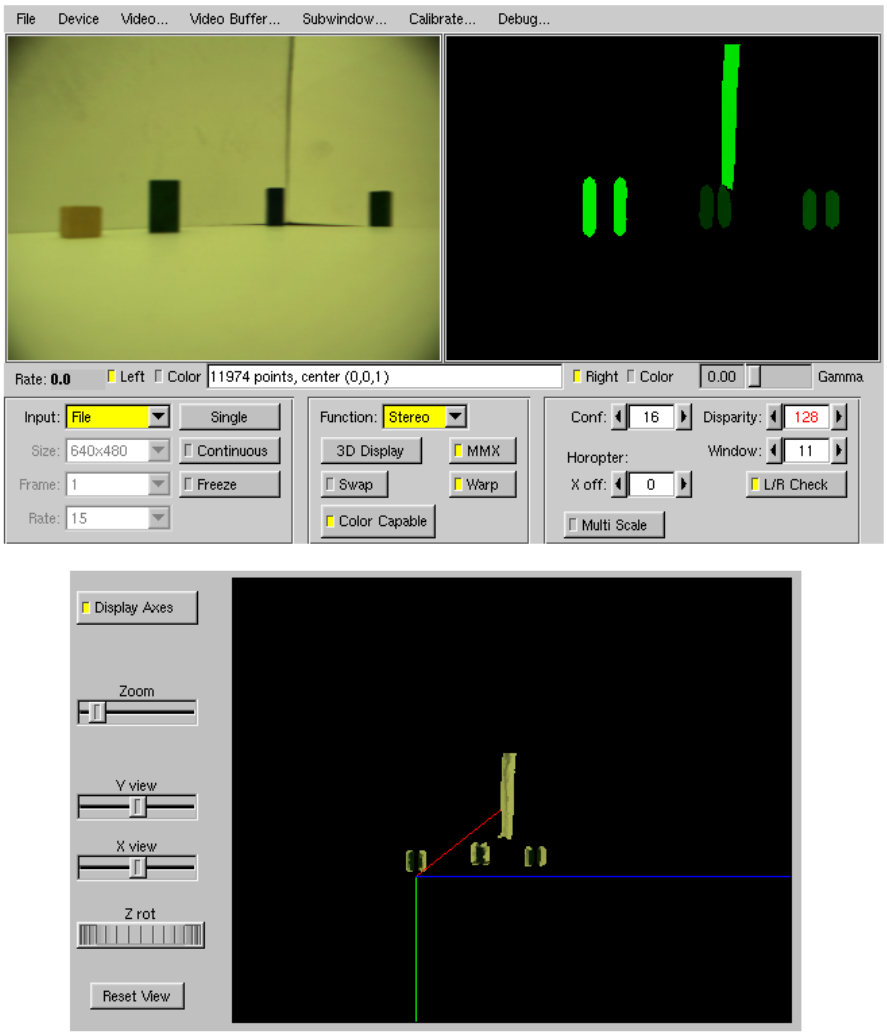


Figure 4.1: SVS picking up the background with the block

There are many ways of choosing salient points. They can be classified as follows:

- Geometric Measures. Lighting, motion and different points of view can dramatically alter the appearance of an object, however, the basic shape of an object tends to remain the same. So long as they are visible, the relationship between the edges or corners of an object are far more stable than other characteristics such as the appearance of colours and textures on its body. Geometric measures take advantage of this observation, and choose either edges or corners as salient features.

- Rarity Based Measures. Rarity based measures for finding salient points are based on the assumption that saliency implies rarity. These measures look for unique features, maximising the probability of the object given a measurement. The higher this probability, the more distinguishing the feature.

- Complexity Measures. Relying on the assumption that in real images complexity

is rare, such measures measure the local signal complexity, or unpredictability. Robust, modern filters such as SIFT rely on such measures. Obvious cases where the assumption underlying this measure will not hold are fractals and textures.

In this thesis geometric measures are used to determine if a point is a salient feature.

Using only salient features is particularly important in this thesis for a number of reasons. The algorithm works by correlating points from one frame's point cloud with another's, then computing the spacial transform required to move one of the pair, into the position of another. This transform is computed by moving all points towards each other, to minimise their distance.

Stereo processing cannot guarantee the correct range and bearing for every point seen through the lenses. In fact, only a small number of pixels have any sort of 3D information associated with them, and only objects in the tend to have this 3D information computed correctly. Foreground objects are usually "frilled" with parts of the background, and these frills have incorrect range and bearing measurements. If the transform function was fed such points, it would compute an incorrect transform, as it would try to minimise the distance between non-existent, fairly randomly appearing, points. By using only points on edges or corners, these points are avoided.

Another reason to use only salient points is because the points chosen need to be matched over several camera frames. In order to be able to do this, the points chosen must be minimally affected by lighting, and by camera angle.

4.4 Matching points

Once the points of interest are chosen, correspondences need to be found between frames. These correspondences can be either computed based on the 2D images, or by matching the points in 3D space. In this implementation this step is done using the 2D images, but checks the 3D distance as a sanity measure.

4.5 Obtaining a transformation matrix

In order to localise, the agent needs to compute how far, and in which direction, it has travelled. This information is also needed to plot the points it sees onto a map.

To find this information, a transform is computed using the pairings obtained in the previous section. The Iterative Closest Point (ICP) algorithm is used to find this transform by moving the points close together, until the distance between them is minimised.

4.6 Integrating Points

To compensate for localisation errors inherent to the stereo processing system the integration process adjusts the spacial position of the acquired 3D points. This stage takes advantage of the redundancy derived from observing each point several times on the agent's path.

Given any two frames, `Frame_a` and `Frame_b`, this stage moves every point in `Frame_a` into the coordinate system of `Frame_b`. These points are then projected onto the image associated with `Frame_b` and for each point, the image in `Frame_b` is searched until the pixel with the highest normalised cross-correlation with search point is found. The position of this pixel is then adjusted into the position of the point from `Frame_a` in `Frame_b`'s coordinate system.

This process is done in both directions, for the last three frames observed.

4.7 Modelling and Visualising the Solution

Once the transform matrix has been calculated for a frame, and its points have been adjusted by looking at its neighbours, its adjusted point-cloud, its 2D image, and its localisation matrix are stored in the world map.

In order to view these points together, each point has to be multiplied by its frame's localisation matrix first. Points are then written to a file, which is parsed by a Python VTK script so that they can be viewed.

4.8 Testing Framework

To be able to test and debug the algorithm, a way to control the many different parameters that can occlude the true performance of the algorithm needs to be found.

If the camera is not calibrating properly, and SVS produces meaningless disparity values, and this data is used to test the algorithm, the results obtained are equally meaningless – garbage in, garbage out.

Equally, if the points are matched incorrectly in the matching phase, and this data is fed into the next stage, and an incorrect transform is calculated, it's very difficult to pinpoint where the problem occurred.

The goals of the testing framework can thus be summarised as below:

1. Ensure data going into the algorithm at each stage is meaningful

2. Be able to evaluate results by comparing them to some "ground-truth"

3. Be able to test each stage of the algorithm independently

To facilitate the achievement of goal 1, two sets of test data are generated. One set is generated by carefully calibrating the camera, and recording a sequence of frames for testing, with some attempt to control environmental factors such as lighting and noise. The points are then checked to see if meaningful disparities have been obtained by SVS. The second set is generated artificially, by ray-tracing and animating a scene using Povray. The images obtained by this process are then fed into SVS to generate disparity information and point clouds. This ensures noiseless, well lit data, which can be used to test the basic functionality of each stage of the algorithm. If it works for this data, it can be modified to work for more noisy data, if it doesn't, it's seriously broken, and needs to be fixed and debugged.

Measurements of the distances between the different objects on the real world scene are taken so they can serve as a ground truth to compare the results with. The second way of obtaining test data is particularly good at meeting goal 2, as the Povray files provide the exact locations that the objects should be located at, and can be visualised by moving the camera in different positions, allowing a good comparison.

The last goal of the test frame-work is independent testing. To do this, a set of unit tests are developed to test each stage of the algorithm independently. This also helps assure goal 1 is met, as it means that data distortion in the middle of the pipeline can be checked for.

5 Implementation

This chapter provides and overview of the various stages of the algorithm implemented and discusses the data structures used, justifying design choices.

5.1 Modelling the World

The aim of this project is to represent the world in 3D. In order to do so, information about the points that have been observed have to be retained, and stored in some sort of world map.

The `World` class consists of a `World_Map`, and operations to facilitate the addition of each new `Frame`.

Each time the camera acquires a pair of images, the svs library process them into a 3D cloud map, and a disparity map. The colour of each pixel, its rectangular coordinates and its 3D coordinates, if known, are stored in a `Point`. A `Frame` is a collection of points, storing information about each pixel, as well as the size and dimensions of the images. A `Frame` can optionally detect features or edges in the stored image.

The world model can be graphically represented as in figure 5.1:

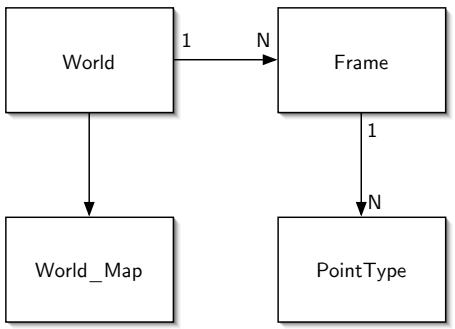


Figure 5.1: Main classes in SLAM implementation

5.2 Localisation and Mapping

The main algorithm can be divided into six discrete, independent stages:

1. Obtaining point clouds
2. Filtering and selecting points
3. Matching points
4. Obtaining a transformation matrix
5. Integrating points
6. Modelling the solution

Each of the above stages can be implemented in a number of different ways. In this chapter, each stage will discussed, and the reasons for design decisions will be justified.

5.3 Obtaining point clouds

Given two identical cameras mounted rigidly such that the distance between their lenses remains constant, a 3D projection of each pixel on the intersection of the images from the lenses can be obtained, by extrapolating disparity information using the pinhole camera model. This model is described in detail in section titled *Integrating Points* in this chapter.

In order to obtain 3D points from the camera images, the software package SVS is used. For each frame SVS generates four images, a left and right colour image as well as a grey-scale image for each lens. It then uses the grey-scale image to calculate point disparities.

SVS is distributed as a set of binaries, some with graphical interfaces, which can be used to collect images and point-clouds, and display them in 3D, as well as C++ header files and libraries for linking with other code.

Initially, this phase of the algorithm was done in the main loop of the program, however, to permit faster testing and isolate the code (SVS is prone to segfaults, debugging the project with that as part of it would have been nightmarish), the data acquisition step was moved out, into a separate program that generates binary files. These files are then read into the main program, allowing the processing to be done off-line. Online processing is entirely possible by simply moving the data acquisition loop back into the main program.

Before a camera can be used with SVS, it must be calibrated. This calibration step is required to adjust various parameters of the algorithm to reflect the physical characteristics of the camera. To calibrate the camera, a chequered board is placed at various angles and distances in front of it, and the `smallvcal` program is run using the images obtained. The software then tries to automatically generate a set of calibration parameters from observing this images. In addition to this calibration step, parameters such as window size, the X offset of the horopter, and the maximum disparity need to be set at run time. Getting

these settings correct is quite a difficult task, it's very easy to generate lots of data with very little 3D information associated with it, or even worse, to generate a data set with lots of incorrect 3D information.

SVS is capable of generating 3D points for any set of 4 images. In order to get better quality 3D points to facilitate testing, sets of images were created artificially, using Povray, and fed into SVS. Using such data for testing allows modelling an ideal situation, free from noise and lighting distortions, allowing a benchmark for performance to be set.

5.4 Filtering and selecting points

Two algorithms for feature detection were used. Using a Harris corner detector was proposed in the paper by Garcia *et al*, and was the first option considered.

A Harris corner detector finds a corner's "strength" by computing the locally averaged moment matrix from the image gradients, then combining the eigenvalues of such matrices. The maximum values indicate corner positions, in other words, corners are pixels with the highest corner strength.

The `cornerResponseFunction` in the Vigra [5] imaging library is used to implement this function.

One of the problems with using corner detection was that it yields too few points. Images typically have few corners, and not all of these will be picked up. Scenes with mostly rounded features would be particularly problematic with this approach. SVS too, typically generates 3D points for less than 10% of the pixels in an image, this makes the problem two fold, the corners that the Harris Corner detector picks up must have corresponding 3D points, otherwise they have to be discarded. If ICP is run with too few points, it yields inaccurate results, as these points are statistically more likely to not be exact matches.

In order to solve this problem, the corner detection was substituted with a Canny Edge Detector [4], again, from the VIGRA library.

A Canny operator first smoothes the image using a Gaussian convolution, then applies a 2D first derivative operator to it, in order to highlight the points with high first spacial derivatives. Edges give rise to ridges in the gradient magnitude image. The algorithm then tracks these ridges, with certain thresholds, giving rise to edges in the output image. Examples of the output of both the Canny edge detector and the Harris Corner detector can be found in the results chapter.

Using an edge detector also reduces the problem with frills. In the previous chapter, Figure 4.1 on page 29 showed SVS picking up segments of the background along with the actual foreground image. By only using pixels off edges in the algorithm, the problem of using points which have incorrectly calculated 3D coordinates reduces, as edges tend to have a better chance at having correctly calculated disparity values.

5.5 Matching points

Once the features have been selected from the two consecutive frames, the points need to be paired up, so a transform can be obtained. From here on, the previous frame is referred to as `Frame_a` and the current frame is `Frame_b`.

For each feature in `Frame_a` a 20x20 window of pixels around its location in `Frame_b` is searched. Since the rectangular coordinates of each 3D point are saved in the `PointType` object, this search can be done in constant time. The 3D distance between the candidate points and the current point are also checked and points lying further than 10 pixels away are discarded.

The above operation yields a set of candidate features for each feature in `Frame_b`. Proximity alone is not a good indication of correlation, so at this stage, each candidates' normalised cross-correlation coefficient is found, and the candidate yielding the maximum cross-correlation is picked as the match.

Cross-correlation is an algorithm for determining the location of corresponding image patches, using grey levels, where a reference patch is moved over a search patch until the cross-correlation coefficient is maximised. The cross-correlation coefficient is calculated as follows [27]:

$$\frac{\sum_{r_R c_R}[g_R(r_R, c_R) - \overline{g}_R] \cdot [g_S(r_R + \Delta r, c_R + \Delta c) - \overline{g}_S]}{\sqrt{\sum_{r_R c_R}[g_R(r_R, c_R) - \overline{g}_R]^2 \cdot \sum_{r_R c_R}[g_S(r_R + \Delta r, c_R + \Delta c) - \overline{g}_S]^2}} \tag{5.1}$$

In this equation, $\overline{g}_R$ and $\overline{g}_S$ denote the arithmetic mean grey level in the reference image and the part of the search image which is covered by the search image, respectively.

Since a list of candidate pixels has already been computed, the whole image need not be searched. Instead, the cross-correlation coefficient for 5x5 squares centred at each of the candidate pixels is computed. A vector of all cross-correlation coefficients for the candidate points are computed this way.

This vector is then traversed, and the point that yields the maximum cross correlation is picked as the matching point.

The coefficients of each of these pairings is stored in a vector, which is then traversed to find the median coefficient as well as the Median Absolute Deviation(MAD). To compute the MAD, the distance between each coefficient and the median is calculated, and the median distance is found.

Once a median, MAD and vector of coefficients is found, this list is traversed, and and pairs that have a coefficient greater than 2*MAD*median calculated are discarded. This process usually eliminates about 20% of the pairings found. Once a vector of pairs is computed, it is passed on to the next stage of the algorithm, which calculates the transform.

Other ways of finding matching pairs can be substituted here without affecting the rest of the algorithm. Some interesting ways to do this include using the recently published SIFT algorithm [29] which can find matching features across a substantial number of affline transforms including translations, rotations, addition of noise and changes in lighting. Another very common way to find pairs is by using a KD-Tree. A KD-Trees allows orthogonal range searching by partitioning the point set. Each node in the tree is defined by a plane through one of the dimensions that partitions the set into left/right and up/down sets. The children are then partitioned into equal halves, and the algorithm continues until the height of the tree is `log n`, at which point the nodes become leaves. This data structure allows finding nearest neighbours in $O(\sqrt{n} + k)$ time, where n is the number of points, and k is the number of points in the result. Construction of a KD Tree takes $O(nlogn)$ time, and $O(n)$ storage.

5.6 Obtaining a transformation matrix

Having generated a set of point pairs, a transform needs to be calculated which transforms the points in the current set, into the coordinate space of the points in the previous set. To do this, the Iterative Closest Point (ICP) algorithm's registration phase is used. This algorithm iteratively reduces the point-to-point distance between the features from each point-cloud, using a least-squares minimisation function. This implementation uses the Singular Value Decomposition (SVD) version of the ICP algorithm.

The ICP algorithm produces a 3x3 rotation matrix and a translation vector. These are combined to produce a homogeneous transformation matrix which transforms a point in the coordinate system of `Frame_b` into those of `Frame_a`:

$$ {}^{a}\tilde{M}_{b} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.2} $$

The code to find this transform was contributed by Raymond Sheh. His code was modified slightly to fit in with this algorithms pipeline, as well as to use the data structures found in the rest of the project.

The above registration phase will yield an exact transform if the input pairings are all correct. If there are correspondences that are not correct, this algorithm will not minimise the distances between all points to zero.

Since grey-level cross-correlation does not guarantee correspondence, this transformation is only an approximation to the true relationship between the frames. In order to refine this approximation, pairs that don't meet two correspondence criteria are discarded. These criteria are:

1. The two corresponding points must be close enough in 3D space when they are referred to in the same coordinate frame. In practice, a pair of points $({}^{a}P_i, {}^{b}P_i)$ is discarded when their distance $d({}^{a}P_i, {}^{b}P_i)$ deviates from the median distance between all current pairs by more than two MADs. The following formula is used to find this distance:

$$d({}^{a}P_i, {}^{b}P_i) = \| {}^{a}P_i - {}^{a}\tilde{M}_b \cdot {}^{b}P_i \| \tag{5.3}$$

2. The pixels corresponding to the points need to be similar in colour. If the distance between the RGB values of the pixels differs by more than twice the MAD of all points, they are discarded.

Each time these criteria are checked, and the points that don't pass them are discarded. A new matrix ${}^{a}\tilde{M}_b$ is generated with the remaining points. Garcia *et al* stop this iterative process once no points are discarded. This means that sometimes all but 2 or 3 points will be discarded, yielding random transforms. This rapid degradation of the size of the pairings means that the results can potentially become very inaccurate, if the wrong pairings are discarded. This was seen to happen fairly often in practice.

In order to improve the situation, the heuristic recommended by Garcia *et al* was relaxed a bit in this implementation. Instead of stopping when there is no change in the size of the pairings, this implementation stops when the difference in size is less than 10. This doesn't solve the above mentioned problem, but it does ease it somewhat.

Once the iteration is terminated the value of ${}^{a}\tilde{M}_b$ is assigned to the final transform matrix ${}^{a}M_b$ which is used to generate the world transform for the frame b, M_b:

$$M_b = M_a \cdot {}^{a}M_b \tag{5.4}$$

Where M_a is the transform matrix of `Frame_a`, the previous frame. This recursive definition is valid since the initial M_a is the 4x4 identity matrix, i.e. the first frame is aligned with the first camera position.

5.7 Integrating points

Once a transform matrix is obtained, the points in the frame need to be integrated into the world model. The integration phase looks at several frames in the world, and compares the points in the current frame to the ones observed before. Points in nearby frames are checked against each other, and their positions are adjusted according to the information in the current frame. This integration phase is best explained using the following pseudo-code 5.2:

```
for(τ=0; τ<k; τ++){
integrate_frame(current_frame, frames[τ]);
integrate_frame(frames[τ], current_frame);
}
```

Figure 5.2: Pseudo-code for integrating frames

In this implementation, k is set to 3, in other words, each frame is integrated with three frames before it, when these frames are available.

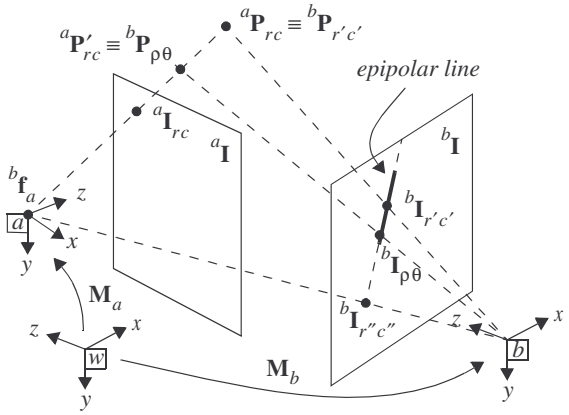


Figure 5.3: Geometrical relationships between images acquired at two camera frames a and b respectively. A 3D point observed from the viewpoint a defines an epipolar line in image ${}^bI$ [17]

Figure 5.3 illustrates the relationship between two camera frames. Each pixel has an image coordinate I_{rc} and a corresponding 3D coordinate P; ${}^aP_{rc}$ represents the point in `Frame_a` whose projection falls on the coordinates (r,c).

To find a point that matches the current point in `Frame_a`, the point is projected to it's position on the image associated with `Frame_b`. Points along the epipolar line, which stretches from the projection of this point, to the projection of the focus of the frame are checked, and the one with the most normalised cross-correlation is picked as the match. Once this point is found, its position is changed to the X, Y and Z coordinates of the current point.

In order to find the disparity of pixels, and generate a 3D point, the SVS software uses the pinhole camera model, so it is natural to use this transform to go from the 3D point coordinates, to 2D image coordinates. The pinhole camera function μ is defined as follows $(r, c) = \mu({}^aP_{rc})$: :

In the case of pure rotation ($z \approx 0$)

$$\mu({}^aP_{rc}) = ([\beta], [\gamma]) \tag{5.5}$$

In case of combined translation and rotation

$$\mu({}^aP_{rc}) = ([\alpha y/z + \beta], [\alpha x/z + \gamma]) \tag{5.6}$$

Where $\alpha\beta$ and γ are calculated from any pair of available points ${}^{a}P_{r_1,c_1}$, ${}^{a}P_{r_2,c_2}$:

$$\alpha = (c_2 - c_1)/(x_2/z_2 - x_1/z_1) \tag{5.7}$$

$$\beta = r_1 - \alpha y_1/z_1 \tag{5.8}$$

$$\gamma = c_1 - \alpha x_1/z_1 \tag{5.9}$$

The `integrate_frame` function does not modify its second argument, instead it shifts the points in the first argument into line with their corresponding points in the second.

In order to represent an arbitrary point ${}^{a}P_{rc}$ in the reference `Frame_b` the following transform can be performed:

$${}^{b}P_{r'c'} = M_b^{-1} M_a {}^{a}P_{rc} \text{ where} (r', c') = \mu({}^{b}P_{r'c'}) \tag{5.10}$$

The integration algorithm proceeds as follows: first the focus of each frame is found, the focus of a frame is the last column in its homogeneous transform matrix. Let f_a denote the focus of the first frame (`Frame_a`) and f_b denote the focus of the second frame (`Frame_b`). From the previous equation, it follows that to get f_a in terms of `Frame_a`, a similar transform must be performed, i.e. ${}^{b}f_a = M_b^{-1} f_a$ and $(r'', c'') = \mu({}^{b}f_a)$. ${}^{b}I_{r'',c''}$ is the projection of this point, onto `Frame_b`.

Then each point ${}^{a}P_{rc}$ is shifted into its correct position ${}^{a}P'_{rc}$ in `Frame_b`. This is done using the following algorithm:

1. Find ${}^{b}P_{r'c'}$ by using the equation above.

2. Traverse the epipolar line which joins the points (r', c') and (r'', c'') using Bresenham's line algorithm. Add points which are within 2 pixels of this line to a list of candidate points.

3. Go through the candidate points, finding the one which has the maximum normalised cross-correlation with ${}^{a}I_{rc}$; call it ${}^{b}I_{\rho,\theta}$.

4. Look around the pixels around ${}^{b}I_{\rho,\theta}$ in a small centroid, in order to find a matching 3D coordinate. Calling this point ${}^{b}\overline{P}_{\rho,\theta}$

5. Traverse the line joining ${}^{b}f_a$ and ${}^{b}f_{r'c'}$, and find the point closest to the point: $({}^{b}\overline{P}_{\rho,\theta} + {}^{b}P_{r'c'})/2$. Call this point ${}^{b}\overline{P}_{r''c''}$.

6. Find the corresponding point to ${}^{b}\overline{P}_{r''c''}$: ${}^{a}\overline{P}_{r,c} = M_a^{-1} M_b {}^{b}\overline{P}_{r''c''}$.

7. Finally, replace the original point ${}^{a}P_{rc}$ with this new estimate, ${}^{a}\overline{P}_{r,c}$

By applying the above algorithm to each point in the frame, over the last few frames, the points come closer together, getting rid of some of the drifting errors caused by inaccuracies in stereo vision. Once this process has been repeated k times, as per the original pseudo-code, the current frame is added to the `World_Map`.

5.8 Modelling the solution

Currently, the world is modelled using a vector of `Frame`s. The model is visualised by incrementally dumping the contents of all frames into a file, which is later viewed using a pyVTK script. The path of the robot can be found by simply looking at the last column of each frame's transform matrix, as this vector represents the focus of the frame.

This is clearly not an efficient way to store the points, and in the future, the points should probably be stored in some sort of tree, a KD-Tree or an Octree, so that repeat points can be amalgamated, and outlying points, which are probably miscalculations, can be discarded before rendering.

An interesting thing to note here is, in VTK RGB values are normalised to 1. This means that all the RGB values obtained from SVS need to be divided by 255 before being displayed by VTK. Not doing this yield a lot of mysterious dark matter in the 3D render.

5.9 Testing Framework

The testing framework can basically be divided into two aspects. Testing individual functions, and testing the algorithm as a whole. To do the former, a small set of unit tests were developed.

The following tests were performed:

- Cross Correlation Function: two tests were written, one that find the cross correlation coefficient for a single item in a 1 dimensional array, and another which locates the centre of a cross in a 2 dimensional array.

- Point Matching: Two consecutive frames in a test run were picked, and the pairings between them were plotted.

- ICP: First a set of points were transformed by a known matrix, then the ICP registration phase was run on them, to check if the same matrix is produced. Noisy data was progressively added, to see how robust the heuristic termination function is.

As well as the above tests, many tests were written to discover the behaviour of SVS, whereby output from the program was written to image files, and inspected. These tests uncovered some interesting facts, possibly the most enlightening was that the grey-scale

image in the SVS data structure does not match with the colour image. The grey-scale version is shifted slightly to the right. This played havoc with other sections of the algorithm, until it was discovered.

Once the various components of the algorithm were tested, full runs through the entire pipeline needed to be performed. To get clean looking data, and test the peculiarities and characteristics of the SVS binaries, Povray was used to ray-trace and animate some images. A script called `makestereo.py` was written to convert the output of Povray into a format SVS can understand (four bitmaps, two colour, two grey-scale, the colour one being 8bit). The Povray animation basically simulates the movement of a stereo camera with its lenses 10cm apart. The Povray source file also provides a very concrete ground truth to compare the output with.

Tests on data obtained directly from the camera were also performed. In order to get meaningful results from the camera images, the parameters had to be adjusted carefully, and the conditions needed to be ideal. SVS can get quite confused when there are multiple lighting sources present, or there is a lot of background texture.

The particular set of test data that was used for most of the testing was that of a set of cubes in a white area. Tests of more complicated settings are left to future work.

To view the output of these tests, another Python script `render.py` was written which renders the points found in a file written by the algorithm, and allows interactive rotation, scaling and translation. A box is displayed showing the axis, and makes moving and rotating the points a lot easier.

The results obtained from the tests above are presented 6.4.

6 Results

This chapter presents the testing framework for this implementation, and shows results obtained.

Since the basic steps of the algorithm are fairly independent, each of them has been tested individually. For some stages, multiple implementations were done, and the results of these different implementations will be presented, as well as the performance of each part of the algorithm on its own. Finally, results obtained from running the whole algorithm will be shown.

All the results shown here were obtained on a system running Debian Linux with the 2.6.10 kernel. Other than endianness issues associated with writing binary files, the main code-base is platform independent.

6.1 How results were obtained

6.1.1 Povray

In order to evaluate the algorithm's basic functionality, an idealised test framework was created using *povray*. This was done by ray-tracing animations from povray scene files and feeding them into the SVS software system.

Idealised situation

By using ray traced images, a lot of problems caused by lighting are avoided. Also avoided is dealing with intrinsic properties of the camera, such as lens distortion, and the camera set up. The lenses are always in focus, identical and parallel.

This however, does not guarantee accurate 3D points. If the surfaces being "seen" are smooth, the algorithm misses them completely. If the objects are either too close to the camera, or too far away, their disparities are miscalculated entirely. In one instance, an image containing two cubes was observed to have one cube at the front (correct) and another stretched out between the first, and where it should be. This is shown in Figure 6.1 on the next page and Figure 6.2 on the following page.

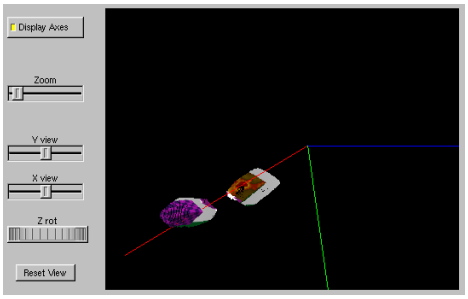


Figure 6.1: Incorrectly identified second cube - 3D window

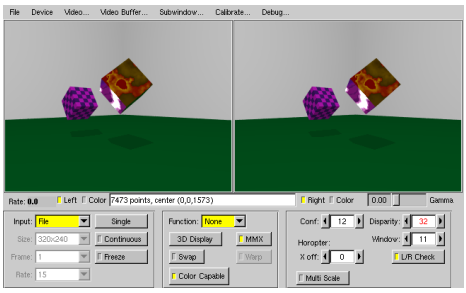


Figure 6.2: Incorrectly identified second cube - SVS window

In order to get nice values from SVS, an idealised environment was created, diffused lighting (soft shadows) and desired surfaces textured.

Even with the best of settings, SVS is still a heuristic, it doesn't give idealised points. The 3D point clouds are littered with erroneously calculated values, which give even the simulations a aura of realism.

6.1.2 Real World

The world is a scary place. In order to get nice, clean data from the camera the parameters have to be set properly, the lenses must be completely in focus (nudging them changes their focus), and the movement must be slow, and smooth. There must only be a single light source, windows and sunlight confuse the software. One must be careful to not cast a shadow on the scene while photographing it, and, above all, the data collection must be done as quickly as possible, as the SVS user programs tend to segfault fairly often, and as soon as everything is configured.

Once the data was collected, it was stored on a CDROM and reused for a multitude of tests. Although several CDs worth of data were collected, only one or two sets were actually usable (the other sets had bad lighting which confused the stereo software), and this is the data used in the tests described below.

6.1.3 Unit Tests

A series of unit tests were designed to test various stages of the algorithm independently. The results below show a few of the outcomes of such tests.

6.2 Performance of Different Stages of the Algorithm

6.2.1 Obtaining point clouds

SVS was used to obtain point clouds, Figure 6.4 on the next page shows the point-cloud obtained by using an image generated by povray. This shows that even using povray data, mistakes are quite common.

Data obtained from the camera is shown in Figure 6.3.

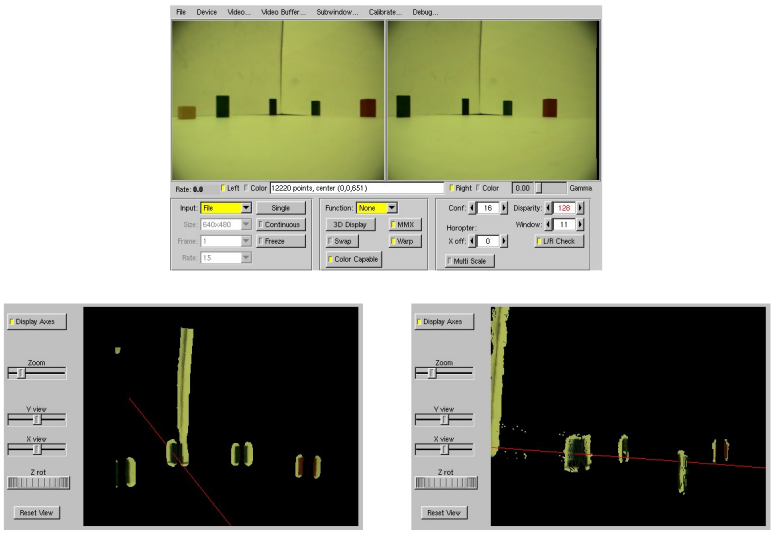


Figure 6.3: A set of blocks and their 3D projection in SVS

6.2.2 Filtering and selecting points

Figure 6.7 on page 48 shows the Harris corner detector being run on the image in figure 6.6 on page 47. It was found that when using a Harris corner detector, an average of 31 corners had a corresponding 3D coordinate. By using a Canny edge detector, this number was increased to 561. Figure **??** on page ?? shows the Canny edge detector being run on the same image. The graph in Figure 6.5 on page 47 illustrates the difference.

The rises and falls in the graph can be attributed to the 3 objects rotating, sometimes occluding each other from view. A number of observations can be made from this graph:

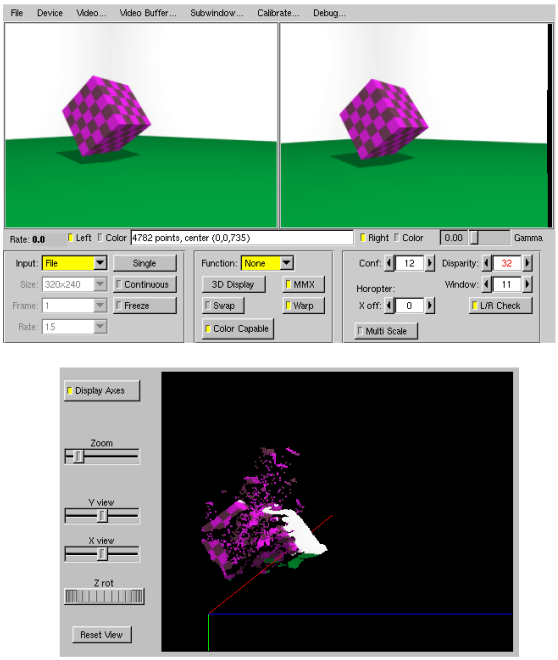


Figure 6.4: A povray rendered cube and its 3D projection in SVS

- The method using the Canny edge detector always superior to the Harris algorithm, even in minima. This clearly shows the advantage of using the Canny detector.

- In both methods, there is a huge variation in the number of points detected. These maxima and minima correspond to the number of objects visible to the camera. This means that relying entirely on geometric representations for determining feature saliency may not be a good idea, as there is a good chance not enough points will be picked up.

6.2.3 Matching points

Initially a unit test was written to test the cross correlation function. The images in Figure6.9 were given to the `max_cross_correlation` function, the centre point given to the function was the point `(3,1)`, the centre of the cross in the right hand image, and the search offsets were given as all the pixels in the left hand image, the algorithm correctly identified the pixel at `(3,2)` in the right hand image as the corresponding point. Various different window sizes were tried, and correspondence with self was tested. All results were positive.

Even though the cross correlation function works properly, there is no guarantee of correct correspondences when using it. Figure 6.11 shows two greyscale images, with the pairings shown in coloured dots. This image illustrates that although using cross-correlation is a reasonable heuristic, it is by no means perfect. Many points in a 100x100 window can potentially look identical. A good example of this is shown in Figure 6.10. Because of

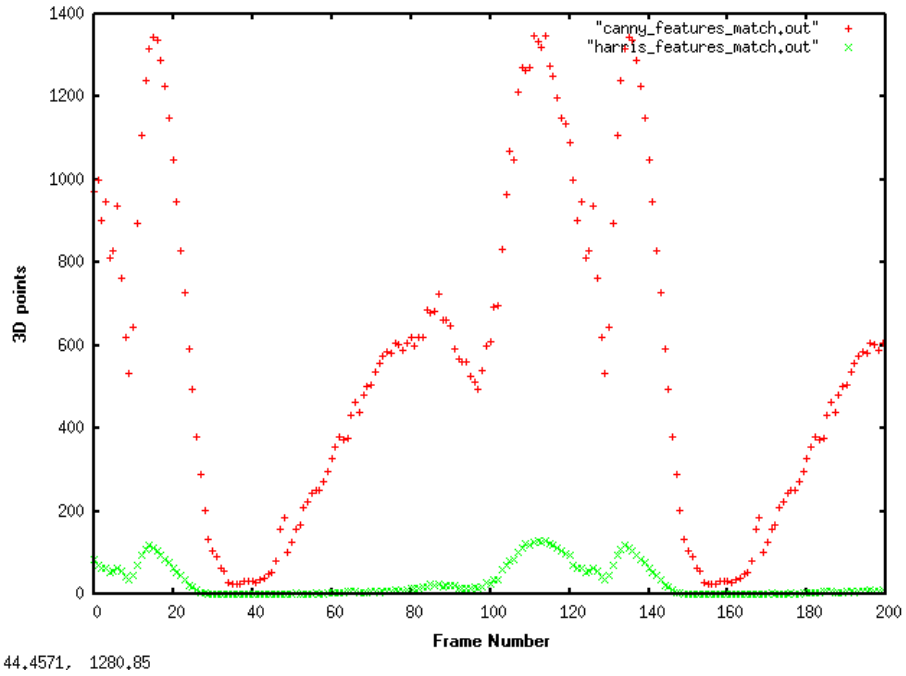


Figure 6.5: Graph illustrating the number of 3D points that match features for each frame

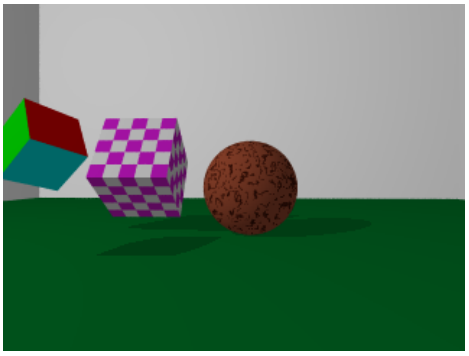


Figure 6.6: Scene which feature detection was performed on

the shadow, the corner in the wall will be picked up as an edge, however, the colouring is uniform from the ground up. Since the `match_pairs` function searches from the upper left corner of a window, down to the lower right corner, the point with the highest correlation will be the lowest point in that window. This gives a bias towards lower points, skewing the heuristic. Similarly, with grass, right most points will be favoured by the algorithm. This same issue also comes up with uniform patterned images. There are two cubes shown in 6.11. The left hand side, chequered one has plenty of 3D point associations found by SVS. In contrast, the right cube barely has any, and there are no matches between the 3D points and the edge points, so it gets ignored completely in this stage of the algorithm. The trouble with the left hand cube, however, is that the different squares on it all have very highly correlated with each other, causing the same bias that appears with the walls to happen there. There are a lot of mismatches between those frames.

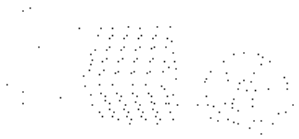


Figure 6.7: Sphere and cubes image after corner detection (in reverse video)

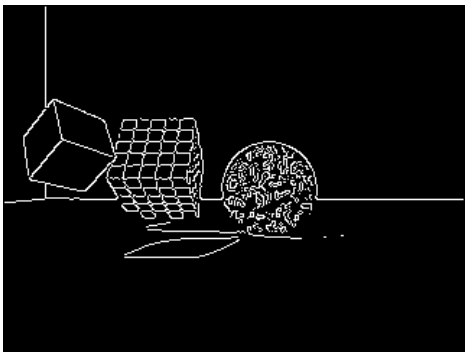


Figure 6.8: Sphere and cubes image after edge detection

The SIFT algorithm was run on the same pair of images, and the results are shown in figure 6.12 on page 53 for comparison. The SIFT algorithm does both feature detection and correspondence, so it could take the place of both the `max_cross_correlation` and `match_pairs` functions. Close inspection of the image reveals that even SIFT has a lot of trouble with it too.

6.2.4 Obtaining a transformation matrix

The ICP algorithm was used to calculate a transform matrix. As explained in the previous chapter, the ICP registration stage will only compute a correct transform if the pairings are all correct. This condition will of course never be satisfied initially, unless the pairings are generated artificially. This was tried in a unit test, to verify the algorithm was implemented correctly.

The iterative part of the ICP algorithm happens when the pairings are altered in some way, and the registration phase is rerun. This process happens until some convergence criteria is met. In the algorithm proposed by Garcia *et al*, pairings are not recomputed; instead, each time the registration phase is run, the distance between the members of each pair (after being placed the same point of reference) is computed. The median distance, and the median absolute deviation (MAD) of the distances are computed, and pairs which have a distance that exceeds twice the MAD plus the median are discarded, and the algorithm

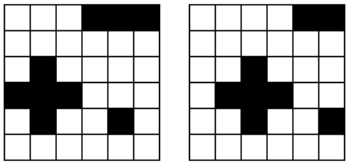


Figure 6.9: Images used to test the cross-correlation function

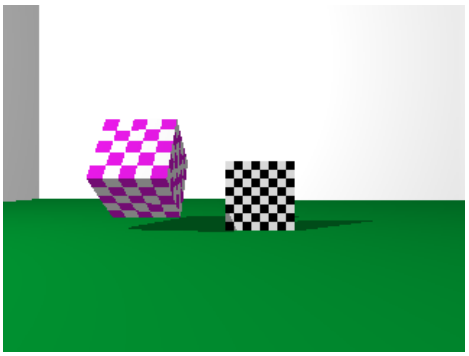


Figure 6.10: Povray cube shown next to the corner of a wall

runs again.

Once no more pairs are discarded, the algorithm terminates and the transform matrix is returned. An experiment to test this heuristic was run as follows:

1. Generate a set of points.
2. Transform them all by a known transform matrix M.
3. Run ICP algorithm over them.
4. Pick 3% of the pairs at random.
5. Swap the values of these pairs randomly.
6. Repeat from step 2.

The results from the above algorithm are shown in figure 6.13 on page 54.

In this graph, each iteration represents 3% of the points being altered randomly; thus after 35 iterations, the data is essentially completely random (changing 3%, 35 times, $0.03 * 35 = 1.05$, so just slightly over 100% of the points have been altered). This graph shows that the mean error this version of ICP follows the curve $146 * (1 - 0.97^x)$ with respect to the number of random pairings, meaning that when random pairings are given, the mean distance will roughly equal 146 units. In order for the algorithm to converge, the pairings must have fairly high certainty, otherwise errors can not be recovered from through the iterative process, at least, not with the heuristic proposed by Garcia and used in this algorithm.

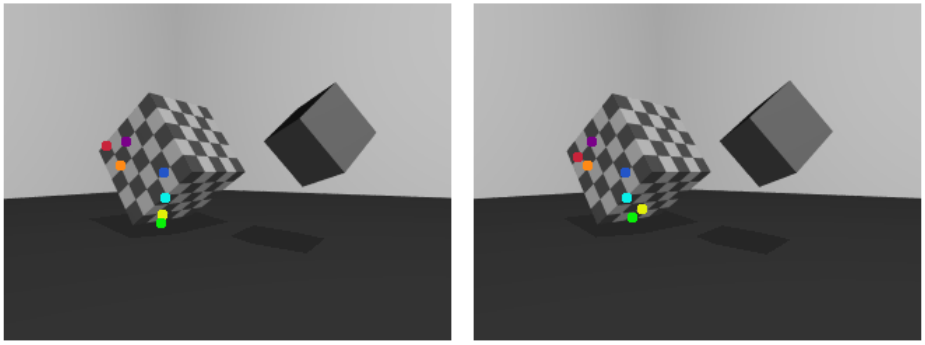


Figure 6.11: Two consecutive grey scale images, with correspondences marked in colour

ICP not converging correctly is a major problem with this approach. Even if the matrix is just a couple of pixels off, this adds up over many iterations, producing a 'snake' effect, as seen in figure 6.14 on page 54. This effect is characterised by a series of objects being mis-matched by just a few pixels at each iteration. In the end, instead of having a single object such as a cube, a whole series of them is produced.

A potentially more serious problem however is the ICP algorithm not just getting it a little wrong, but getting it completely wrong. On each iteration, pairs that deviate too far from the median distance are discarded. This heuristic works if most pairs are correctly matched up, and the wrong pairs are removed each time; however if the majority of pairs are incorrectly identified, or there's a large cluster of incorrect pairs found (perhaps due to noise, or an incorrectly identified feature), the algorithm can delete the wrong pairings. Eventually, the number of pairings decays down to single digits, sometimes just two, and this produces an essentially random transform. This can be enough to totally flip the points in any direction, and cause general havoc. This problem is discussed in greater detail in the evaluation section.

6.2.5 Modelling

The results were modelled in both colour, and in monochrome. Monochrome modelling had the advantage that the general shape of the structures was more easily visible, with the distractions of colour. Colour however let exact correspondences be seen with greater ease.

6.3 Results From Whole System

This section describes the outcomes of running the entire algorithm from start to finish. Again, the results are derived from both simulated, and real data.

Povray Images

In one experiment, the scene shown in figure 6.15 on page 55 were put through the SLAM algorithm. The initial state of the world, after a single frame can be seen in figure 6.16 on page 55. Here some of the chequered cube is occluded by the coloured cube. Even at this stage, SVS mistakes can be observed.

After 3 frames, more points have been integrated, as seen in figure 6.17 on page 56. The green and red colours on the coloured frame are visible here, and have been correctly matched with previous frames.

After 7 frames the result is shown in figure 6.18 on page 56. The problem with walls is really becoming visible in this frame. Even though the structure of the cubes are correct, the colouring is not. This is because the disparity of the walls, and of the cubes are both calculated incorrectly. Because an edge detector was used, these points haven't been used for calculating the transform, but their presence still makes the image look noisy.

After 18 frames, there are a lot more points in the map. This is shown in figure 6.19 on page 56. Again, structurally the cubes are sound, however the colours look even more noisy.

Images from the camera

In order to test the algorithm on a real camera, the stereo camera was moved around the scene pictured in figure 6.20 on page 56. Figure 6.21 shows the initial position of the camera. In this frame, only 3 of the 5 blocks are visible.

After 4 frames, and extra two blocks (the yellow block, and green triangular prism) are added to the scene. Figure6.22 shows that the points belonging to the previous blocks have all been integrated in correctly. The column visible at the back is the crack between the two slabs of white cardboard placed behind the scene. It was probably ignored by the pairing algorithm as the cross-correlation between the pixels of adjacent frames would have been too low, the grey visible is just a shadow that would have changed in colour and position.

Even after 30 frames, the blocks are still well formed. There is little error visible from mismatched pixels and the positions look correct. This remains the case for at least another 40 frames or so.

6.4 Running Time and Resource usage

The following benchmarks were taken on an AMD Athlon XP 2600+ (1913.142 MHz) running Debian Linux 2.6.10. This machine has 1 GB of RAM and an additional 500 MB of swap space.

The implementation in this thesis is divided into two programs, `read_svs` which interfaces with the SVS binaries, and collects stereo data. This program then writes the data to binary files, which are read by the second program `slam` which runs the SLAM algorithm. It is worth mentioning that `read_svs` also performs feature detection.

Without writing to the disk, `read_svs` took 12.03s in total to process 200 frames, each of size 320x240 pixels. Writing to disk increased this time to 13.50s. This equates to about 0.0675 seconds per frame.

The `slam` program spends around 10-14 seconds per frame, however, much of this is due to writing to disk. When writing to disk was turned off, the program processed 200 frames in 1350.50 seconds, which equates to just over 6.75 seconds per frame. All together, the program can process a single frame in under 7 seconds. This is comparable to Garcia *et* al's result of 12 seconds per frame, on an 800Mhz CPU.

The program uses a fair bit of memory, mostly because all frames are kept in memory until the program terminates. When processing the 200th frame, the program was using about 53% of the system's available memory. This number can reduced dramatically by not storing redundant information about points by storing them in a tree structure.

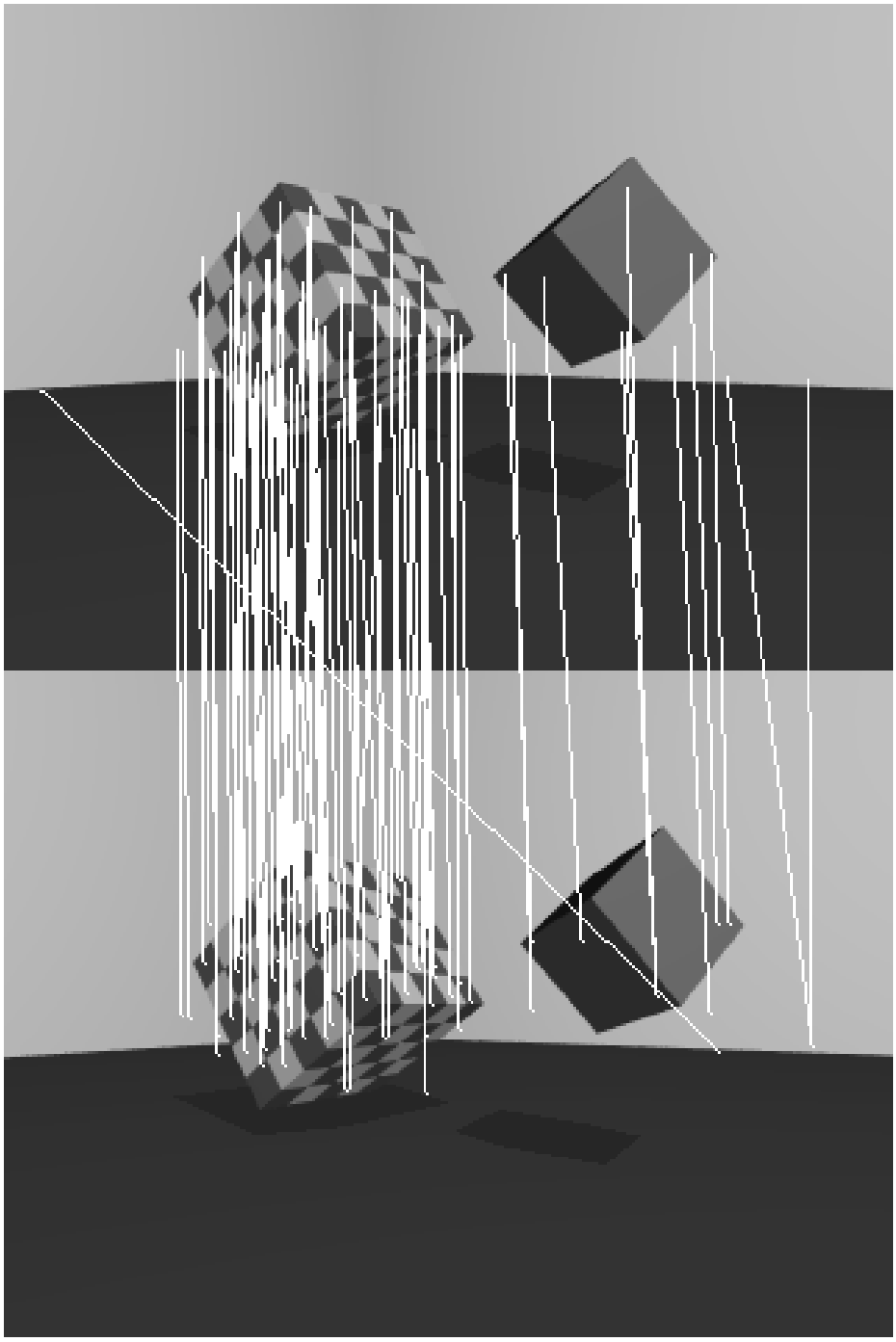


Figure 6.12: Two consecutive grey scale images, with correspondences found using SIFT

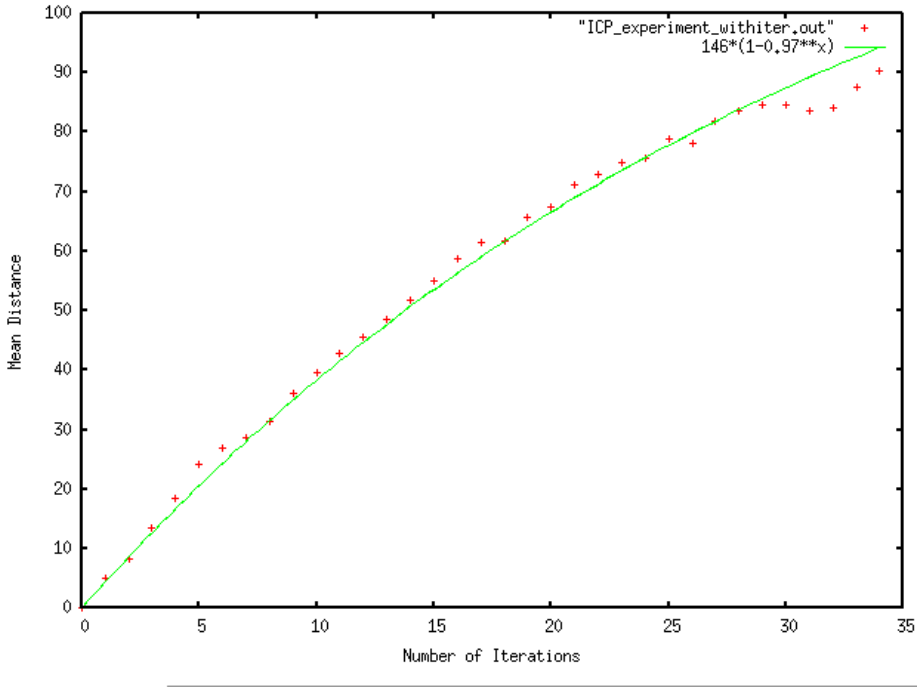


Figure 6.13: Results of experiment to verify the robustness of the ICP convergence heuristic

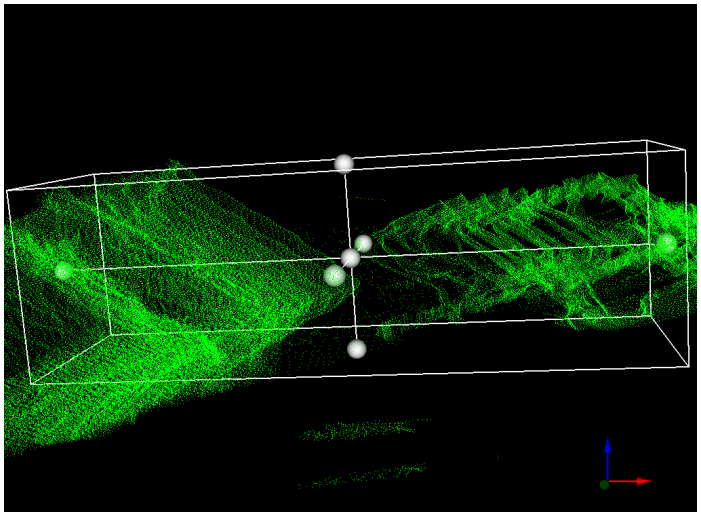


Figure 6.14: “Snake” like output from several iterations of the algorithm

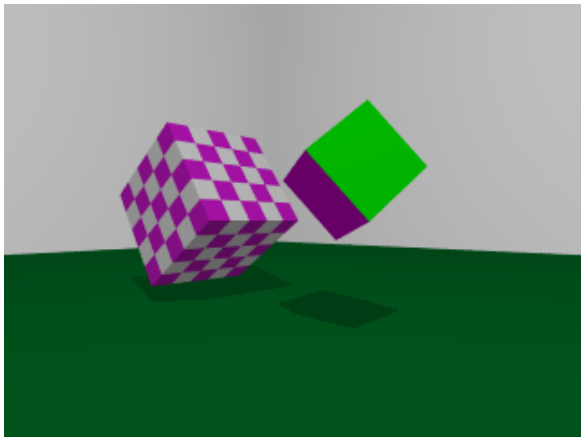


Figure 6.15: Two cubes, as rendered by Povray

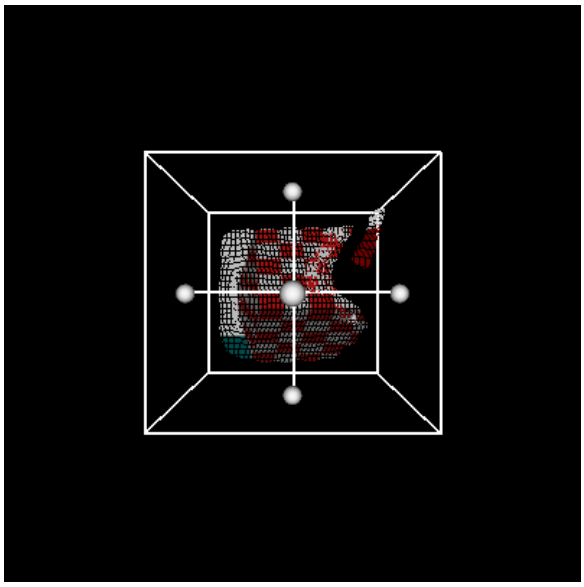


Figure 6.16: Two cubes, initial 3D points

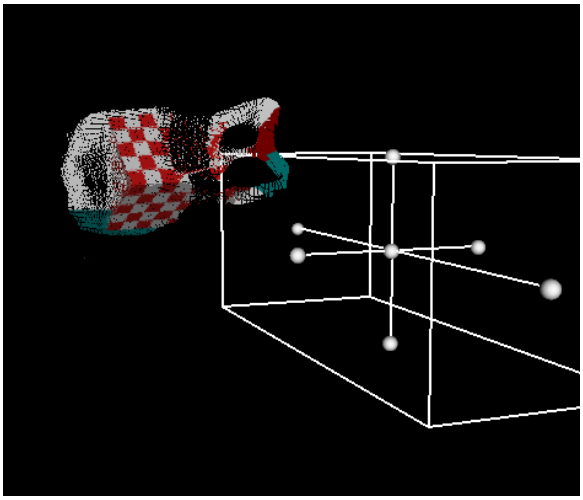


Figure 6.17: Two cubes, after 3 frames

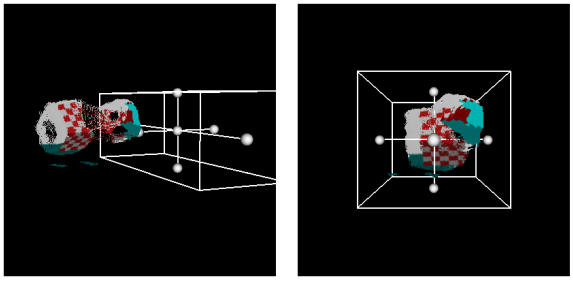


Figure 6.18: Two cubes, after 7 frames

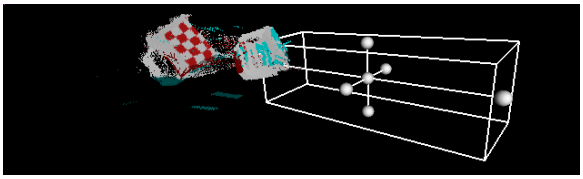


Figure 6.19: Two cubes, after 18 frames

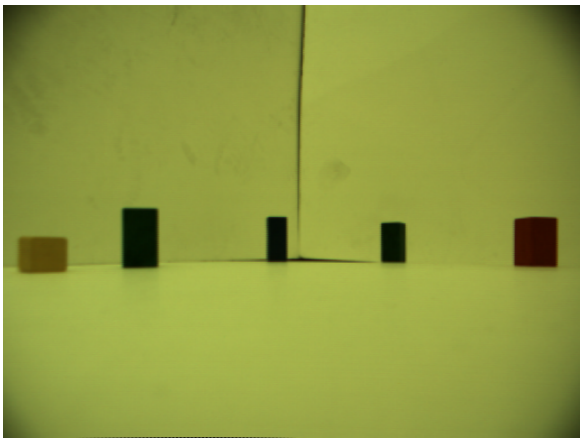


Figure 6.20: A view of all 5 blocks in the scene

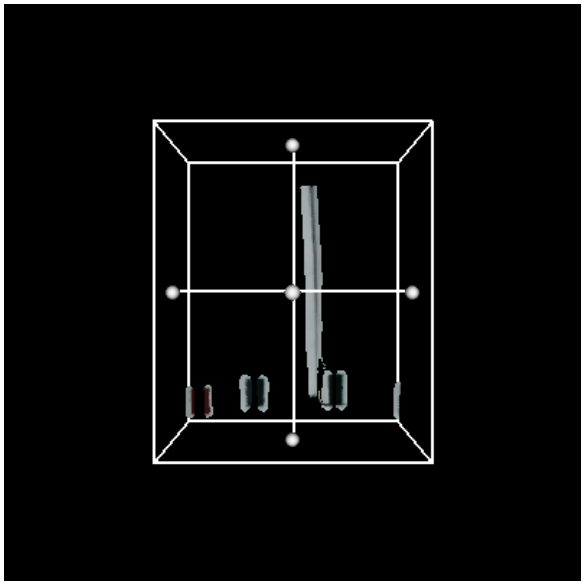


Figure 6.21: Initial camera frame

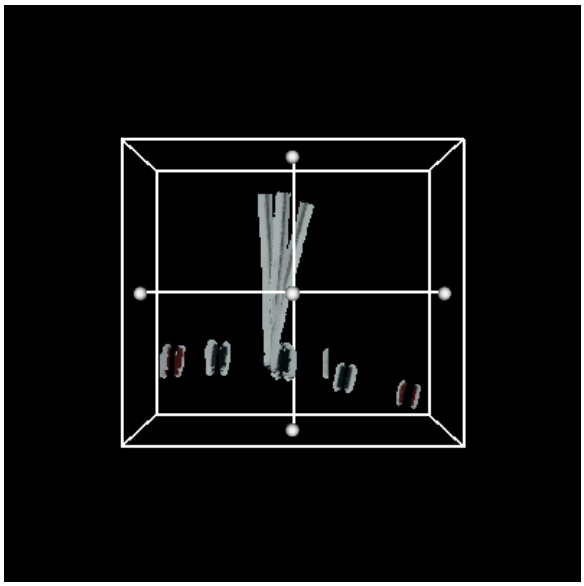


Figure 6.22: Frame 4, two extra blocks detected and placed on the map

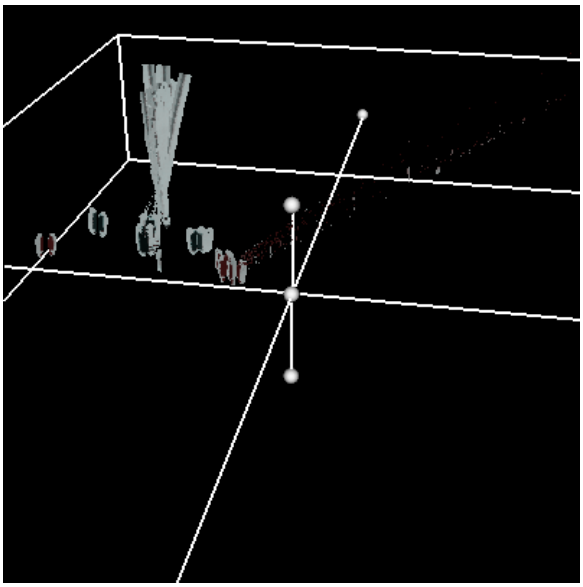


Figure 6.23: Frame 30, All 5 blocks visible, and still correctly placed

7 Evaluation

In the previous chapter, results from running various parts of the system, as well as the algorithm as a whole were presented. This chapter aims to extrapolate on these results and evaluate both the strengths and the weaknesses of the system developed.

The aim of this thesis was to design and implement SLAM on the TXT-1 Monster truck. The goals were initially stated as below:

- Use a sensor that is small in both weight and footprint.
- Build a map that allows for motion with 6-DoF.
- Accurately localise on the generated map, at each stage of the process.

The entire implementation of this thesis is based around the use of a stereo camera head. This apparatus has been successfully fitted on the truck, and thus the first criterion has been met.

The system developed is capable of localising and mapping regardless of the angle or direction of motion, so long as it's smooth. Jerky movements produce sudden blur on the camera, making the task of feature extraction and matching far more difficult. Techniques such as smoothing the image before feature extraction can be used to compensate for this situation, but it is ultimately a problem inherent to vision based techniques.

At each step of the algorithm the pixels that have had their 3D coordinates computed are transformed, and then plotted on a world map. This map can then be used to reconstruct a scene, or to locate an object of interest such as a victim in a disaster site. This map however, is not a full map of the environment, and cannot be used to perform autonomous navigation. Due to the nature of stereo vision, smooth un-textured surfaces are not detected at all. This means that if a robot to solely rely on data obtained from a stereo camera to navigate, it will almost certainly run into a smooth surface such as a wall. Of course, this system could quite easily be extended to take advantage of sensors capable of providing more robust feedback on range. By complementing this technique with data obtained from a range sensor, such as a laser range finder or swiss ranger (a small time-of-flight based sensor measuring range), full mapping can be performed.

Using ICP at each stage of the process ensures that a localisation matrix is generated for every frame, and thus, for every position of the robot.

In implementing this algorithm an extensible system was designed. Any stage of the al-

gorithm can be replaced with an alternative implementation quite easily and tested using the testing framework developed. There is no coupling between the source of the point-cloud, and the rest of the algorithm, so the input method and sensors used can be altered trivially.

Other than mounting a correctly calibrated camera, with its lenses in focus, no calibration is necessary to bootstrap this system. The agent on which the camera is mounted can also be altered with no ill effect, so theoretically this solution should be able to use any sensor or combination of sensors that provide range, bearing and colour data.

Much of this thesis was based on the technique proposed by Garcia *e*t al [17], however, certain improvements to parts of the algorithm yielded far better results. In the second stage of the algorithm the feature detection was done using a Canny edge detector, instead of the Harris corner detector recommended by Garcia. The main problem with the Harris corner detector was that it produced far too few points. This meant that very few of them had associated 3D points, which in turn meant that the ICP algorithm had very few pairings to start off with. Starting off with way too few pairings, and discarding many of them in the iteration step meant that the transform matrix produced was essentially random. The Canny detector produces far more points, and these points have greater correspondence to 3D points generated by SVS. By using a Canny detector the performance of the algorithm increased dramatically.

Another improvement made was a slight alteration in ICP algorithm's terminating case. As mentioned before, the outcome of the ICP registration phase becomes fairly unpredictable once the number of pairings drops to single digits. The iterative step of the algorithm eliminates those pairings whose distance in local 3D space differ by more than a certain margin above the median distance. The iteration stops when no more points are discarded after the registration step. This process works well if most of the initial pairings are correct, and those which are not are not clustered together. If the algorithm starts eliminating correct pairs in favour of those which are not, it degenerates very rapidly into ICP using only a few points, in many cases, just 2. In order to delay the onset of such a problem, the stopping heuristic was relaxed somewhat. Instead of stopping only when no eliminations were made, the algorithm stopped when the total number of eliminations fell below 10.

Both the above problems illustrate a major shortcoming of the algorithm — its total dependence on the calculation of a correct transformation matrix. In the implementation section, it was shown that to derive the localisation matrix of `Frame_b`, the following calculation is done:

$$M_b = M_a \cdot^a M_b \tag{7.1}$$

where matrix ${}^aM_b$ is the transform that takes the points in `Frame_b` to the coordinate system of those in `Frame_a` and M_a is the localisation matrix of `Frame_a` and is initially set to the $4x4$ identity matrix. This equation, which is central to the entire algorithm, means that if a single frame's transformation matrix is miscalculated, even by a small amount, all the

subsequent frames will also be wrong. This, at best, causes the "snaking" effect mentioned in the previous chapter, and at worse causes the entire world to flip in one direction or another, due to a completely degenerate matrix.

The ICP registration phase is very accurate in the absence of badly matched pairs, but this accuracy drops exponentially in relation to the number of random pairings in the sample. This makes the previous stages of the algorithm, the feature extraction and matching, extremely important to get right. In future work different algorithms for finding features and pairings should be experimented with, in order to determine what sort of heuristic gives more accurate matchings. The heuristic employed in the ICP algorithm itself also needs careful consideration. The accuracy of the heuristic in the presence of mismatched pairs is essential, and this property can be tested using the methodology developed in the previous chapter.

Even if the feature matching and ICP heuristic are perfected, there is one major contributing factor to ill-formed maps remaining. This is measurement noise and miscalculated 3D values obtained at the stereo processing acquisition and processing phase.

As illustrated multiple times in chapter 6, SVS does not generate perfect 3D points, even if the lenses on the camera are perfect (as in the simulated scenes on Povray). Erroneously calculated points are common, even in the simplest of scenes. When erroneous points correspond to features in the image, they result in serious problems when calculating the transformation matrix. Edge detection helps prevent this to a great extent, however even on edges, miscalculated points are often a problem. When these points occur in places other than edges, the result is a map that's distorted, and looks noisy.

A semi-systematic source of such distortion in the output of the SVS program is the background of a scene. A "frill" of background is often picked up with the coordinates of the foreground image, so that the foreground image looks larger than it actually is, and surrounded by discoloured, random looking pixels. Although these pixels are not used in calculating the transformation, they pollute the final outcome quite severely, at times making it unidentifiable. For this algorithm to produce a robust 3D occupancy grid, the sources of such noise need to be identified, it it has to be systematically removed.

A possible way to do this could be to segment the image into background and foreground areas, and ignore the background pixels completely. This however, is beyond the scope of this document and will be relegated to future work.

In summary, the solution presented in this thesis is a start to solving the problem of performing SLAM using stereo vision. Although it appears that the algorithm is theoretically sound, there is a definite need for improvement in certain stages of it. Once it is coupled with a few extra sensory inputs and more robust ways of dealing with noise, the performance of the algorithm should improve dramatically.

8 Future Work

8.1 Advanced filtering of data

Stereo vision is not perfect, currently, the implementation only removes points that are off at infinity (greater than 400 units away) or are very close to the camera frame.

Better statistical modelling of the data from SVS could result in better maps. SVS, when configured correctly, tends to sample in clusters, so points that are away from clusters are probably noise, and computed incorrectly.

8.2 Blur and noise tolerance

The current system is unable to deal with sudden camera movements which cause blurring. By smoothing the image before doing feature detection, this could potentially be improved.

8.3 Image Segmentation

As mentioned in the evaluation, the cause of much of the distortion in the final output is parts of the background being picked up in the foreground. If there was an accurate way of describing what is foreground and what is background (or which object the different parts of the scene belong to), incorrectly identified points could potentially be filtered out and discarded. In [6] Cole *et* al segment a laser scan according to the amount of "local interest". Some segments are more interesting than others because they come from unusual or surprising parts of the scene. Scan matching based algorithms such as the one presented in this thesis tend to excel when the scan is complex. The opposite is true for feature based representations (a typical EKF SLAM implementation has a very restricted feature set size, adding more features adds complexity).

8.4 Better modelling

Currently the `World_Map` class simply stores the world as a vector of frames. The three main disadvantages of this are:

- There is a huge amount of redundancy. Each point in the image is sampled many times, and all of these samples are stored independently. This means that the algorithm progressively stores more and more data, and the program eventually gets killed by the operating system for running out of memory.

- There's no way to make intelligent decisions about rendering. Currently, every single point gets rendered.

- By having a proximity model of the points, outliers can be filtered out easily. A flat list of points doesn't provide this information

By storing the points in a tree, either a -Tree or an Octree, duplicate nodes can be illuminated. Highly sampled places can also be marked as points whose bearings have higher certainty. Rendering, which is now done externally from a text file can also be done online, showing the model in real-time.

8.5 Using Additional Sensors

By using a range finding sensor as well as a visual sensor the quality of the data obtained could be greatly improved. If range information is available, the problem that stereo vision has with un-textured or smooth surfaces would also disappear, allowing a map that can be used for navigation to be created.

Other sensory devices could also complement the algorithm, for example, the inertial measurement unit on the track could potentially be used to sanity check the ICP transform, and odometry could be used to check if the translation component of the transformation matrix is correctly calculated.

9 Conclusion

The goal of this thesis was implementing a system which does Simultaneous Localisation and Mapping on the TXT-1 truck. To do this an extensive literature survey was conducted, and a solution presented in the survey was implemented.

The algorithm presented in this thesis works with a stereo camera, which can be easily mounted on the truck. The major drawback of this approach was the fragility of the algorithm used. There is a great reliance on the heuristics employed yielding the correct outcome. Another problem was the essentially unreliable nature of stereo processing.

By augmenting this technique by more robust heuristics, and filtering out "bad" data more reliably, the results can probably be improved a lot. Another interesting technique to try is using extra sensors to verify the heuristics employed.

Although the programs developed in this thesis do not accurately localise and map the agent's environment, a framework for future work using a similar technique has been put in place, together with a fairly extensive testing suit for evaluating results. A few ideas about future directions this research could take have also been presented. It is envisaged that by obtaining better quality input, and matching points more accurately the overall result could be improved dramatically.

10 Appendix

10.1 User Manual and Installation Guide

The software written to accompany this thesis is available on the CDROM attached. The latest version can be downloaded from:

`http://www.cse.unsw.edu.au/~saraf/slam/`

or by using the `Darcs` version control system:

```
darcs get http://www.cse.unsw.edu.au/~saraf/slam/
```

10.1.1 Introduction

The source provided will compile into two binaries:

- `read_svs` which given a directory of svs-compatible images, will read them in sequence and run stereo processing on them, saving the result of each frame in a separate binary file, in a directory called `bindata` in the current working directory.

- `slam` which given a directory of binary data generated by `read_svs` will read the frames starting from the first frame saved. Optionally a second argument to `slam` can be given to specify the starting frame number. This program saves a map of the world for each frame processed in a directory called `dump` under the current working directory.

Also supplied is a pyVTK script titled `render.py` which can render the output of the `slam` program using VTK. A monochromic version called `grender.py` is also supplied.

10.1.2 Package Layout

The package root directory contains four directories:

- **slam** — Contains source for the `slam` and `read_svs` programs. A copy of the Newmat library source is also found here.

- **rendering** — Contains the scripts **render.py** and **grender.py**. Used for rendering the output of the `slam` program.

- **povray** — Contains the python script **makestereo.py** as well as a directory containing assorted povray scenes which can be used or modified for rendering.

10.1.3 Pre-Requisite Packages

In order to compile the `slam` program, the following standard libraries are required to be installed on the system: libjpeg, libpng and libtiff. In addition, the `Newmat` matrix library is required. The source for this library is provided in this distribution.

To compile the `read_svs` program, the SVS binary package and headers must be installed on the system. Additionally, the `Vigra` library available from `http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/` must be installed before proceeding.

To run the renderer, the `VTK` must be installed. In particular the `pyvtk` extension must be installed. This option is off by default in the source distribution, but available in the Debian package automatically.

For generating test data from Povray, Povray needs to be installed on the system. The python script which converts a povray animation file into SVS readable data needs the python imaging library (PIL) to be available.

10.1.4 Installation

The following steps executed in order should yield you working binaries for the project, provided the paths are all set correctly, and the pre-requisite packages are all installed.

1. Go to the package root directory.
2. `cd slam/newmat`
3. `make`
4. `cd ..`
5. `make`
6. `cd read_svs`
7. `make`

The python scripts are interpreted, and require no installation.

10.1.5 Usage

The first step is data collection. You can either use the `smallv` application which comes as part of the SVS package to generate data from the camera (use `save frame buffer` to save the camera frames to disc), or you can generate them artificially using `povray`.

To use povray, first write a povray source file. You can also use the examples found in the `povray/src` directory. Now, simply run the `makestereo.py` script to generate a directory filled with data SVS can use. Running this script will bring up a povray rendering window, which you can close with no ill-effect.

Optionally (this is probably a good idea), open up frames in `smallv` and then save one to disk. This saves a *.ini file for the image, which you can copy for all the frames. A one line shell script to do this can be printed by running `makestreo.py` with no arguments.

Next, you want to generate binary data from this sequence of frames. To do this, simply run the `read_svs` program with the path to the first frame in the sequence you want to use. For example:

```
./read_svs /cdrom/run5/final001-C.bmp
```

will process the sequence of images found in the directory `/cdrom/run5`.

A directory called `bindata` must exist in your working directory for this program to work, otherwise it will not write any output files.

Once you have the binary data, you can run it through the SLAM algorithm by simply giving the `slam` program the directory storing the binary data as its first argument. An optional second argument is the starting frame number. `slam` will write output to a directory called `dump` in the current working directory. It will fail if this directory doesn't exist.

After all this data is processed, and you terminate the program, you can view the map it created using the python rendering script. Simply supply the script a slam output file, e.g.:

```
./render.py dump/33world.pts
```

You can move around and inspect the map from different angles by using the mouse in the VTK window. The map can be translated, rotated and scaled by moving the box frame that appears in the window along with the map.

Bibliography

[1] A.J, D. 3d simultaneous localisation and map-building using active vision. In *IEEE Int. Conf. of Computer Vision and Pattern Recognition. vol.1* (2002), pp. 384–391.

[2] Bailey, T. Constrained initialisation for bearing-only slam. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation* (2003).

[3] Besl PJ, M. N. A method for registration of 3-d shapes. In *IEEE Trans Patt Analysis Machine Intelligence 14(2)* (1992), pp. 239–256.

[4] Bob Fisher, Simon Perkins, A. W., and Wolfart, E. Canny edge detector. `http://www.cee.hw.ac.uk/hipr/html/canny.html`.

[5] Cognitive Systems Group, University of Hamburg, G. Vision with generic algorithms (vigra) library. `http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/`.

[6] Cole, D. M., Harrison, A. R., and Newman, P. M. Using naturally salient regions for slam with 3d laser data. In *IEEE International Conference on Robotics and Automation SLAM Workshop* (Barcelona Spain, April 2005).

[7] Davies, R. Newmat website. `www.robertnz.net/`.

[8] Davison, A., Cid, Y. G., and Kita, N. Real-time 3D SLAM with wide-angle vision. In *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon* (July 2004).

[9] Davison, A., Mayol, W., and Murray, D. Real-time localisation and mapping with wearable active vision. In *Proc. IEEE International Symposium on Mixed and Augmented Reality* (Oct. 2003), IEEE Computer Society Press. (To appear).

[10] Dellaert, F., Seitz, S., Thorpe, C., and Thrun, S. Structure from motion without correspondence. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'00)* (June 2000).

[11] Ferguson, D., Morris, A. C., Haehnel, D., Baker, C., Omohundro, Z., Reverte, C., Thayer, S., Whittaker, W. R. L., Whittaker, C., Burgard, W., and Thrun, S. An autonomous robotic system for mapping abandoned mines. In *Advances in Neural Information Processing Systems (NIPS 03)* (2003).

[12] Hähnel, D., Thrun, S., Wegbreit, B., and Burgard, W. Towards lazy data association in SLAM. In *Proc. of the International Symposium on Robotics Research (ISRR)* (2003).

[13] Kim J. H., C. M. J. Slam with omni-directional stereo vision sensor. In *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada* (2003).

[14] Kitware. Vtk website. `http://www.vtk.org/`.

[15] Konolige, K. Small vision systems, hardware and implementation. In *Eighth International Symposium on Robotics Research* (1997).

[16] Liu, Y., and Thrun, S. Results for outdoor-SLAM using sparse extended information filters. Submitted for publication, 2002.

[17] M.A.Garcia, A. "3d simultaneous localization and modeling from stereo vision". In *"IEEE Int. Conf. on Robotics and Automation, New Orleans, USA "* (2004), pp. 847–853.

[18] Mayol, W., Davison, A., Tordoff, B., Molton, N., and Murray, D. Interaction between hand and wearable camera in 2d and 3d environments. In *Proc. British Machine Vision Conference* (Sept. 2004), BMVC. (To appear).

[19] Montemerlo, M. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[20] Mount, D. Ann programming manual.

[21] Newman, P. M., and Ho, K. Slam - loop closing with visually salient features. In *IEEE International Conference on Robotics and Automation SLAM Workshop* (Barcelona Spain, April 2005).

[22] Nieto J, Guivant J, N. E. Fastslam: Real time implementation in outdoor environments. In *ACRA 2002 Auckland, NZ, (in press November 2002).* (2002).

[23] nist.gov. Urban search and rescue (USAR) – rules. `http://robotarenas.nist.gov/rules.htm`.

[24] povray.org. Povray website. `http://www.povray.org/`.

[25] Robocup.org. Robocup rescue competition. `http://www.robocup.org/games/36.html`.

[26] Rotomotion. Rotomotion web page. `http://rotomotion.com/prdREV2.4.6DOFK.html`.

[27] Rottensteiner, D. F. *Semi-automatic extraction of buildings based on hybrid adjustment using 3D surface models and management of building data in a TIS*. PhD thesis, Vienna University of Technology, Faculty of Science and Informatics, Febuary 2001.

[28] Rusinkiewicz, S., and Levoy, M. Efficient variants of the ICP algorithm. In *Proceedings of the Third Intl. Conf. on 3D Digital Imaging and Modeling* (2001), pp. 145–152.

[29] Skubic, M. Qualitative spatial referencing for natural human-robot interfaces. *interactions 12*, 2 (2005), 27–30.

[30] Systems, I. Indigo Systems web page. `http://indigosystems.com/product/micron.html`.

[31] Thrun, S. Learning occupancy grids with forward models. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS'2001)* (Hawaii, 2001).

[32] Thrun, S. Robotic mapping: A survey, 2002.

[33] Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z., and Durrant-Whyte, H. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research* (2004). To Appear.

[34] Videre. Videre Design home page. `http://www.videredesign.com/sthmdcs.pdf`.