Practical Visual Odometry For Small Embedded Systems

A thesis presented

by

Shawn Samuel Schaerer

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in the subject of

Computer Engineering

The University of Manitoba

Winnipeg, Manitoba

August 2006

Thesis advisor

Dr. Jacky Baltes

Author

Shawn Samuel Schaerer

Practical Visual Odometry For Small Embedded Systems

Abstract

Localization and mapping are important abilities for any robot to have if it wants to navigate intelligently in the real world. The goal of the research designed in this thesis was to develop a practical embedded visual odometer that utilized common features found in real world environments. The visual odometer is a system that measures the self motion of a mobile robot using visual feeback.

The developed visual odometer was tested on a custom mobile robot in several different tests that were derived from the robotic soccer domain. This system's performance was compared to two other systems. These systems were a KLT [1] feature tracker based robot and a commercial shaft encoder based robot. The results of the completed tests showed that the developed visual odometer's performance was less than expected. It also showed that this system has good potential. As well, the test results showed the limitations of using a KLT [1] feature tracker based robot and that the commercial shaft encoder based robot also had performance less than expected.

Contents

List of Figures

Acknowledgments

I would like to thank the following people for their help, support and guidance throughout my studies.

- My wife Tracy for her understanding, guidance and patience while I spent most of my time with the robots. I would also like to thank her for her help in editing this thesis.
- My Daughter Amelie for making the last year of my studies more enjoyable.
- Dr. Jacky Baltes for his guidance, support and research opportunities.
- Dr. Bob McLeod for his help with all of my electrical engineering department problems. Especially the help in getting special course approval.
- Dr. John Anderson for the use of his Pioneer robot and thesis guidance.
- Mr. Paul Furgale for developing the offline Tsai calibration program.
- Mr. Mike Gauthier for setting up the Pioneer robot.

Chapter 1

Introduction

Localization and mapping are very important skills for any mobile robot to have when it needs to navigate intelligently in the real world. Localization can be defined as knowing where one is at all times and mapping as taking that knowledge to develop a map of where one has been.

In simple domains such as the one shown in Figure 1.1, a mobile robot needs only to identify which landmark (i.e A1, B3, etc.) it is situated on to localize itself. In this example, a mobile robot is able to sense the landmark in the cell that it currently occupies. It is easy for the robot to navigate as it traverses the domain because it always knows in which cell it is situated. This means that the robot is globally localized at all times. These simple domains are rarely encountered because they require the addition of many unique landmarks.

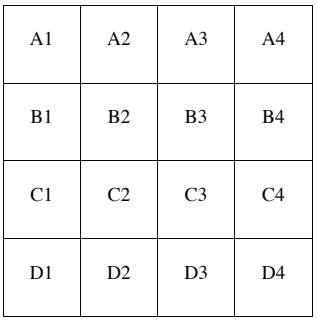


Figure 1.1: An example of a simple domain where the robot needs only to know which landmark it is on to localize itself.

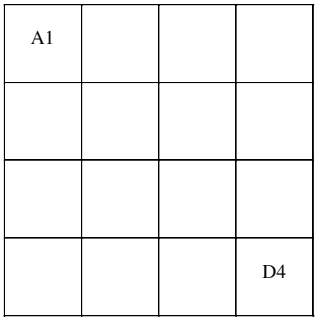


Figure 1.2: An example of a domain where the robot requires more information to localize itself in all areas.

A more common example is shown in Figure 1.2. In this example there are only two landmarks (A1 and D4) that the robot can sense. This means that a mobile robot can be globally localized in cells A1 and D4, but not in any other cell. In this situation as the mobile robot traverses the domain it needs to keep track of its position.

Figure 1.3: An example of a complex domain that contains few or no unique landmarks, which makes it very difficult or impossible for a robot to localize itself.

In real world examples such as the Urban Search and Rescue domain shown in Figure 1.3, a mobile robot requires additional information to localize itself in the world. In order for a robot to localize itself in complex domains, it must be able to measure its ego-motion. Ego-motion is the ability for a robot to determine its self-motion via feedback from its external sensors.

Most mobile robots utilize dead reckoning to measure their ego-motion. Dead reckoning relies on mechanical sensors (shaft encoders) to provide motion feedback after a set of motion commands are issued. The feedback is then directly used to update the position and orientation of the robot. This can be problematic as these feedback sensors can introduce incremental errors due to wheel slip or the robot not being able to move.

In most research situations this is not that much of an issue. Today, the majority of mobile robots are deployed in labs or ideal condition environments. As new technologies are developed and the cost of computation and sensing decreases, the deployment of autonomous mobile robots will increase to the point where they become a part of everyday life. When this occurs, autonomous robots will need to interact intelligently in dynamic situations such as autonomous driving on major thoroughfares, yard maintenance and personal care. In these real life situations, incremental errors due to the mechanical feedback system can lead to serious problems such as high speed collisions, damage to property and loss of human life.

One way to overcome the problem of determining the ego-motion of a mobile robot is to use visual feedback. Visual feedback does not suffer from the same problems as its mechanical counter-parts. It is a passive system that does not need to actively engage the environment or the robot to get feedback. The use of visual feedback in ego-motion estimation is called visual odometry or "the problem of determining the motion of a robot from a given sequence of images taken during motion" [2].

In this thesis I have developed an embedded visual odometer that utilizes common features found in real world environments which allows mobile robots to navigate in complex domains. The visual odometer that was developed extracts strong lines that are found in acquired images. It then tracks the movement of these lines over a sequence of images. This line movement information is used to calculate the change in position and orientation of each tracked line over time. This change in line position and orientation over time is then used to calculate the ego-motion of a mobile robot.

This visual odometer was tested in two different test beds. The first was a linear traversal of the Robocup E-league [3] soccer field and the second was a figure eight traversal of the same soccer field. The results obtained from the test runs were compared to the results from running the same tests on two separate systems. These systems were a commercial shaft encoder based robot and a Kanade, Lucas and Tomasi (KLT) [1] feature tracker based robot. The preliminary results obtained found that the developed visual odometer's performance was less than expected and that this system has good potential but needs further development. The results also showed the limitations of a KLT [1] feature tracker and that the commercial shaft encoder based robot does not perform well during turning.

This thesis is broken down as follows: Chapter 1 Sections 1.1 and 1.2 describe the problem and motivation for this research, Chapter 2 discusses the background for this thesis and some of the current research in visual odometry, Chapter 3 describes the visual odometry system that was designed, Chapter 4 describes the designed system's hardware and software, Chapter 5 outlines the methods used to evaluate this research and discusses the results obtained from running the system on the mobile platform described in Chapter 4 and Chapter 6 concludes this thesis.

1.1 Problem

Navigation is a major problem for all mobile robots. Without navigation, robots would be unable to move intelligently in the world. For example, humans use visual information to navigate automobiles at high speeds on major freeways. This visual information helps the driver decide how to avoid an obstacle or which way to turn the steering wheel when the driver wants to pass a car. If the visual feedback was occluded by blindfolding the driver, they would not be able to make intelligent decisions on how to react in crucial situations such as collision avoidance.

Navigation for mobile robots consists of three parts: mapping, localization and ego-motion estimation. Figure 1.4 shows the hierarchy and the importance of these components. One can see from Figure 1.4 that ego-motion estimation is the foundation of intelligent navigation. Without ego-motion estimation, a robot cannot localize itself in complex domains.

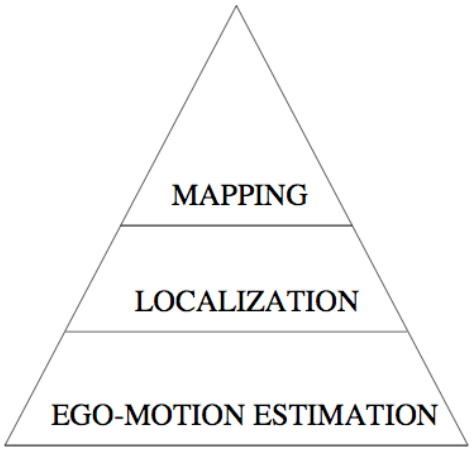


Figure 1.4: Navigation triangle and its importance structure.

Most systems take ego-motion estimation for granted and assume that they yield highly accurate results. This assumption allows researchers the ability to develop complex localization algorithms without worrying about the underlying problems associated with ego-motion estimation. This is not a good assumption to use if one is developing a simultaneous localization and mapping system (SLAM) to be used in real world environments. Perfect ego-motion estimation does not exist and designers cannot take into account all of the different terrains that a mobile robot will encounter during its traversal of an environment. For example, if the mobile robot is operating on a busy factory floor, designers cannot guarantee that the environment will be free of debris and obstacles.

In summary, in order to perform intelligent navigation, ego-motion estimation must exist. The focus of this research was to develop an accurate visual odometer that would allow a mobile robot operating in complex environments to accurately determine its ego-motion without other external feedback.

1.2 Motivation

The motivation for this research comes from many different areas. The first stems from the rules of local vision RoboCup [3] soccer. If one were to review the previous years rules for the RoboCup [3] legged and mid-size leagues, they would notice a trend that incrementally removes artificial field markers. This means that an autonomous robot should not depend on these landmarks as they could be eliminated in the next year's competition. The obvious solution for this situation would be to use features that naturally occur in the environment, such as field lines and goal areas.

Another area that provides motivation is the area of Urban Search and Rescue (USAR) [4]. In this domain, robots navigate themselves in unstructured environments where there can be few assumptions made. The goal of USAR research is to have fully autonomous robots that can go into a disaster situation and provide a map of the area and casualties to rescue workers. For this type of system to be successful, it requires an accurate SLAM system that uses accurate ego-motion estimation.

Another area of inspiration is the Darpa Grand Challenge [5] where fully autonomous robots navigate through an unstructured desert environment trying to complete a trek from Barstow California to Las Vegas.

Chapter 2

Background and Related work

This chapter details the background for this thesis and summarizes some of the work in visual odometry research.

2.1 Background

The background for this thesis comes from the research that was conducted by myself and Dr Jacky Baltes [2] at the University of Manitoba. The main considerations and assumptions used as the starting point for the research were that natural features exist in most scenes and that these features could be exploited and used as visual feedback in a visual feature tracking system. The features that were of most interest to us were lines and line segments that occurred naturally in our everyday environment. Image points also provide visual feedback but are susceptible to noise which can lead to erroneous results and were therefore not considered.

This is not the first time that lines have been used in estimating motion. In 1995 Zhang [6] described a methodology for estimating motion and structure using line segments in an image and Taylor et al. also stated that "straight line features are prominent in most man-made environments [and] they can be detected and tracked relatively easily in image data" [7]. This is different from the research conducted in this thesis. It differs in that, the real world positions of extracted image lines are tracked over time to measure the ego-motion of a mobile robot. Zhang [6] and Taylor [7] both try to reconstruct 3D scenes from the motion of a stereo camera system and do not measure the ego-motion of a mobile robot.

The major source of environmental lines comes from walls, floors and the junction between these two elements. Other sources of lines can come from patterns that exist in a scene such as a tiled floor or a group of trees. A visual example of this can be seen in Figures 2.1 and 2.2. Figure 2.1 shows two examples of scenes where lines can be exploited. Figure 2.2 shows the same scenes with the lines enhanced via edge detection.

Figure 2.1: This is an example of two naturally occurring scenes. One can see that there are a lot of visual features that yield lines in the image.

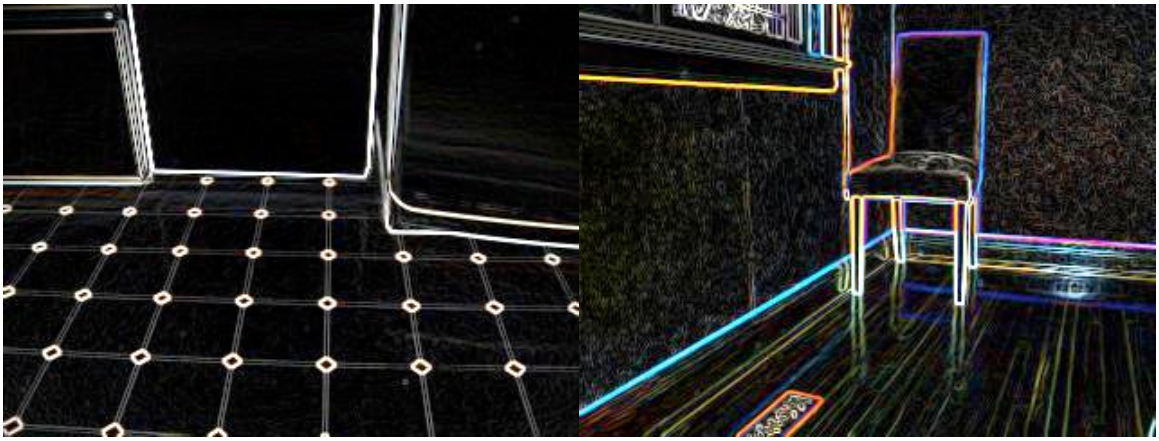


Figure 2.2: Here are the same scenes as seen in Figure 2.1 with the lines enhanced via edge detection.

Ego-motion can be calculated by tracking environmental lines and measuring their change in position and orientation over time. This translates into robot displacement S and change in orientation $\Delta\theta$. This allows the visual odometer developed in this thesis to measure the motion of a mobile robot using only visual feedback. Schaerer et al. [2] calculated ego-motion differently. In their work, it was calculated in terms

of a robot's right and left wheel velocities.

The visual odometer can be thought of as a passive motion feedback sensor that does not suffer from the same problems as mechanical sensors (shaft encoders). The developed sensor provides sensor measurements that can be utilized by higher level localization systems.

The focus of the research conducted in this thesis was to develop a visual odometer (sensor) that could determine a mobile robot's ego-motion. This is different from the work of Thurn and Fox [8] and other SLAM researchers. Thurn and Fox [8] have researched Monte Carlo based localization (MCL). MCL (also referred to as particle filters) is a probabilistic algorithm that tries to determine a robot's current pose based on its sensor readings. These sensor readings can include laser range finders, mechanical odometers and could include the visual odometer developed in this thesis.

2.2 Related work

Visual odometry (which is also referred to as visual ego-motion estimation) research has been investigated by a number of different groups. This work can be divided into two different categories; one that assumes that there is structure in the image data and that it can be used as a means of detecting image motion and one that does not assume that visual data contains any structure that can be used to calculate image motion.

The following sections investigate and summarize the research that has and is being conducted by various research groups around the world.

2.2.1 Category 1: Assumed Structure

In this section, all research conducted by the various groups has used one major assumption. This assumption is that in all the acquired image data, structure exists and can be exploited to determine the ego-motion of a camera. Examples of structure are field lines and goal areas that exist in the RoboCup [3] soccer domain or 3D landmarks such as tables and chairs that exist in an office environment.

Amidi [9] developed the first autonomous helicopter with an on board visual odometer. This visual odometer was responsible to estimate the helicopter's 3D position and velocity with a high frame rate and low latency. The visual odometer was an on-board custom hardware solution comprised of a stereo vision rig (two high quality NTSC cameras), two A/D converters (which provided low latency image acquisition) and four separate DSP's responsible for all of the image processing and visual odometry calculations.

The visual odometer performed two functions in this research. The first was to calculate the helicopter's 3D position using the stereo pair and the second was to estimate the velocity of the helicopter. Initially, the helicopter knew its 3D position. In order to perform the 3D position estimate, the visual odometer detected objects at the centre of the image via the stereo pair. It then tracked these objects as they moved throughout the image. For each new image captured, the visual odometer tried to locate the objects that were currently being tracked. If the tracked objects were not found in the image then new objects were selected to be tracked. Object tracking was accomplished by using high speed template matching of the currently tracked objects and the new objects found in the image. If a match was found, the

visual odometer would calculate the 3D position of the helicopter. The 3D position was estimated by using the depth calculated by the stereo pair, the previous 3D position, the onboard angular sensors and the estimate of the change in lateral and longitudinal displacement calculated by the high speed template matching.

The second function of the visual odometer was to estimate helicopter velocity. Velocity was estimated by calculating the optical flow between two successively captured images. The optical flow was calculated at the centre of the images and was estimated by taking the derivative with respect to time of the tracked objects change in X and Y displacement.

Amidi's design of the visual odometer allowed the stable control and flight of the helicopter. The performance of the visual odometer running on the on-board custom vision hardware enabled the system to process images at 60 frames per second. This high throughput of the visual odometer allowed the helicopter to control its 3D position within 0.5 meters of the desired 3D position.

Sim and Dudek [10] estimate the position (ego-motion) of a robot by visually detecting landmarks from images. In this work, they tried to eliminate the use of artificially placed landmarks by utilizing only visually detected landmarks. The only assumptions that they made are that the general starting position of the robot was known and that the camera's orientation was always the same.

This developed system consisted of two stages, an off-line stage and an online stage. The off-line stage was used to build a database of detected landmarks which allowed the robot to estimate its position during the online stage. During the off-line stage, the robot manually moved throughout a test environment in a grid while it

took images and detected visual landmarks. Landmarks were grouped into sets. Each set contained the same landmark but at different positions and orientations in the test environment. This grouping allowed the robot to detect the landmarks in a large environment. After the landmarks had been detected and grouped into sets, they were placed in a database so that they could be used during the online stage.

The online stage was used when the robot was traversing the test environment. It allowed the robot to estimate its position after every movement. To estimate position, the robot first acquired images as it moved throughout its environment (the environment was partitioned in a grid, the same as in the off-line stage). Next, the images were processed and visual landmarks were detected and extracted. After the landmarks had been detected, the off-line database was searched and an attempt was made to match the newly detected landmarks to the landmarks that were stored in the database. If landmarks were found in the database, the robot's position was estimated by comparing the positions of the detected landmark to the positions of the landmarks in the database. To remove outliers from the estimates, the estimates were run through a median filter. The final position estimate was taken as the mean position value.

This work was tested in two very small test environments. The first was a grid of 30x30 cm with 2 cm spacing and the second was a grid of 1.2x3.0 metres with a 20 cm spacing. Sim and Dudek [10] found that the robot had good position estimation in both tests. In the first test the average position error was 3.8 mm and the second had an average position error of 5 cm.

Although this work produced very good results, it is unknown how it would scale in large environments. It is also unknown what would happen if the camera had orientation changes. Another issue with this system is the need for off-line population of the database. This system would not work in situations where the robot had to traverse unknown environments and it would not work if the robot was picked up and moved to a different location. Finally, it is not mentioned which type of system is needed to implement their design. It is assumed that this system would not be viable in small scale embedded systems.

The problem of having the need for an off-line database population is addressed by Se, Lowe and Little [11] [12]. They developed a stereo vision based SLAM algorithm that detected and tracked visual scale invariant feature transform (SIFT) landmarks in an arbitrary environment. They used these 3D SIFT features (which are scale and orientation invariant) to estimate the ego-motion of a shaft encoder based robot. The goal of this research was to improve the error of the shaft encoders by augmenting its position estimate with SIFT based visual odometry.

Ego-motion was estimated by first extracting SIFT features in the images and then by matching the SIFT features in the stereo pair. This provided the 3D positions of each SIFT feature. After all the features were matched, they were stored in an online feature database so that they could be tracked over time. The use of an online database allowed the robot to explore unknown environments because it did not need to populate the database off-line (by first traversing the environment).

The next step used the odometry data from the shaft encoders and a new image pair to locate the SIFT landmarks. The SIFT landmarks were located (tracked) in

the new images via a search that was concentrated in areas that the features should be in, given the data from the odometer. After the landmarks were found and matched between frames, a rough estimate of the ego-motion of the robot was produced. In order to refine the ego-motion estimate, a least squares minimization algorithm was used on the estimated ego-motion. This minimized the calculated ego-motion's error and produced the final ego-motion of the robot.

Overall, this method yielded good ego-motion estimation and localization results that allowed a mobile robot to generate a 3D map and localize itself in an office (lab) environment. This implementation would not be feasible to run on a small embedded system given that it can only run at two updates per minute on a desktop computer.

Sim et al. [13] developed a SLAM system that utilized a visual odometeric motion model. This system differs in that most common approaches for odometeric measurements use shaft encoders or laser scanners to determine a robot's ego-motion. Sim's group utilized visual information to allow the development of a visual odometry system to update the robot's motion model. This visual odometry system relied on a stereo vision system and past research in multiple view geometry to determine a robot's ego-motion. Determining the robot's ego-motion was accomplished by identifying and classifying landmarks in consecutive image pairs using SIFT descriptors.

Herrero-Perez, Martinez-Barbera and Sffiotti [14] developed an embedded localization system that ran on the Sony Aibo robotic dog. This localization algorithm detected corners and goal areas to allow the Aibo to localize itself in its environment. This implementation was successfully used in competetion at the 2004 RoboCup [3]. Although their approach does not calculate the ego-motion of the robotic dog di-

rectly, it does utilize the natural features found in the environment (field lines, goals) to estimate the position of the robot.

2.2.2 Category 2: No Assumed Structure

The research that was carried out in this section does not assume that the visual data contains any structure that can be used to calculate ego-motion. This means that there are no assumptions about the scene and what the world looks like.

Milella and Siegwart [15] developed a realtime six degree of freedom visual odometer using a stereo vision based feature tracker and an iterative closest point image registration algorithm (ICP). In their research they looked at applying stereo vision and ICP to develop a visual odometer. In order to accomplish this they assumed that images contain "visual distinct features [that] can be tracked [over time]" [15]. This work is similar to that of Cheng et al. [16] but it differs in the way the motion is estimated.

Their approach to visual odometry was as follows. First, 3D features are generated using an "SRI Stereo Engine algorithm" [15]. This produces 3D points that have good matches between the stereo pair. Next, features are detected using a Shi-Tomasi [17] feature detector. After the features have been detected, they are tracked over time. Finally, ego-motion of the robot is estimated by comparing all current and previous positions of the 3D points. This is done by finding a "3D transformation matrix T that minimizes " [15] the squared error between all of the 3D point's current and previous positions. After the motion has been estimated, ICP is applied to this estimate to eliminate outliers in the motion calculation.

This visual odometer was tested using two different test beds. The first was a flat surface and the second a rocky environment. In the first test, the robot traversed a path that was 1.78 metres by 2.2 metres. After the robot had completed its path, its absolute position error was recorded and found to be only 1.9 millimetres in the X direction and 2.4 millimetres in the Y direction. In the second test, the robot traversed the environment 2.2 metres. After it had finished its traversal, it ended up with a position error of 2.4 millimetres.

The results of this system are very impressive but it is noted that the computational system that they used in this system was a 2.4 Ghz Pentium 4 class machine. It is not known whether their developed visual odometer would be able to run on a small scale low power embedded system.

Campbell et al. [18] [19] [20] developed a visual odometry system that utilized commodity hardware and software. In their research, they developed a visual egomotion system using openCV [21] (open computer vision library) and a web camera. They utilized the Lucas-Kanade optical flow algorithms [1] available in openCV to determine the optical flow of an image sequence. With this information, they were able to accurately determine the position and orientation of the robot as it moved throughout its environment.

This system allowed their demonstrated mobile robot to correct for any deviation from its control path while avoiding any sudden drop offs like the edge of a table. Although their demonstrated system performed well, the overall processing was still being performed on a PC and not on an embedded system.

McCarthy and Barnes [22] compared the effects of temporal filtering on a gradi-

ent based optical flow algorithm which was used in mobile robot corridor centring and visual odometry experiments. Their work looked at the effects of applying different temporal filters to the incoming single camera images to discover which filter performed best in mobile robot navigation.

The application of these filters was to remove image noise before the optical flow step. Gradient based optical flow algorithms are highly sensitive to noise and therefore removing image noise would improve the accuracy of these algorithms. Three different filters were used in this work, a Gaussian filter, a Simoncelli matched pair filter and a recursive temporal filter. To compare performance, each filter was used in a different visual odometry experiment and corridor centring experiment. The visual odometry used in the experiments was based on the Lucas-Kanade [1] optical flow algorithm. After testing each filter in mobile robot corridor centring and visual odometry experiments, they found that the recursive filter performed the best and improved the results.

Nister et al. [23] developed a real-time visual odometer that can obtain motion from a mono or stereo vision system. This system made no assumptions on the type of motion or scene data that is acquired from a sequence of images. The visual odometer algorithm that was implemented was first applied to a mono vision system and then to a stereo vision system. The algorithm was developed in such a way that it could be applied to either vision system with little change.

The algorithm consisted of three parts; a feature detector, a feature tracker and a motion estimation phase. The algorithm differs in the motion estimation phase depending on what type of vision system is available.

The feature detector extracts corners from each captured image. This feature detector can detect up to a maximum of 5000 features per image. This algorithm is very computationally expensive because the detector has to search the entire image space for the features. In order to speed up this step, Nister et al. took advantage of the MMX hardware acceleration available on Intel microprocessors. After all of the features have been extracted from the image, a feature tracker tracks the extracted features between frames. For a feature to be successfully tracked from one image to the next, a matching criteria was introduced. This criteria states that for a feature to be successfully matched, the disparity of the feature between successive images must be within a predefined limit. If the features are not within that limit they are not matched.

Once all of the features have been detected and successfully tracked, the motion estimation phase is executed. In the case of a mono vision system, motion is estimated by first tracking the detected features over a number of separate images. The motion estimation phase then calculates a feature's pose for each tracked feature using a 5-point pose algorithm that was previously developed by Nister [24]. Next, the 3D position of each detected feature is calculated using the first and last image of the acquired images. After the 3D position of every feature has been calculated, the 3D pose of the camera is estimated using the 3D point information. The output of the motion estimation phase is the relative motion of the camera (robot). In the case of a stereo vision system, motion estimation is carried out in the same way as the mono system except that the 3D position of each detected feature is calculated using stereo matching of the features between the two images obtained by each of the cameras.

This research was successfully tested on a mobile ground robot using a Pentium III 1GHz machine and the test results compared against the results obtained using a differential GPS system. The test performed for the comparison was for the mobile robot to complete a 20 metre diameter circle three times and travel a total of 184 metres. After the test was complete, the mobile robot that had used the developed visual odometer ended up 4.1 metres away from its starting point versus the DGPS system which ended up exactly where it had started.

Although the performance of the system is very good it still requires large computational power to run and like most of the research discussed in this chapter, it would not be well suited for a small scale embedded system.

Ali et al. [25] used a visual odometry algorithm to aid position estimation during tele-autonomous operation of the Sprit and Opportunity Mars Rovers. The current Mars Rovers update their positions when commanded by the operators. To do this, the Rovers use a number of different sensor combinations. The sensors that are available for normal position estimation are gyroscopes, accelerometers and shaft encoders. The combination that is used depends upon the movement of the Rovers. The visual odometry system is used to refine the position estimation or replace it in environments with high wheel slip. The Rovers require accurate position control in order to carry out successful missions. To do this, they require "that the position error accrued during a drive must be no greater than 10% of the traveled distance, up to a distance of 100 meters" [25]. This means that using shaft encoders alone in high slip environments is not an option.

To alleviate the problems of the mechanical shaft encoders and meet the require-

ments for position control Ali et al. [25] utilized the visual odometry system developed to Cheng et al. [16]. This visual odometer calculates the Rover's 6-DOF pose which is stated to be "X, Y, Z, Roll, Pitch, Yaw" [16] and does not assume any structure in the image.

The visual odometer uses a stereo vision system to calculate the ego-motion of the Rovers by detecting features and tracking these features over time. Ego-motion is calculated by first detecting features in both images using a feature detector. The features that are detected in this system are corners similar to Nister et al. [23]. The detected corners are extracted and the ones that can "easily be matched" [16] between both cameras are kept. Next, the extracted corners are matched using stereo matching. After all of the corners have been matched or discarded their 3D pose is estimated using the data from the stereo matching.

Once the corners have been matched and their 3D position estimated, they are tracked over time. After the Rover has moved, the features are tracked by looking for these corners in a newly acquired image pair. This is done by using the estimate of wheel movement provided by the shaft encoders and a "correlation-based search [algorithm]" [16]. When the corners have been successfully tracked, their new 3D position is calculated using stereo matching.

The motion of the Rover is estimated by comparing all of the tracked features current 3D positions to their previous 3D positions. If the difference in the 3D positions are within a tolerable range, then the motion is considered to be matched to the motion estimated by the shaft encoders. If the difference between the positions of all of the features are not within the error tolerance, then the motion estimation from

the shaft encoders is considered invalid and the Rover's new position is estimated using a motion estimation algorithm.

The motion estimation algorithm estimates motion using the 3D position data of the features by first performing a "least squares estimation" [16] on the data. This produces a coarse motion estimation which is refined using a maximum likelyhood estimation to produce the final motion estimation.

The visual odometery system developed by Cheng et al. [16] integrated into the Mars Rovers as described by Ali et al. [25] is still in service today on both the Sprit and Opportunity Rovers. The Rovers were able to travel an average distance of over 4.75 kilometres by "using a combination of wheel-odometry, gyro readings, and visual odometry" [25].

Cheng et al. [16] discovered that their visual odometer performed as well as shaft encoders in "simple terrain" [16]. It performed much better than the shaft encoders in high slip / slope environments. They also found that it did not perform well if the Rover motion was too large. When the motion was too large, features were lost and could not be tracked. The visual odometer was not used at all times because their method was not suited to run in real-time on the the simple embedded platform used on the Rovers. The use of visual odometry on the Mars Rovers has led to more "scientific observations" [16] being made. It has also allowed for missions to be stopped and re-planned if no progress to the target position has been found by the visual odometer.

The visual odometer developed by Cheng et al. [16] was also used by Helmick et al. [26] who developed an autonomous path follower for the Mars Rovers. This path

follower allowed for the Mars Rovers to be fully autonomous.

This system used sensor fusion in the form of a Kalman filter to fuse the data from the gyroscopes, accelerometers and the visual odometer in order to estimate the Rover's motion and position. The system was implemented on the same Rover platform used on Mars and was tested in the Jet Propulsion Laboratory Mars yard. Two different tests were performed. One was on normal desert terrain and the other on "sandy slopes" [26] and each test lasted an average of 50 metres. The results obtained from their tests showed that the visual odometer system was able to perform position estimation with an accuracy of 2.5%. This is better than the shaft encoders which have at "best an accuracy of 10 percent" [26] on normal terrain. The results also showed that the visual odometer system is far superior than the shaft encoder system in high slip environments.

Iida [27] looked at using optical flow in the form of a visual odometer for "long distance goal-directed navigation" [27]. In this research, he was trying to understand how bees use vision for the purpose of navigation. Iida states that "behavior studies with honeybees have recently uncovered a mechanism behind visual mediated odometry, in which the primary cue is the integral, over time, of the image motion that is experience en route" [27]. The visual odometer tries to estimate rotational motion of a flying blimp robot.

The visual odometer uses a single panoramic camera as input and two arrays of the "Reichardt model of elementary motion detection" [27]. The Reichardt model of elementary motion detection uses a combination of spatio-temporal filters on the image data to produce a response to image motion [27]. The two arrays in the

visual odometer are horizontal left and horizontal right motion detectors. Each array provides an output which represents a signal that is proportional to the angular velocity of the robot. The right array provides a response to the angular velocity in the right direction and the left array provides a response to the angular velocity in the left direction. To estimate the motion of the robot, the outputs of each array are combined together. To estimate the position of the robot the motion estimate is integrated over time.

Iida tested his visual odometer in an unstructured environment and compared the position results obtained by the visual odometer and the position results obtained from observing the robot using ground mounted stereo cameras. His results showed that his visual odometer performed well in environments full of structure. The results also revealed that the performance of this system degrades in environments with less structure.

This approach to visual odometry is interesting in that most other systems use some sort of featuring tracking to measure ego-motion. It would be a worth while study to see how this system performs as compared to the more complex feature tracking systems.

Wilson et al. [28] are working on a visual odometry based system to accurately track how far an endoscope travels inside of a patient. This research is still in the initial phase of design. As a starting point, they have focused their efforts on developing the system to measure the ego-motion of a bronchoscope in order to accurately measure its position at all times. When this system is completed, it will allow doctors to efficiently use endoscopes during diagnostic tests and provide an estimate of the

size and structure of the tested area.

The visual odometer that they are developing in this research is similar to those that have already been summarized. The visual odometry algorithm operates by first capturing images. The features in the images are then detected and tracked over time using an implementation of the Lucas-Kanade feature tracker [1]. After the features have successfully been tracked, their 3D positions are calculated using their 2D position information and the intrinsic parameters of the camera. Once all of the feature's 3D positions are known, ego-motion is calculated using the feature's current position, previous position and a five point pose algorithm described by Nister [24].

Although this work is still in the initial phase, Wilson et al. [28] have successfully implemented this system and have tested it off-line using images captured during live bronchoscope procedures. No test results were provided in this research and it is not known which type of system is required to run this algorithm in real-time while the procedure is being performed live in an operating theatre.

This example of using a visual odometer for measuring the position of a scope during a medical procedure solidifies the fact that visual odometry is not only limited to just the field of robotics. It shows that visual odometery can be deployed in fields that can aid the general public.

2.3 Final Thoughts

This chapter has provided the background for this thesis and investigated and summarized the current and previous research in visual odometry. From the information gathered above, a few observations can be made.

The first observation is that most of the research cannot be implemented on a small scale embedded system. The computational requirements are too great. At a minimum, most of the implemented systems required at least a Pentium 3 class system to operate.

Amedi [9] has shown that this is not always the case because his system was implemented as an onboard solution. The downside to this kind of approach is that it requires the use of an expensive custom hardware solution. For the majority of research and commercial applications, this is too cost prohibitive. Cheng et al. [16] have also proven that this observation is not correct by having their system operating on the Mars Rovers which is implemented on a small scale embedded system. It is noted however that this system required a minimum of two minutes to update. This is practical on Mars where the robot does not need to move quickly and where there is no chance of encountering moving obstacles. This would not work in environments such as high speed driving where control feedback needs to be as quick as possible in order to avoid collisions with other moving vehicles.

The second observation that is that most of the systems follow a similar approach to calculate ego-motion. This approach is that image features are tracked over time using a feature tracker such as KLT [1] and the ego-motion is calculated by measuring the displacement of the features over time.

Chapter 3

Visual Odometry System

This chapter describes the complete visual odometry system that has been developed over the course of this thesis. Figure 3.1 shows a high level diagram of the designed system.

The visual odometry system consists of a visual odometry algorithm, a motion

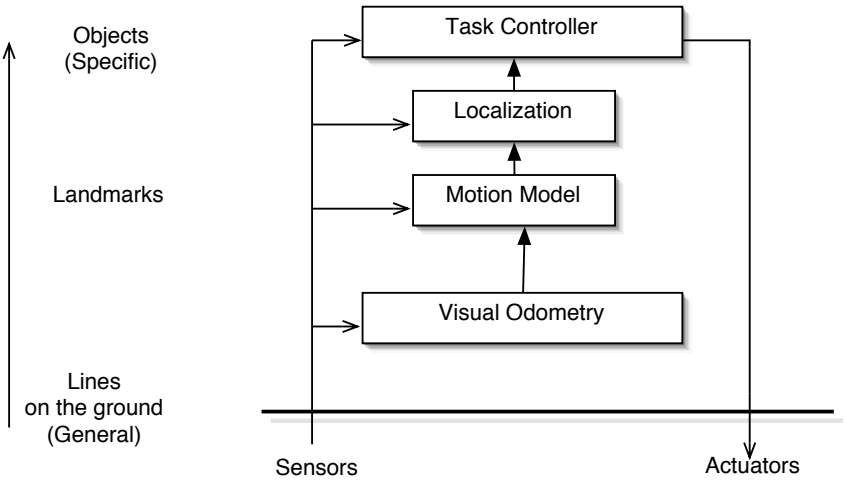


Figure 3.1: System Design of the Visual Odometry System.

model and a task controller. The motion model describes the movement of the robot as it is commanded to move throughout the world. The visual odometry algorithm is responsible for processing the visual feedback and calculation of the ego-motion of the robot. The task controller is responsible for navigation, control and position estimation of the robot. Each part of the visual odometry system will be explained in more detail in the following subsections.

3.1 Motion model

The motion model is a representation of how the robot behaves when a motion command is issued. The motion model is broken down into four separate parts:

- Move forward model. Describes the robot's forward movement
- Move reverse model. Describes the robot's reverse movement
- Turn left model. Describes the robot's left turns
- Turn right model. Describes the robot's right turns

Each model represents a different movement of the mobile robot. Each model in turn tries to accurately describe how the mobile robot moves throughout its world and is modeled by using an average of the last ten successfully measured movements. These models are updated only when visual feedback is available. For example, if the robot is commanded to move forward by 5-cm and it actually moves 7-cm forward with a positive angle of 0.2 radians, it needs a method of keeping track of its actual movement. If this does not occur, the incremental error would be greater

than expected when motion feedback is unavailable. In the absence of this motion model there would not be a good way to predict how the robot moves if no feedback is available. One could use the statically defined motion command distances and rotations, but this would lead to greater error as the mobile robot progressed over time.

The motion model also allows for dynamic motion estimation when the robot changes from one surface to another. This is accomplished by updating the models when visual feedback is available. For example, if a mobile robot's drive system is tuned and modelled to run on smooth surfaces such as an office floor, its physical motion will change if it moves from the smooth surface to a plush surface (i.e. carpet). The motion model will change its internal belief of how the robot moves on this new surface as new feedback information is available and therefore provide a better estimation of the robot's actual motion when feedback is unavailable.

In summary, the motion model allows the robot to predict the amount of motion that was supposed to take place after every motion command has been executed and is primarily used when motion feedback is unavailable.

3.2 Visual odometry algorithm

The visual odometry algorithm is a complex system. It is responsible for determining the ego-motion of a mobile robot. The system receives input in the form of a sequence of images which it uses to calculate the change in robot position and orientation. The algorithm consists of image processing, line detection, line tracking and distance / orientation calculation steps. Figure 3.2 shows a high level overview of the complete algorithm. Details of each component are provided in the subsections that follow.

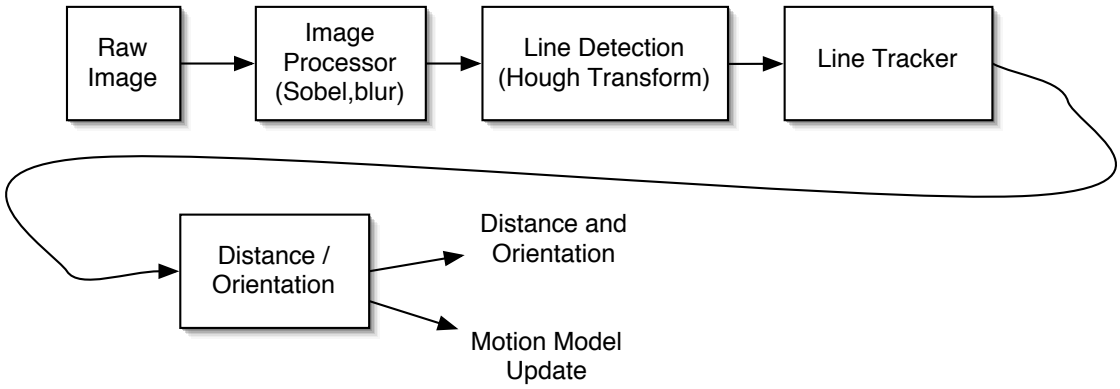


Figure 3.2: Graphical realization of the visual odometry algorithm.

Normal operation of the algorithm begins when the image processor receives a raw image. The image processor performs an image blur, Sobel edge detection [29] and a binary threshold on that image. The resulting data is then passed on to the line detection algorithm. This algorithm attempts to find and extract all strong feature lines that exist in a processed image. If lines have been found and extracted from the processed image, their image position and orientation information are passed on to the line tracking step.

The line tracker receives the extracted line data and tries to track these extracted lines in the sequence of images. It accomplishes this by first creating a list of the extracted lines and then comparing this new list to a list of all previous found lines that were being successfully tracked. The resulting data provides the distance / orientation step with information on how each tracked line has changed between each image in the image sequence.

Ego-motion of the mobile robot is calculated in the displacement / orientation calculation step. This step takes the change in line position and orientation of each tracked line and computes the real world change in robot displacement and orientation. It does this by first converting each line's position and orientation to real world values. It then calculates the robot's actual change in position and orientation by computing the difference between a tracked line's current position and orientation and its previous position and orientation. Once each tracked line's displacement and delta orientation has been calculated, the robot's ego-motion is measured by taking the median displacement and delta orientation of all tracked lines. If the visual odometry algorithm is not tracking lines, then the displacement / orientation calculation step calculates the robot's ego-motion by using the position and orientation information from the motion model.

A visual example of using extracted lines to calculate robot ego-motion is shown in Figure 3.3.

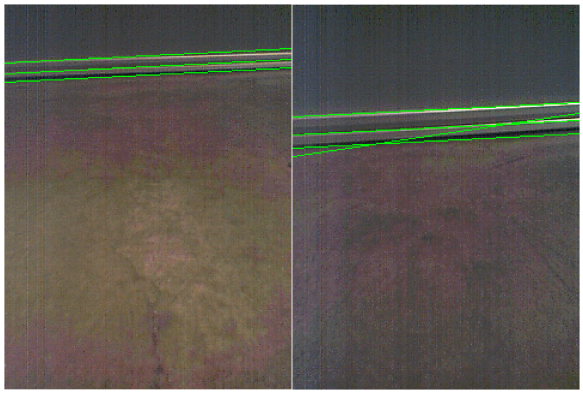


Figure 3.3: This is an example of a two image input sequence that is applied to the system. The system calculates the change in robot orientation and position by comparing the change in line position between the two images. The output from the system will be delta position and delta orientation.

3.2.1 Image processing

Each new image from the visual sensor must be processed in order to obtain the strong lines that exist in the scene. The image processor is a pipeline that receives a new image and then processes that image. Each image is converted from colour to grey scale, blurred and finally convoluted with a Sobel convolution mask to detect edges in both the X and Y directions. Figure 3.4 shows a graphical representation of the complete image processing pipeline.

Grey scale conversion

To begin with, images must be converted to grey scale because the additional information that is available from the colour information will not aid in the detection

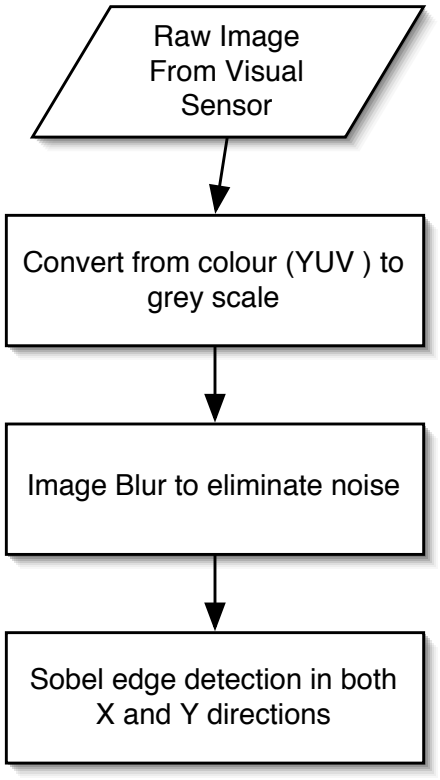


Figure 3.4: Graphical realization of the image processing pipeline.

of strong lines. Depending on the image format, there are a number of ways to convert from colour to grey scale. In the design for this thesis, the image format of the camera is YUV. For YUV image formats, there is no conversion necessary as the YUV format's main channel Y represents the brightness of the image. This brightness channel Y is used as the grey scale component of the image.

Image blur

In order to eliminate some of the high frequency noise that is present in the acquired image, a low pass filter needs to be applied. For this thesis a blurring filter is used. Blurring is also known as image smoothing since it removes some of the

image's high frequency components. An image's high frequency components contain the image's detail and the image's noise. The blurring filter is applied as a convolution operation on the image. The convolution mask used in the blur filter is a three by three neighbourhood mask that takes the average of all the pixels located in that neighbourhood. The result of applying the blur filter to the captured images is that the images look smoother and most of the image noise is eliminated.

Edge detection

After blurring is complete, the image is run though an edge detection phase. In a given image, an edge represents a large spatial rate of change in a group of pixel intensities. This large rate of change can be discovered by calculating the gradient of an image. By calculating the gradient of the image in the X direction, all vertical edges can be found. Calculating the gradient in the Y direction finds all horizontal edges. Both horizontal and vertical edges can be found at once by calculating the gradient of the image in both directions at the same time. For this thesis, a Sobel edge detector [29] is used to find the gradient of the image in both the X and Y directions. The Sobel edge detector [29] performs a convolution operation on the image in both directions. The masks that are used for this convolution operation can be seen in Figure 3.5. After the edge detection phase is complete, the resulting image is then thresholded to create a binary image. This binary image contains all found edges with black (zero) representing the background and white (255) representing the edges in the image. Figure 3.6 shows a raw image and the resultant image after image processing.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 3.5: Sobel convolution mask.

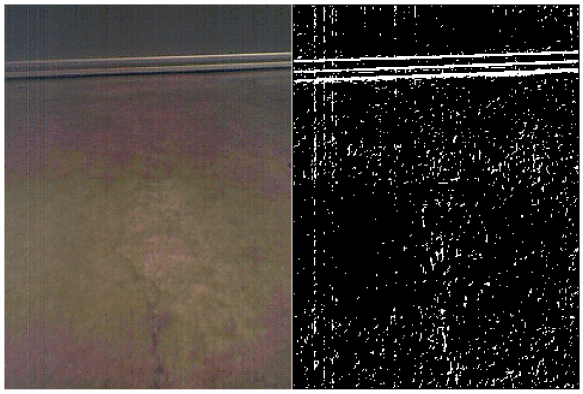


Figure 3.6: a) Raw image. b) processed image.

3.2.2 Line detection

Under normal circumstances, line information is represented with the standard equation for a line (see Equation 3.1).

$$Y = mX + B \tag{3.1}$$

This standard representation of a line is not a good choice for embedded implementations. Vertical lines have infinite slopes which cannot be represented in a computer.

It is for this reason that lines need to be represented in their polar form. Polar form lines are represented by ρ and ϕ, with ρ representing the distance from the origin to the perpendicular intersection of the line and ϕ being the angle between the line that ρ represents and the zero radian mark. Equations 3.2 and 3.3 show the representation of a line in polar form and Figure 3.7 shows it graphically.

$$\phi = \arctan(X/Y) \tag{3.2}$$

$$\rho = \cos(\phi)X + \sin(\phi)Y \tag{3.3}$$

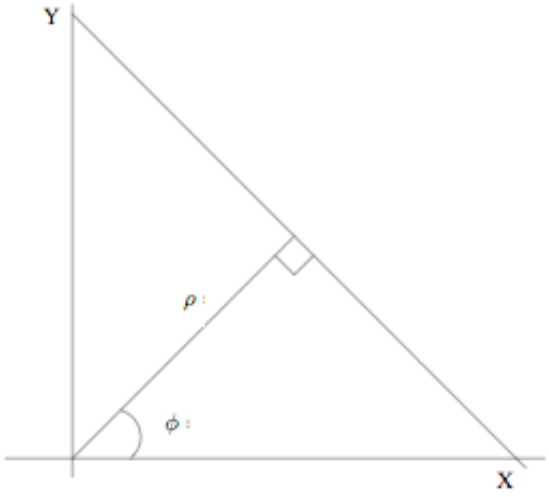


Figure 3.7: Representation of a line in polar form.

The line detection algorithm uses the Hough transform [30] to detect lines [31]. In the Hough transform, lines are represented and stored in Hough space [30]. Hough space is a two dimensional matrix with X representing ϕ in radians and Y representing ρ in pixels. The cells that are defined by X and Y represent a count or vote of how many pixels are found with a specific ρ and ϕ.

The Hough transform traverses every pixel in an image. If a pixel is an edge pixel, it votes in the Hough space for every line that would run through it. After the Hough transform is complete, each cell in the Hough space will have a count of how many pixels have voted for a line with a specific ϕ (X) and ρ (Y). The higher the count, the higher the confidence that a true line exists at that distance and orientation.

To clarify how the Hough transform works Figure 3.8 shows an example of a line detected by the Hough transform in image space and its representation in Hough space. In this example (Figure 3.8), there are several individual edge points that represent a line with a distance of ρ and an orientation of ϕ that is detected by the Hough transform. Each edge pixel in image space represents a sinusoidal curve in Hough space. The intersection of these curves represent a line in image space with a distance of ρ and an orientation of ϕ. The more curves that intersect at ρ and ϕ, the stronger the possibility that a line exists at that point.

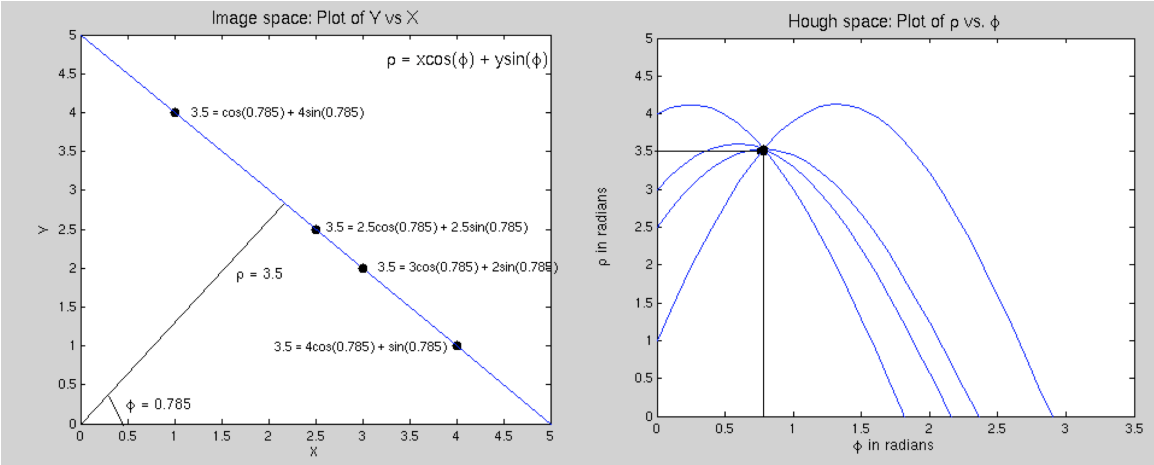


Figure 3.8: Example of a line detected by the Hough transform and its representation in Hough space.

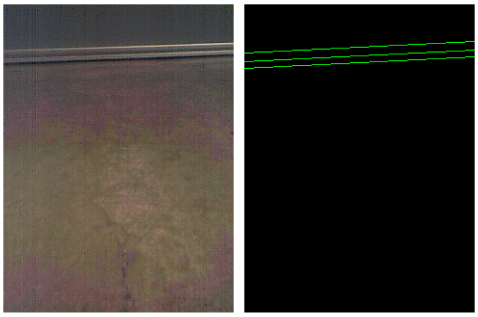


Figure 3.9: Example of extracted Hough lines. The image on the left shows the raw image. The image on the right shows the extracted Hough lines.

Once the Hough transform is complete, all lines with votes over a certain threshold will be extracted from the Hough space and stored in polar form for continued processing. An example of the extracted Hough lines can be seen in Figure 3.9

One problem that exists with trying to detect lines using the Hough transform is that image noise and discretization error can lead to lines in an image that are not detected. One way to compensate for this is to smooth the Hough space. In this thesis, each cell in the Hough space still represents a single ρ and ϕ but each time an edge pixel votes for a line in a cell, the neighbouring cells are also given a partial vote for that line.

Another problem with the Hough transform is that it is very computationally expensive and it is not well suited for real-time applications on an embedded platform. To increase the execution speed of the Hough transform, the line detector is limited to detecting lines within a specific range of orientations. The range of orientations is determined by analysing the lines that exist in the captured images. Lines that are close to being parallel to the view of the robot or horizontal lines have the most information that is useful in measuring ego-motion. The information that horizontal lines contain allow the ego-motion calculations to calculate vertical displacement as well as orientation changes in the mobile robot. Vertical lines are not a good feature to use in the estimation of ego-motion because they do not provide enough information alone to measure the vertical displacement of the robot. They are only useful to estimate rotational motion of the robot. Using this reasoning, the line detector is limited to search for lines with orientations in the range 40° to 140° (near horizontal to horizontal).

3.2.3 Line tracker

The line tracker is responsible for tracking detected image lines as they move throughout the image over time. It accomplishes this by comparing all newly extracted lines that have been found by the line detector to all lines that are currently being tracked.

When the line tracker was designed, two cases needed to be incorporated into the algorithm. These cases were; how the line tracker functions if the robot has moved a large distance and how the line tracker will distinguish one line from another when tracking multiple lines.

In the first case, when the robot is moved by an external force or has moved a large distance, newly detected lines cannot be guaranteed to be the same lines that the system is tracking. It is for this reason that I have conducted my research under the assumption that the robot cannot move in large increments and will not be moved by external forces.

In the second case, when the system is tracking several lines in one image and several in the next, it is a problem for the system to distinguish one line from another. In the most simple scenario, if the system is tracking one line, line tracking becomes a trivial problem. To track the single line, the system monitors the line's position as it moves throughout the image. When the system is tracking several lines at once, there needs to be a way to distinguish that an extracted line is a tracked line and not a new line. To solve this problem, I have come up with a simple method to track multiple image lines. This method uses a criteria of how a line should change given the motion of the robot. The criteria states that the change in ρ and change in ϕ can only be

within a certain threshold or the lines will not be considered the same. There are two separate thresholds for $\Delta\rho$ and $\Delta\phi$. The $\Delta\rho$ threshold is defined as MaxRhoChange and the $\Delta\phi$ threshold is defined as MaxPhiChange. The MaxRhoChange and MaxPhiChange thresholds used in this thesis were determined during testing using trial and error. There are two trade-offs that need to be considered when selecting the thresholds. The first is that if the thresholds are too restrictive, lines will not be tracked and the visual odometry system will rely solely on the motion model for ego-motion estimation. The second is that if the thresholds are non-restrictive, the line tracker will not be able to distinguish one line from another and the visual odometry system could produce a very large ego-motion estimation error. Given these two trade-offs it is better to rely solely on the motion model for ego-motion estimation rather than relying on a line tracker that could generate large errors in ego-motion estimation.

Line tracking begins when the line tracker receives a list of extracted lines from the line detector. If this is the first time that the line tracker has been executed or no lines are currently being tracked, the line tracker adds all the lines found by the line detector to a list of currently tracked lines. If the line tracker is currently tracking lines, the line tracker compares the list of currently tracked lines to the list of extracted lines provided by the line detector. Each entry in the list of tracked lines contains information about a tracked line's current position and its previous position. Figure 3.10 shows the information stored for each tracked line.

```
TrackedLineData{
    CurrentLinePosition.rho
    CurrentLinePosition.phi
    PreviousLinePosition.rho
    PreviousLinePosition.phi
    NotSeenCount
}
```

Figure 3.10: Information stored in each tracked line.

The line tracker tries to match each extracted line to a line in the list of tracked lines. It does this by comparing the ρ and ϕ of the extracted line to all of the lines in the list of tracked lines. If the difference between the extracted and a tracked line's ρ and ϕ are within the MaxRhoChange and the MaxPhiChange thresholds, the lines are considered to be the same and a successful match has been achieved. At this point, the tracked line's previous position is set to its current position and the tracked line's current position is set to the position of the extracted line. If no match is found, the extracted line is added as a new line in the list of tracked lines. If a tracked line is not found in the list of extracted lines, a count is kept of how many times this line has not been seen. If this count is above a set limit, then the line could not be tracked and it is removed from the list of tracked lines. Figure 3.11 displays the pseudocode for this algorithm.

```
//Traverse list of all lines currently being tracked
for( i = 0; i < SizeOf( TrackedLines ); i++){
        MATCHFOUND = FALSE
        //Traverse list of extracted lines
        for( j = 0; j < SizeOf( ExtractedLines ); j++){
                DeltaRho:= abs( TrackedLines[ i ]. currentPosition.rho - ExtractedLine.position.rho)
                DeltaPhi:= abs( TrackedLines[ i ]. currentPosition.phi - ExtractedLine.position.phi)
                //If tracked line and extracted line within threshold, a match found
                if( DeltaRho < MaxRhoChange AND DeltaPhi < MaxPhiChange){
                        MATCHFOUND = TRUE
                        //reset line not seen count
                        TrackedLines[ i ].notSeen = 0
                        //update tracked lines new position
                        TrackedLines[ i ].previousPosition = TrackedLines[ i ].currentPosition
                        TrackedLines[ i ].currentPosition = ExtractedLines[ i ].position
                        //Remove extracted line from list, so it is not checked again
                        ExtractedLines.removeElementAt( i )
                }
        }
        //If line not found in extracted line list then update not seen count
        //If not seen count greater than notSeen threshold remove tracked line
        if( MATCHFOUND == FALSE)
                TrackedLines[ i ].notSeen++
        if( TrackedLines[ i ].notSeen > NOTSEENTHRESHOLD)
                TrackedLines.removeElementAt( i )

}
```

Figure 3.11: Pseudocode for the line tracking algorithm.

The output of the line tracking algorithm is a list of all successfully tracked lines. This information can be utilized by the displacement / orientation calculation to calculate the ego-motion of the robot.

3.2.4 Displacement / Orientation Calculation

The displacement and orientation calculation is responsible for calculating the ego-motion of the mobile robot. This step calculates the ego-motion by analysing the list of tracked lines that is populated by the line tracker. Once the list of tracked lines has been analysed, the ego-motion of the mobile robot is estimated. If the tracked line list does not contain any lines, then this step uses the motion model to determine the ego-motion of the robot.

The displacement and orientation calculation estimates robot ego-motion in terms of the change in robot orientation $\Delta\theta$ and the displacement of the robot S. The calculation analyses all of the lines in the tracked line list and calculates every line's $\Delta\theta$ and S. After $\Delta\theta$ and S have been calculated for all individually tracked lines, the ego-motion of the robot is determined by taking the median $\Delta\theta$ and the median S. The median values of $\Delta\theta$ and S are chosen to eliminate outliers in the ego-motion estimation. To help reinforce the overall picture of the algorithm, Figure 3.12 displays its pseudocode.

```
//Traverse list of all lines currently being tracked
for( i = 0; i < SizeOf( TrackedLines ); i++){

   // Calculate change in line orientation relative to centre of robot's view
   DeltaOrientationArray[ i ] := CalculateChangeInLineOrientation ( TrackedLine[ i ] )

  // Calculate change in robot's distance to line relative to centre of robot's view
   DeltaDistanceArray[ i ] := CalculateChangeInLineDistance( TrackedLine[ i ] )
}

// Change in Robot orientation relative to centre of robot's view
changeInRobotOrientation := medianValue (  DeltaOrientationArray[ ] )

// Robot displacement relative to centre of robot's view
robotDisplacement := medianValue ( DeltaDistanceArray[ ] )
```

Figure 3.12: Pseudocode of the displacement and orientation algorithm.

The change in a tracked line's orientation $\Delta\theta$ is calculated by analysing its previous and current positions. Using this position information, the line's previous and current orientations are calculated. The change in the line's orientation is the difference between its previous orientation and current orientation. A positive change in line orientation indicates that the robot has turned left. A negative change in line orientation indicates that the robot has turned right. Figure 3.13 shows the pseudocode of the orientation calculation.

```
//Calculate change in line orientation relative to centre of robot's view
calculateChangeInLineOrientation( TrackedLine ){

   //Calculate the tracked line's previous orientation relative to centre of robot's view
   prevOrientation :=  calculateOrientation( TrackedLine.previousPosition )

   //Calculate the tracked line's current orientation relative to centre of robot's view
   currentOrientation :=  calculateOrientation( TrackedLine.currentPosition )

  //Calculate change in robot orientation
  orientationChange := prevOrientation - currentOrientation

  //Return change in robot orientation
  return  orientationChange
}
```

Figure 3.13: Pseudocode of the change in orientation calculation.

The displacement S of a tracked line is calculated by analysing its previous and current positions. A tracked line's previous and current distances to the robot are calculated using this position data. The displacement of the line is the difference between its previous distance to the robot and its current distance to the robot. A positive line displacement indicates that the robot has moved forward. A negative line displacement indicates that the robot has moved backwards. Figure 3.14 shows the pseudocode of the displacement calculation.

```
//Calculate the change in the line's distance to the robot, relative to centre of robot's view
calculateChangeInLineDistance( TrackedLine ){

   //Calculate the tracked line's previous distance to the robot relative to centre of robot's
view
   prevDistance :=  calculateDistanceToLine( TrackedLine.previousPosition )

   //Calculate the tracked line's current distance to the robot relative to centre of robot's
view
   currentDistance := calculateDistanceToLine( TrackedLine.currentPosition )

  //Calculate robot displacement
  displacement := prevDistance - currentDistance

 //Return robot displacement
 return  displacement
}
```

Figure 3.14: Pseudocode of the displacement calculation.

3.3 Task Controller

A simple task controller was developed to determine the feasibility of the visual odometer. The task controller determines the overall behaviour of the robot. It utilizes the ego-motion estimated by the visual odometry algorithm to perform three functions. These are to update the robot's belief of where it is in the world (localization), execute the path that the robot must follow and develop the necessary control outputs that are sent to the drive system.

Given the initial hardware that was supplied for this thesis (see chapter 4 for description), it was noted that the captured images could not be processed in real time. This was due to the limitations of the embedded system and the requirements of the image processing algorithms. A normal iteration of the image processing algorithms (from capture to final processing) required 25 seconds to complete. Therefore, the task controller had to be designed to allow for the slow processing time.

The task controller's overall or high level design allows the robot to capture an image, process the image, calculate the ego-motion, decide what action needs to take place and finally execute that action (drive the robot). This is considered one iteration or update of the control loop of the robot. Figure 3.15 shows a flow chart of one iteration of the control of the robot.

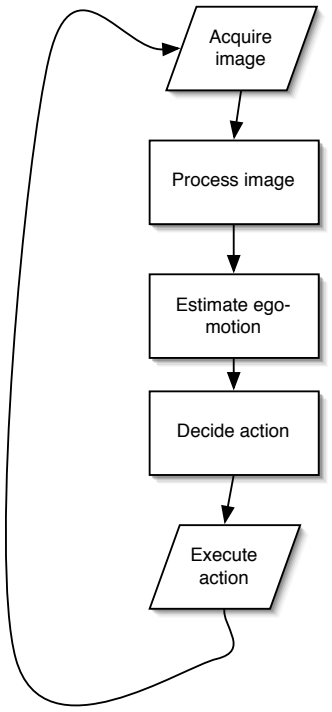


Figure 3.15: Flow chart of one iteration of the robot controller.

The low level design of the task controller allows the robot to update its belief of where it is using the information from the visual odometry algorithm or motion model. It also executes and controls a state-machine that decides which path the robot should travel by sending commands to the drive system. This state-machine will be referred to as the path state-machine. Figure 3.16 shows a flow chart of one iteration of the low level control of the task controller.

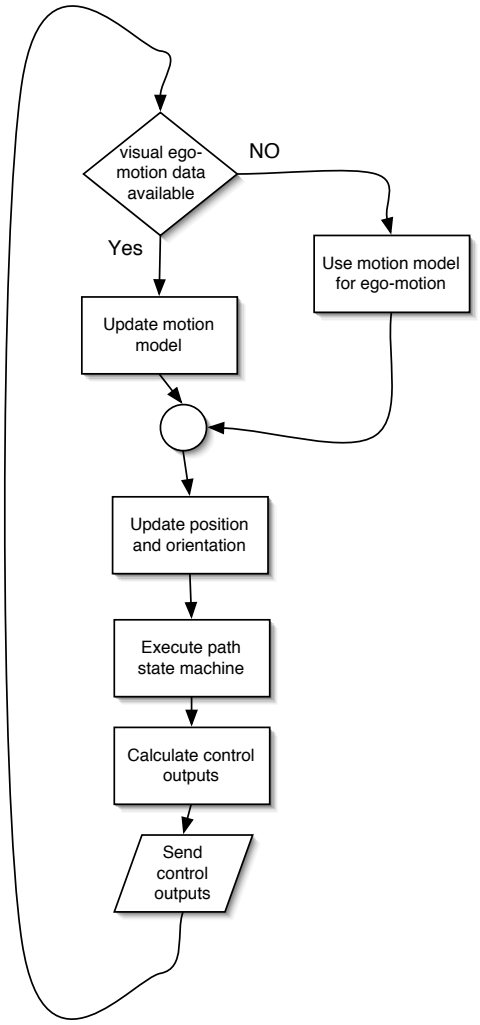


Figure 3.16: Flow chart of the low level design of the task controller.

The low level control of the task controller begins after the robot's ego-motion has been calculated. If the visual odometry algorithm is able to estimate the ego-motion of the robot, then the task controller uses the calculated ego-motion to update the robot's position, orientation and motion model. If the visual odometry algorithm is unable to estimate the ego-motion of the robot, then the task controller uses dead reckoning (by using the motion model information) to estimate the ego-motion and

update the robot's position and orientation. Once this is complete, the path state-machine is executed. The path state-machine decides which action should take place based on where the robot is positioned. After the path state-machine has decided which action the robot should execute, the drive command is determined and sent to the drive system.

The path state-machine controls the robot to execute different paths or patterns that are particular to different applications. In this thesis the patterns that are executed are specific to the evaluation that took place and are a linear traversal and a figure eight. Each state representes a different part of the pattern that the robot is supposed to execute. The path state-machine has two different behaviours that can be executed in each state. These behaviours are defined as turn angle and move distance.

The turn angle behaviour controls the robot to turn an angular amount specified by the path state-machine. It does this by commanding the robot to leftTurn( ) increment or rightTurn( ) increment and checks how far the robot has turned at the next pass through the path state-machine. If the robot has turned the angle that has been specified in that state, then that state is complete and the robot executes the next state in the path state-machine. Figure 3.17 shows the pseudocode for this state behaviour.

```
//Example  of the turn angle behaviour in the path state-machine.
//This example is to turn the robot left by PI radians.
else if ( pathState == 1 ){       // Turn by PI

            SP = 3.14; //Set point is PI radians
            //deltaTheta calculated by visual ego-motion or motion model
            MV+=  deltaTheta;  //Measured variable is the sum of the angle changed for this state
            motionState = MOTION_STATE_LTURN;  //Commands drive system to turn left

            if ( MV >= SP ) { //if reached set point, turn stop turning
                        motionState = MOTION_STATE_OFF;
                        MV = 0;
                        pathState++;
            }
}
```

Figure 3.17: Pseudocode of the turn angle behaviour.

The move distance behaviour controls the robot to move a distance amount specified by the path state-machine. It does this by commanding the robot to moveForward( ) increment or moveReverse( ) increment and checks how far the robot has moved at the next pass through the path state-machine. If the robot has moved the amount that has been specified in that state, then that state is complete and the robot executes the next state in the path state-machine. Figure 3.18 shows the pseudocode for the move distance state behaviour.

```
//Example of the move distance behaviour in the path state-machine.
//This example is to drive the robot straight by 200cm
else if ( pathState == 2 ){       // drive forward 200 cm

            SP = 200.0; //Set point is 200cm forward
            //deltaS calculated by visual ego-motion or motion model
            MV+=  deltaS;  //Measured variable is the sum of the distance changed for this state
            motionState = MOTION_STATE_FWD;  //Commands drive system to drive forward

            if ( MV >= SP ) { //if reached set point, turn stop driving
                        motionState = MOTION_STATE_OFF;
                        MV = 0;
                        pathState++;
            }
}
```

Figure 3.18: Pseudocode of the move distance behaviour.

By using a combination of both the turn angle and move distance behaviours, any pattern can be developed and the robot can be commanded to execute the developed pattern. For example, if the desired pattern is for the robot to drive straight for 200 cm, turn around (turn 3.14 radians) and drive back (drive 200 cm), then the path state-machine in Figure 3.19 would be used. Figure 3.19 shows the pseudocode for this state-machine.

```
//Traverse 200 cm forward
if ( pathState == 0 ){   // drive forward 200 cm
        SP = 200.0; //Set point is 200cm forward
        //deltaS calculated by visual ego-motion or motion model
        MV+=  deltaS;  //Measured variable is the sum of the distance changed for this state
        motionState = MOTION_STATE_FWD;  //Commands drive system to drive forward

        if ( MV >= SP ) { //if reached set point, turn stop driving
                motionState = MOTION_STATE_OFF;
                MV = 0;
                pathState++;
        }
}

//Turn robot around
else if ( pathState == 1 ){          // Turn by PI
        SP = 3.14; //Set point is PI radians
        //deltaTheta calculated by visual ego-motion or motion model
        MV+=  deltaTheta;  //Measured variable is the sum of the angle changed for this state
        motionState = MOTION_STATE_LTURN;  //Commands drive system to turn left

        if ( MV >= SP ) { //if reached set point, turn stop turning
                motionState = MOTION_STATE_OFF;
                MV = 0;
                pathState++;
        }
}
```

```
//Come back to start
else if ( pathState == 2 ){          // drive forward 200 cm

        SP = 200.0; //Set point is 200cm forward
        //deltaS calculated by visual ego-motion or motion model
        MV+=  deltaS;  //Measured variable is the sum of the distance changed for this state
        motionState = MOTION_STATE_FWD;  //Commands drive system to drive forward

        if ( MV >= SP ) { //if reached set point, turn stop driving
                motionState = MOTION_STATE_OFF;
                MV = 0;
                pathState = 99; //pattern completed
        }
}
```

Figure 3.19: Pseudocode of example path state-machine.

Chapter 4

Visual Odometry System Hardware / Software Implementation

The visual odometry system cannot function on its own. It requires the addition and integration of multiple system components. This chapter describes each of the additional system components which include both hardware and software.

4.1 Mobile Platform

The mobile platform is a custom platform that was developed over the course of this research. It consists of two continuous rotation servos, a servo controller and behaves as a differential drive robot.

The initial goal was for the platform to function as a disposable Urban Search and Rescue (USAR) robot. This led to the first version of the design. The initial design was constructed out of thick plastic sheeting. Figure 4.1 shows the first plastic

prototype of the designed platform. This platform competed in the AAAI robot competition in July 2003 Mexico. This platform did not survive the competition because it was not sturdy enough, so a second design had to be created.

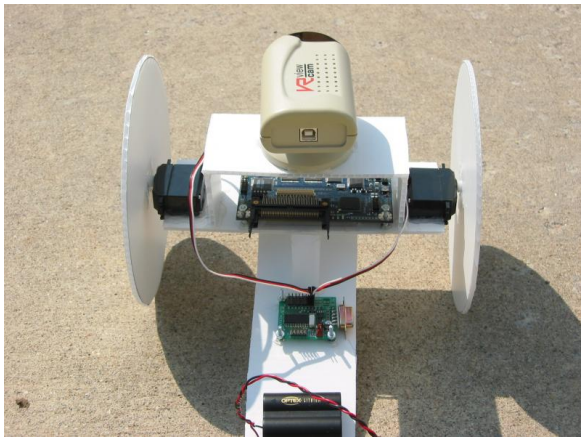


Figure 4.1: First design of the mobile platform.

The second design of the platform was the same as the first except that it was constructed to withstand more stress and handling. This was accomplished by reinforcing the platform's joints with plastic sheeting. This design of the platform was used for six months of research until it finally self-destructed from excessive use and testing. Figure 4.2 shows the second design of the platform. It was determined that plastic sheeting was not strong enough to use in the USAR domain.

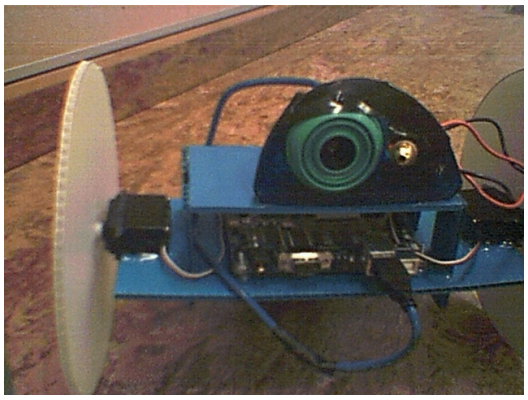


Figure 4.2: Second design of the mobile platform.

The experience gained from constructing and using the first two models of the platform led to the third design which incorporated a steel frame for the mobile robot. This design was much more stable than the first two and it was used for most of the conducted research. The steel platform competed successfully in the 2004 Robocup Rescue Competition in Lisbon, Portugal. Figure 4.3 shows the third design of the mobile platform.

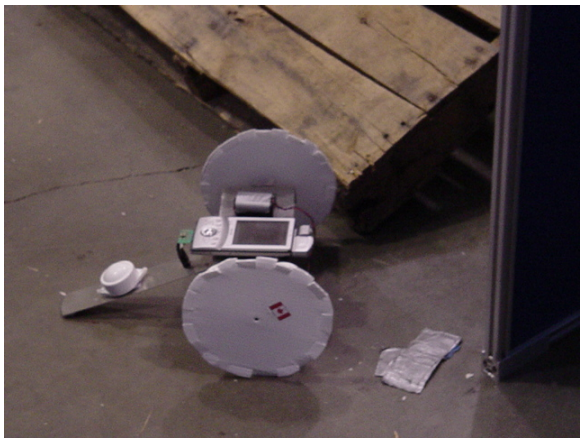


Figure 4.3: Third design of the mobile platform.

The steel frame design of the platform worked best overall and is still in use today. Because of the evolution of the embedded platform (see Section 4.2), the mobile platform had one more design revision to incorporate the external USB camera and accommodate for the larger PDA. Figure 4.4 shows the current version of the mobile platform that was used for the final research that was conducted.

Figure 4.4: Current version of the mobile platform.

4.2 Embedded Platform

The initial embedded system that was used for the visual odometry system was an Intel Stayton board. The Stayton board is a single board computer with the following specifications:

- 400Mhz Intel PXZ250 Xscale processor using ARM instruction set
- 32MB Ram
- ARM Linux embedded OS
- USB host port

Early on in the design phase, it was discovered that using a colour screen to view the captured images would aid in the troubleshooting of the image processing algorithms. The colour screen allowed the results of these algorithms to be seen in real-time rather than waiting for them to be uploaded and viewed off-line. Given the debugging capabilities of the colour screen and the fact that the Stayton board was a pre-production embedded system that was not easily obtainable, it was decided to obtain an embedded system that was readably available and had a colour screen.

The embedded system that was chosen to replace the Stayton board was a Sharp Sl-5000D Zaurus PDA with the following specifications.

- 200Mhz Intel StrongARM processor
- 32MB Ram
- ARM Linux embedded OS
- Compact flash camera
- 3.5 Inch TFT touch screen

After using this system, it was found that the colour screen was an invaluable resource that reduced the amount of time needed to design and test algorithms. One problem was found with this Zaurus. The processor and compact flash camera were not able to capture and process images at rate which would allow the robot to move continuously. This limitation led to the control algorithms that were developed in Section 3.3.

In the final testing phase of the research, a new embedded system was available and used for the evaluation of the system. This system was a Sharp SL-C1000 Zaurus Linux handheld with the following specifications.

- 416Mhz Intel PXZ270 Xscale processor using ARM instruction set
- 64MB Ram
- ARM Linux embedded OS
- USB host port
- 3.7 Inch TFT touch screen

Additional hardware that was used by the embedded platform was a Logitech QuickCam 4000Pro which captured the required images and a serial connection to the drive system which allowed communications between the drive system and the embedded system.

4.3 Camera Calibration

To successfully calculate the ego-motion of the mobile robot, the visual odometry system needed to be able to understand what a pixel's real world coordinates were. These real world coordinates provided the visual odometry system with a pixel's real X coordinate and Y coordinate values in millimetres. To determine a pixel's real world values, the visual sensor needed to be calibrated. The calibration methods used in this research are based on Roger Tsai's calibration model [32] and Reg Wilson's initial PC based implementation [33]. To calibrate the camera, test images were acquired from a test pattern that was placed in front of the robot (see Figure 4.5). The test images were then run through an off-line Tsai model [32] based coplanar calibration program. Coplanar calibration is a two dimensional camera calibration that requires the calibration points to be two dimensional. This meant that all calibration points were situated on the ground plane with a Z axis value of zero. The output of the calibration program yielded a two dimensional matrix that was integrated into the robot program as a look up table. After the sensor had been calibrated, the system was able to obtain a real world (X,Y) coordinate value from a given pixel position using this table.

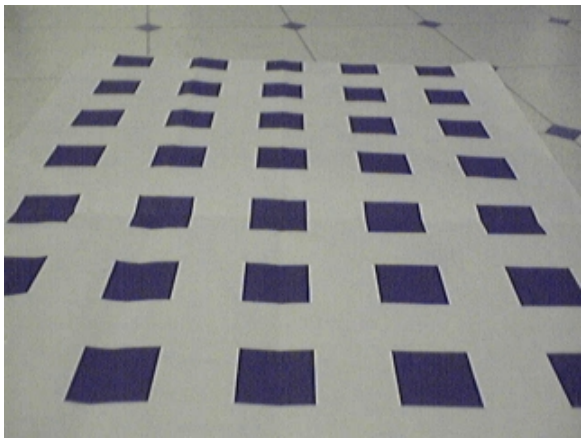


Figure 4.5: Picture of the calibration test pattern used in the calibration of the camera.

4.4 Application GUI

To take advantage of the embedded system's colour screen, a custom application GUI needed to be developed. The GUI allowed the camera's raw image and the processed images to be displayed to the user. The GUI also allowed for the robot's drive system and control parameters to be quickly modified. The GUI was constructed using the QT toolkit from Trolltech Inc. The GUI consists of a main control pane that controls the starting, stopping and other different functions of the robot. Figure 4.6 shows the control GUI pane.

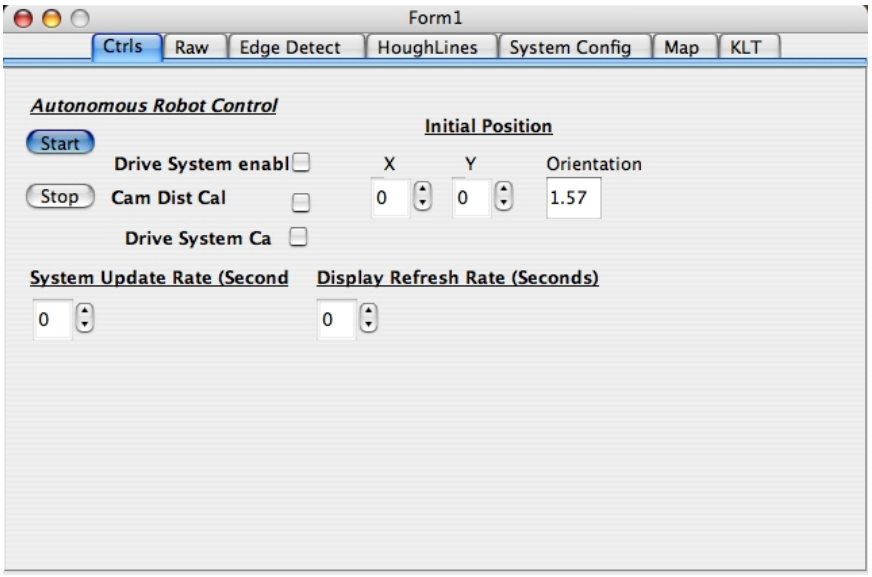


Figure 4.6: GUI pane for controlling the robot.

The GUI also contains three image viewers, one for viewing the raw images, one for viewing the detected edges and one that displays all extracted lines found by the line detection algorithm. Figure 4.7 displays the raw image viewing pane.

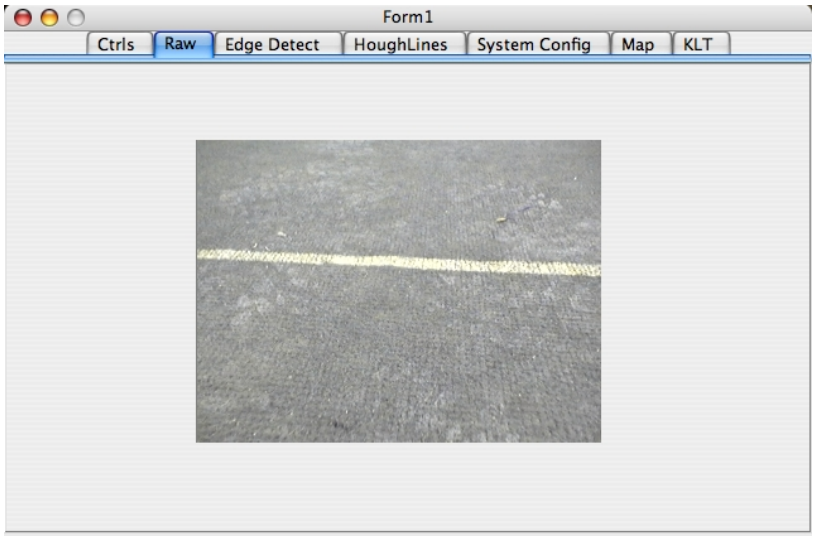


Figure 4.7: GUI pane that displays raw images captured by the camera.

The GUI also has a pane that allows a user to set all of the parameters of the drive system and tune the drive system for accurate dead reckoning. Figure 4.8 displays the drive system control pane.

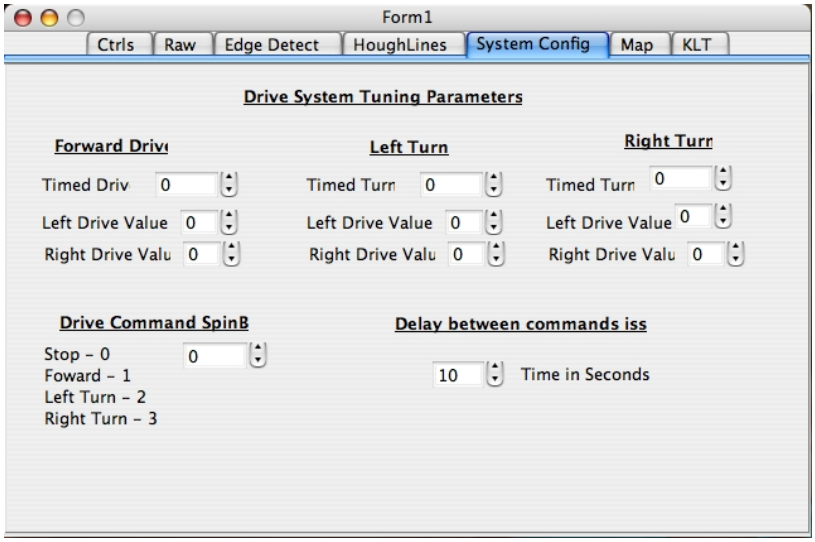


Figure 4.8: GUI pane that controls the drive system.

4.5 Drive System

The drive system consists of a standalone servo controller and software interface. The servo controller is a Pontech SV203 micro-controller which communicates with the embedded platform (Zaurus) to control the mobile platform's actuators. The software interface consists of a separate thread to control the robot's motion and five motion commands to move the robot. The following lists the commands developed for this system.

- driveForward( ); //drive robot straight
- driveReverse( ); //drive robot straight back
- stop( ); //stops robot
- turnLeft( ); //turn robot left on the spot
- turnRight( ); //turn robot left on the spot

The driving and turning commands turn and move the robot a predetermined amount of radians and distance in millimetres. These distances and radians are set for accurate dead reckoning performance by tuning each actuator through the drive system GUI pane.

Chapter 5

System Evaluation

The following chapter describes the evaluation of the visual odometry system that was developed. The visual odometry system was evaluated against two comparison systems. Each of the three systems completed two different tests. The results of each of the comparison systems were compared to the results obtained by the visual odometry system. A final discussion will evaluate the results and performance of each of the three systems.

5.1 System Comparisons

Two different systems were implemented and compared against the visual odometry system in order to evaluate its performance.

The first comparison system used a commercial mobile robot. It was based on an ActivMedia Pioneer. The Pioneer is a differential drive robot that uses only shaft encoders for ego-motion estimation. All of the shaft encoder based position

measurement algorithms have already been developed and are part of the Player-Stage [34] software that currently controls the Pioneer. This robot was programmed to run the predetermined test courses which are explained later in this section. For the rest of this thesis, the Pioneer will be referred to as the shaft encoder system.

The second comparison system was an optical flow based odometry system. This system utilized the mobile platform and the embedded system that was developed for the visual odometry system. The optical flow based system is an implementation of a KLT [1] based visual ego-motion system. The KLT algorithm has been developed by Stan Birchfield [35]. The optical flow odometry algorithm utilizes the tracked features returned by the KLT algorithm to determine the ego-motion of the robot. Ego-motion is calculated by first taking each tracked feature and then measuring its change in position between two consecutive images. The measurement provides the change in X and Y of a feature point. This information is then converted to the change in robot distance and angle. In this implementation, the KLT feature tracker tracks a maximum of 100 different features in a image. When the KLT feature tracker has completed an update, the algorithm is left with 100 data points that each provide the change in robot position and angle. In order to try to eliminate outliers in the data and provide a more accurate estimation of the ego-motion, the median change in robot distance and angle is selected to be the estimated ego-motion of the mobile robot.

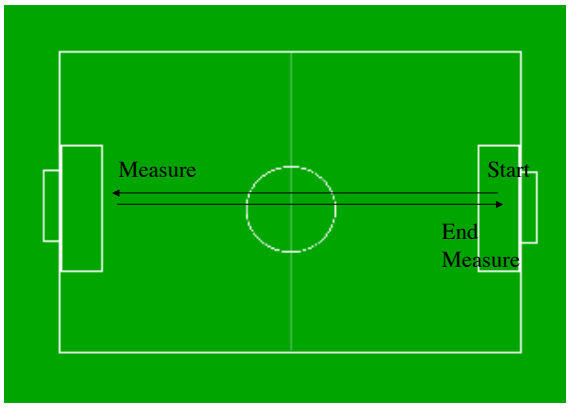


Figure 5.1: Example of Test 1.

5.2 Test 1: Linear traversal of a soccer field

In this test, each robot started at one end of a 274 cm by 152 cm soccer field, traversed the field and stopped when it thought that it had reached the other side (see Figure 5.1 for a diagram of the field and test). At this point, a record was made of the robot's position. The test continued and the robot autonomously turned around and stopped. Once again, a record was made of the robot's position. Next, the robot traversed the soccer field back to its starting point. The robot's position and orientation were once again recorded. When the test was finally complete, the recorded data was analysed and the position and orientation data of each robot was compared to all other robots. Each robot was tested in two separate test runs, Run 1 and Run 2. The decision to only conduct two test runs will be explained in detail in 5.6.2.

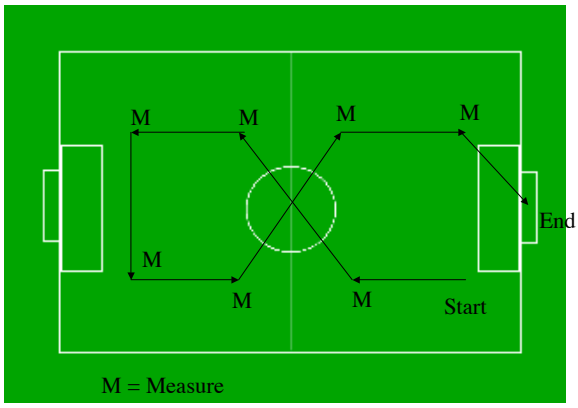


Figure 5.2: Example of Test 2.

5.3 Test 2: Figure eight path traversal of a soccer field

In this test, each robot started at one end and side of a 274 cm by 152 cm soccer field, traversed the field autonomously in a figure eight and ended up at the goal line near its starting position (see Figure 5.2 for a diagram of the field and test). Each time the robot thought that it had reached a measurement point, its position and orientation were recorded. A measurement point is where the robot stopped after a linear traversal or a turn. When the test was finally complete, the recorded data was analysed and the position and orientation data of each robot was compared to all other robots. Each robot was tested in two separate test runs, Run 1 and Run 2. The decision to only conduct two test runs will be explained in detail in 5.6.2.

5.4 Test variation: Environmental debris

Test 1 was repeated with thin rods placed arbitrarily on the field. These rods were used in order to cause an introduction of error in the odometeric system of each robot. The purpose of this test was to evaluate how each system performed with this introduced error. See Figure 5.3 for an example of the testbed with environmental debris.

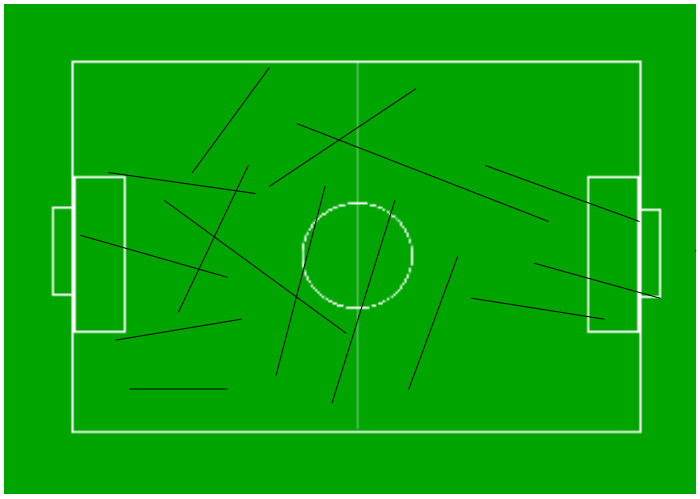


Figure 5.3: Example of testbed with environment debris.

5.5 Experimental Results

The following section evaluates and compares the test results of the developed system to the results obtained from each of the comparison systems. All experimental result graphs (Figures 5.5 through 5.10) show the outcomes of running each system through all tests and test runs. The graphs represent each system's traversal of the specific test pattern on the Robocup E-league [3] soccer field. The Y axis on the graphs represent the soccer field from one goal area to the other goal area. The X axis on the graphs represent one side of the soccer field to the other side. This is graphically described in Figure 5.4.

When initially testing the visual odometry system, it was discovered that this system needed to be evaluated using two different camera orientations. These were with the camera positioned facing down and the camera looking forward. It was determined early on in testing that having the camera pointing down resulted in the soccer field lines only being observed for a fraction of the test time. This caused the visual odometry system to rely heavily on dead reckoning for ego-motion estimation. When the camera was pointed in a more forward looking position, the system was able to observe field lines for the majority of the test time and each pixel in the upper part of the image represented a change in distance from 1 cm to 6 cm. This meant that if the line data had an error of one pixel, the robot could think it had moved 6 cm or rotated by 40 degrees. Each of these camera orientations caused the developed system to behave differently. For the remainder of this thesis, the two camera orientations will be considered as two separate systems and will be referred to as, visual odometry system camera down (VOCD) and visual odometry system camera forward

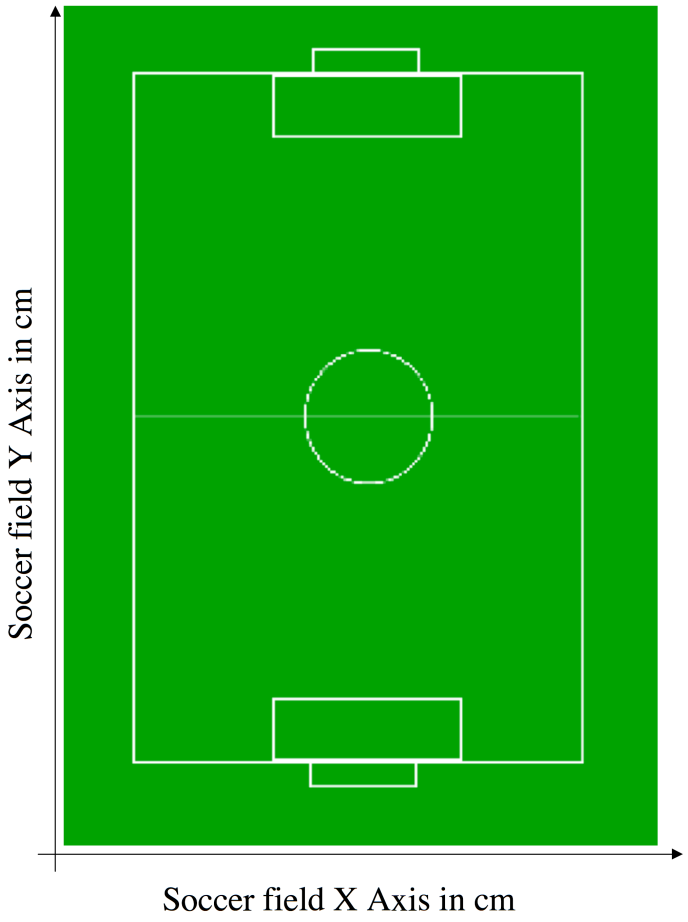


Figure 5.4: Graphical representation of the test result graphs X and Y axis.

(VOCF). The problems and shortcomings associated with each camera orientation will be explained in more detail in Section 5.6.

Another problem that was observed when initially testing the visual odometry system was that the platform's motion was unpredictable. This unpredictability caused the robot to move a variable amount of distance forward or turn by an unknown amount of radians after a motion command had been issued. The cause and problems associated with this unpredictability will explained in detail in 5.6.

In the initial implementation of the KLT system, a problem was found when the algorithm was cross compiled to run on the Zaurus embedded computer. The problem was that each iteration of the KLT algorithm took approximately 90 seconds to complete. This meant that a test requiring 200 iterations would last for five hours. If there was a problem with the robot during a five hour test, it would mean that the test would have to be rerun. This is not feasible as the battery capacity of the Zaurus was only two hours.

To alleviate this problem, a new approach had to be developed. During the testing of the visual odometry system, the robot collected and stored images at every iteration. Once each test was complete, the test images were fed into an offline KLT ego-motion system where the ego-motion of the robot was estimated. Using these saved images, the offline KLT system was able to process each image in less than a second resulting in an improvement of a factor of 90.

After analysing the results of running both Test 1 and Test 2 through the KLT system (Subsections 5.5.1, 5.5.2 and 5.5.3), it was discovered that this system did not provide good results. The motion field returned by each iteration of the KLT algorithm provided an inaccurate representation of the robot's motion. It was thought that this motion field problem was due to feature aliasing caused by the robot moving

too far between successive image captures.

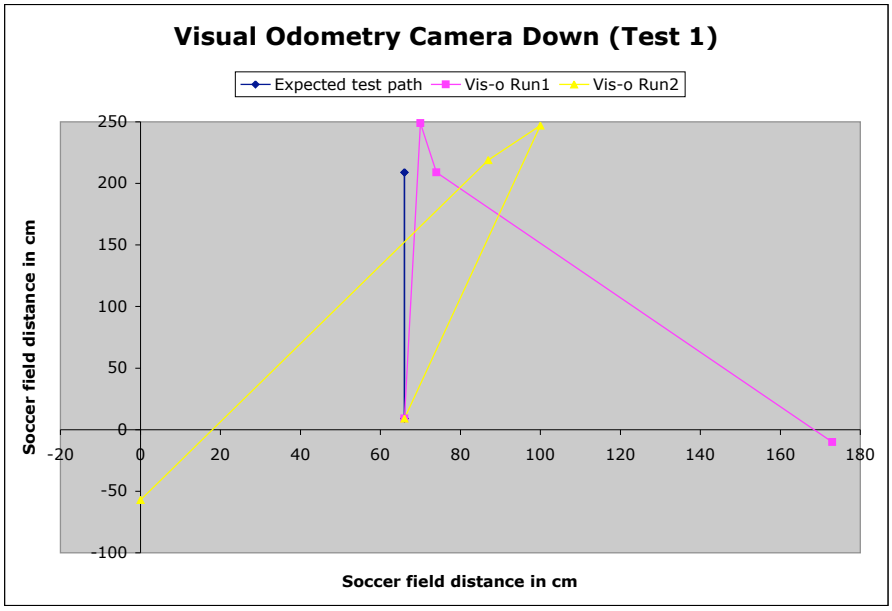
At this point, an informal test was performed to discover if this was actually the cause of the problem. In this test, the robot was moved a small distance between successive image captures. This step was repeated until the robot had captured 100 images. These images were then processed in the offline KLT system. The motion fields were visually analyzed and it was found that these motion fields accurately represented the robot's motion. Therefore, the initial assumption that feature aliasing had caused the poor performance of the KLT system in Test 1 and Test 2 was correct. This will be explained in detail in 5.6.

5.5.1 Test 1 Results: Linear traversal of a soccer field

Figures 5.5 and 5.6 show the results obtained from running Test 1 on the VOCD, the VOCF, the KLT system and the shaft encoder system. All data in the test was determined and recorded by physically measuring the position and orientation of each robot at every measurement point.

All of these systems could not perform the test pattern. The shaft encoder system performed somewhat better than the VOCD and the VOCF. It was surprising to discover that the shaft encoder system did not follow the test pattern. It was assumed that this system would have performed better than it had, given that there was no wheel slip present. The VOCD performed better than expected given that the robot only saw the field lines part of the time during both test runs. The difference in the performance of the VOCD and the VOCF are due to their different camera orientations and the problems associated with these orientations (as already discussed at the beginning of this section). These problems will be discussed in detail in Section 5.6.

The results of the KLT system did not reflect the motion of the robot and did not produce good ego-motion estimation during both Run 1 and Run 2. This was due to the problem of the robot moving too far between image captures (as already discussed at the beginning of this section). This problem will be discussed in detail in Section 5.6.



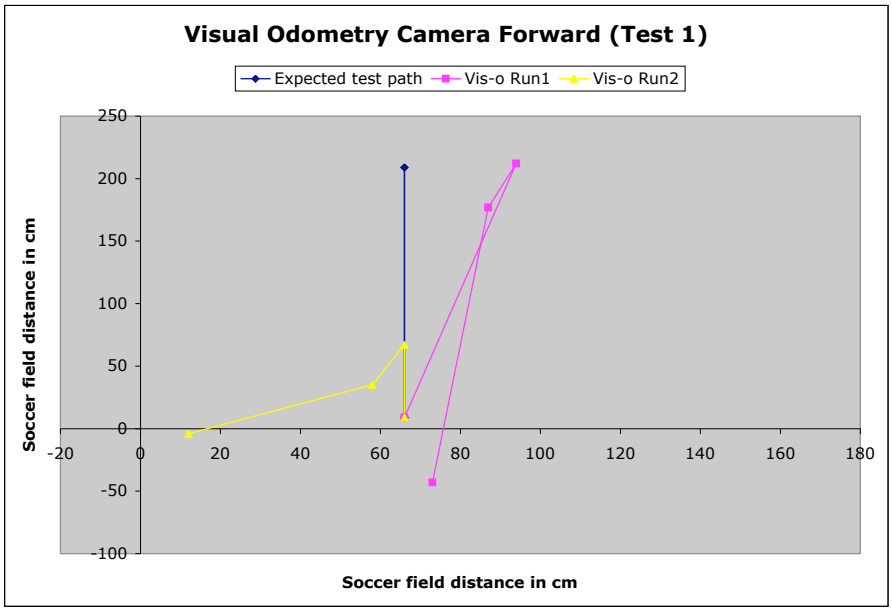
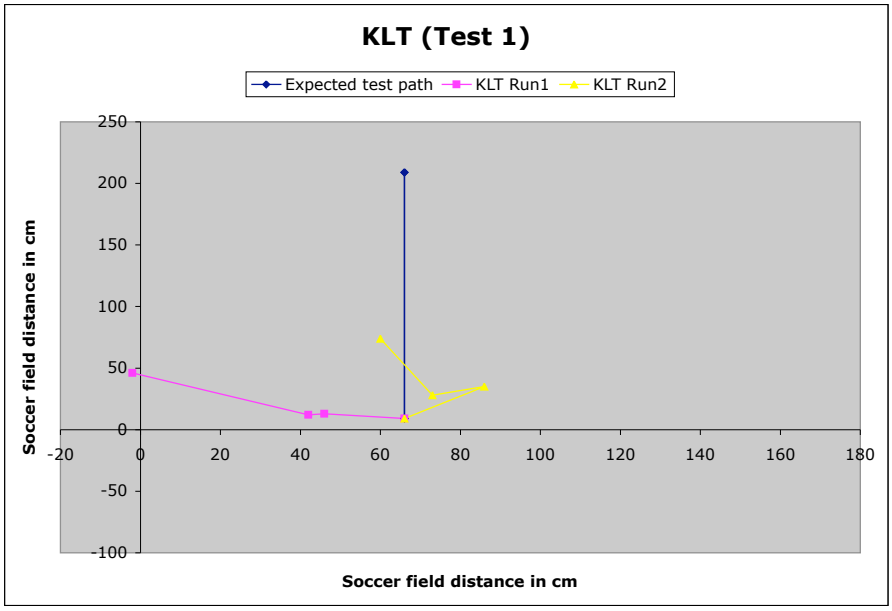


Figure 5.5: Test 1 visual odometry camera down and forward results.



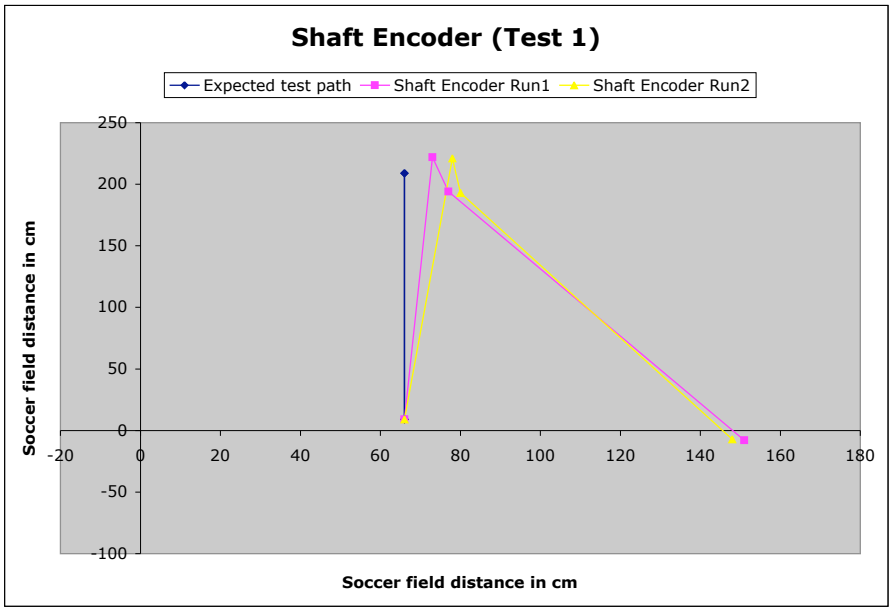


Figure 5.6: Test 1 KLT and Pioneer results.

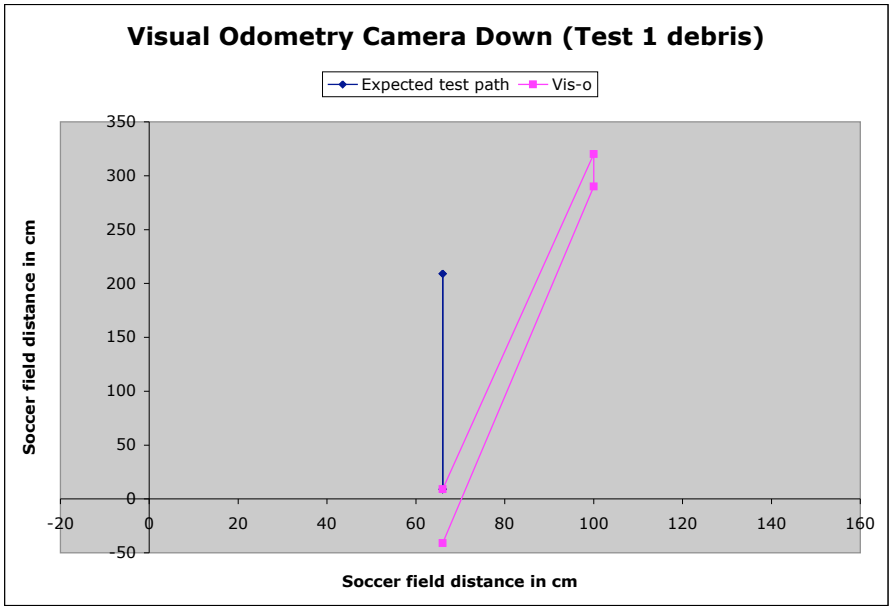
5.5.2 Test 1 variation: Environmental debris

Figures 5.7 and 5.8 show the results of running all systems once through Test 1 with environmental debris placed on the course. In this test, the shaft encoder system could not complete the test pattern and its results were similar to the results obtained in Test 1 (no debris). The environmental debris that was placed on the field had no large negative impact on the system. Possibly there was not enough course debris or the robot was too heavy to be affected by it.

The assumption for the purpose of this thesis was that the introduction of environmental debris would affect the visual odometry system during operation. It was also assumed that the visual odometry algorithm would compensate for this and allow the system to correct for the wheel slip introduced by the debris. From the results obtained, this was not the case. The environmental debris introduced a situation that had not been anticipated beforehand. The debris (thin rods) occluded the field lines adding many more lines to the captured images. This situation caused the Hough transform [30] to detect false lines created by the debris. With the introduction of the debris lines and the falsely detected lines, the line tracker was unable to distinguish one line from another and track the real field lines. The outcome of this situation was that the visual odometry algorithm was rendered unusable for the majority of the test. This resulted in incorrect ego-motion estimation which degraded the performance of the VOCF and VOCD.

The results of the KLT system did not reflect the motion of the robot and did not produce good ego-motion estimation during this test. This was due to the problem of the robot moving too far between image captures (as already discussed at the

beginning of this section). This problem will be discussed in detail in Section 5.6.



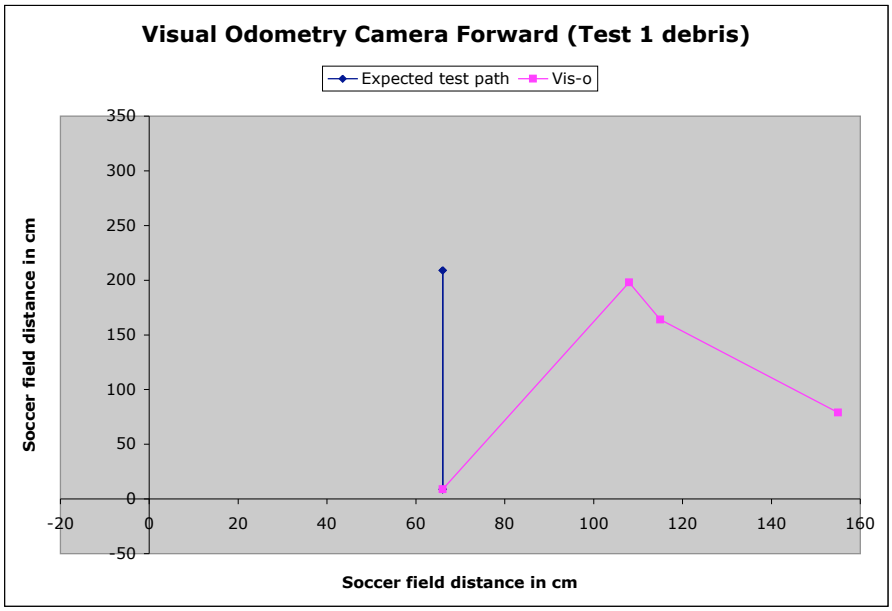
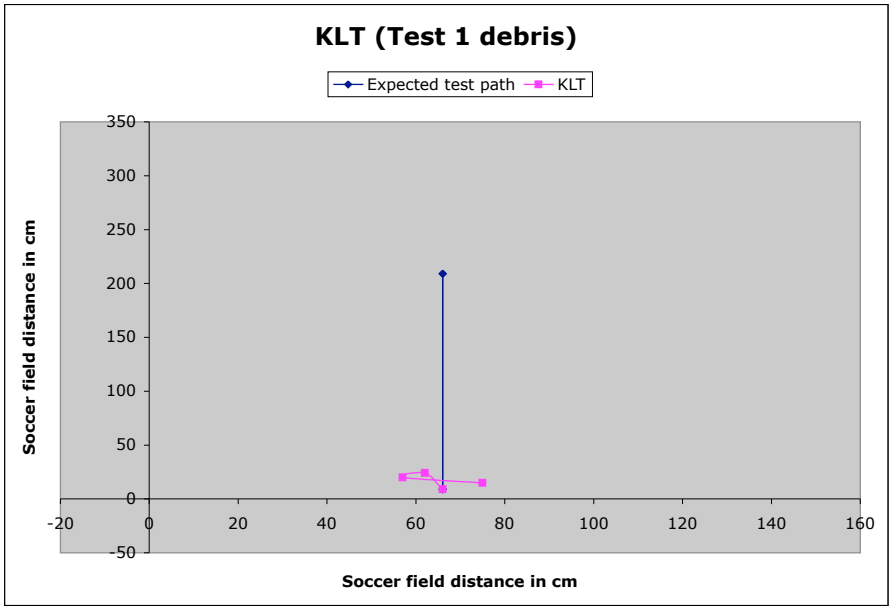


Figure 5.7: Test variation: Environmental debris visual odometry camera down and forward results.



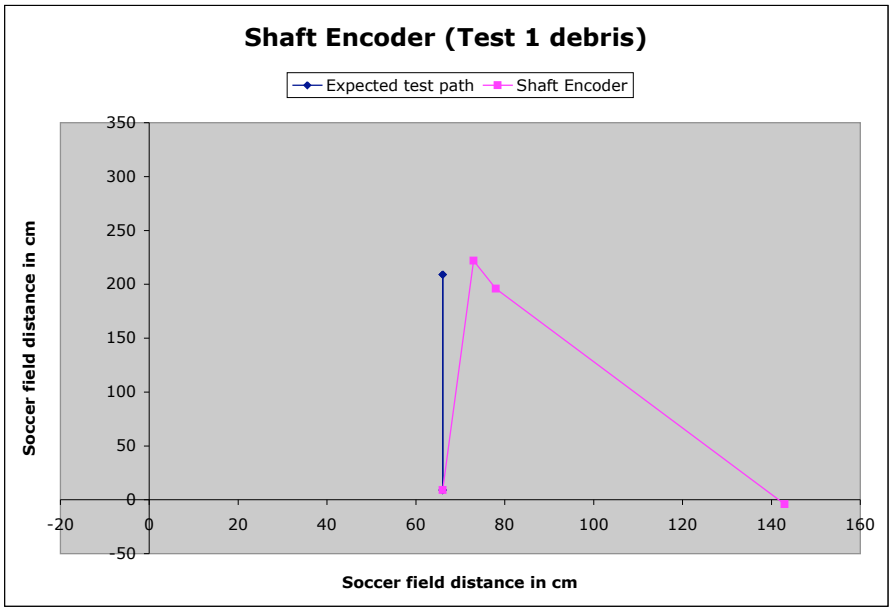


Figure 5.8: Test variation: Environmental debris KLT and Pioneer results.

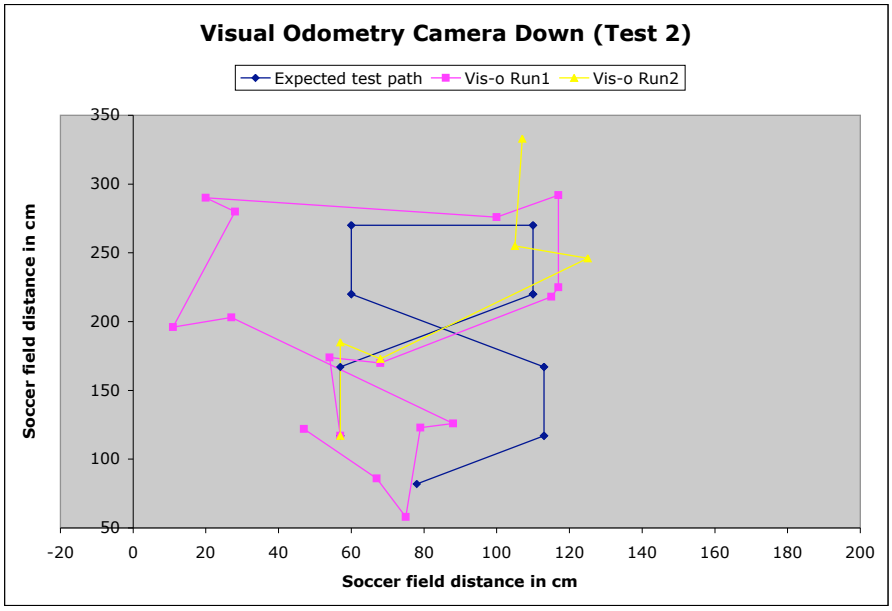
5.5.3 Test 2 Results: Figure eight path traversal of a soccer field

Figures 5.9 and 5.10 show the results obtained from running Test 2 on the VOCD, the VOCF, the KLT system and the shaft encoder system. All data in the test was determined and recorded by physically measuring the position and orientation of each robot at every measurement point.

During the execution of the test, it was discovered that the shaft encoder system did not perform well when turning. This poor performance is surprising, given that this system was a commercial robot.

The VOCD in Run 1 performed well even with the problem of not observing the field lines at all times. In Run 2, the VOCD did not perform well. The testing had to be stopped midway because the robot had driven off of the field and could not recover its position. This poor performance was due to limited field line observation and the fact that the platform's behaviour (motion) was not predictable. Overall, the VOCF performed poorly and the robot did not even remotely follow the figure eight course. This is due to the problems of having the camera in the forward looking position and the behaviour (motion) of this platform. These problems will be explained in Section 5.6.

The results of the KLT system did not reflect the motion of the robot and did not produce good ego-motion estimation during both Run 1 and Run 2. This was due to the problem of the robot moving too far between image captures (as already discussed at the beginning of this section). This problem will be discussed in detail in Section 5.6.



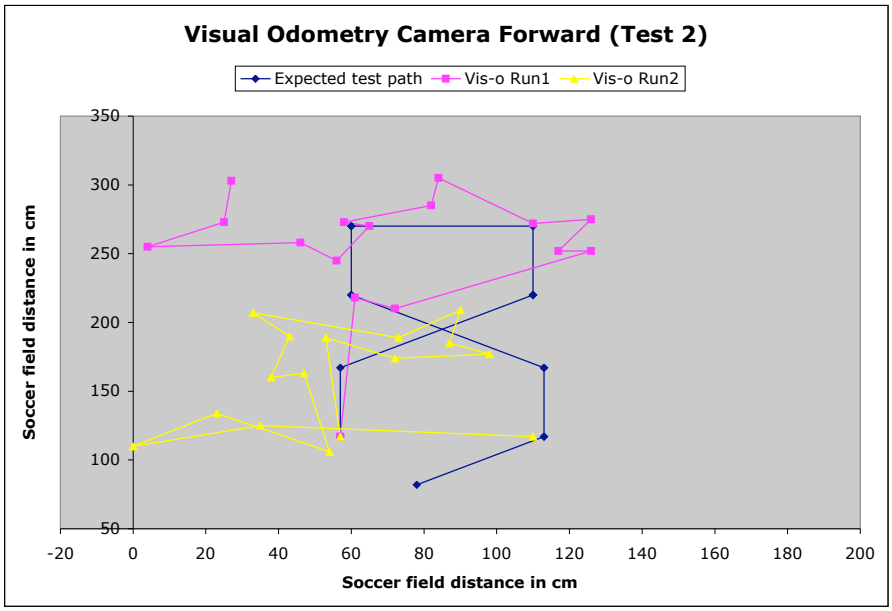
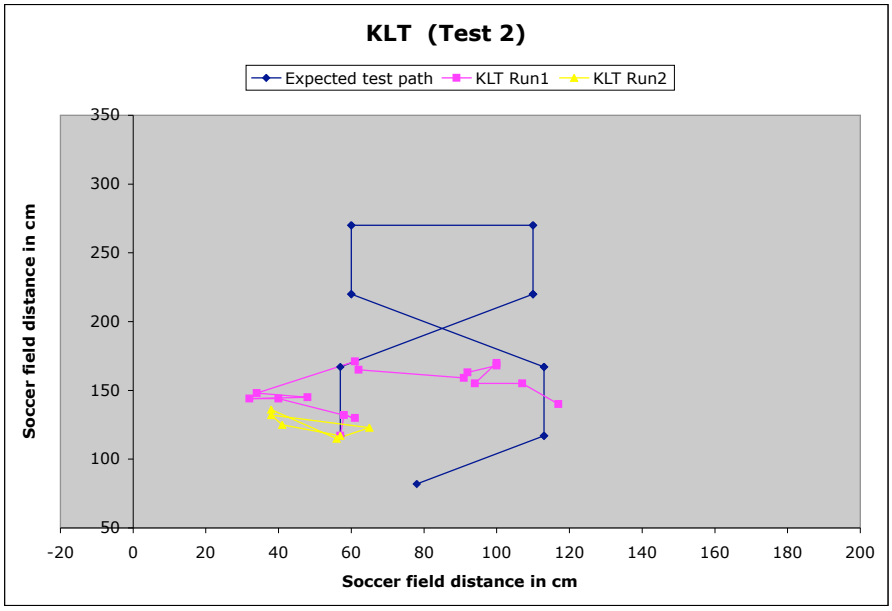


Figure 5.9: Test 2 visual odometry camera down and forward results.



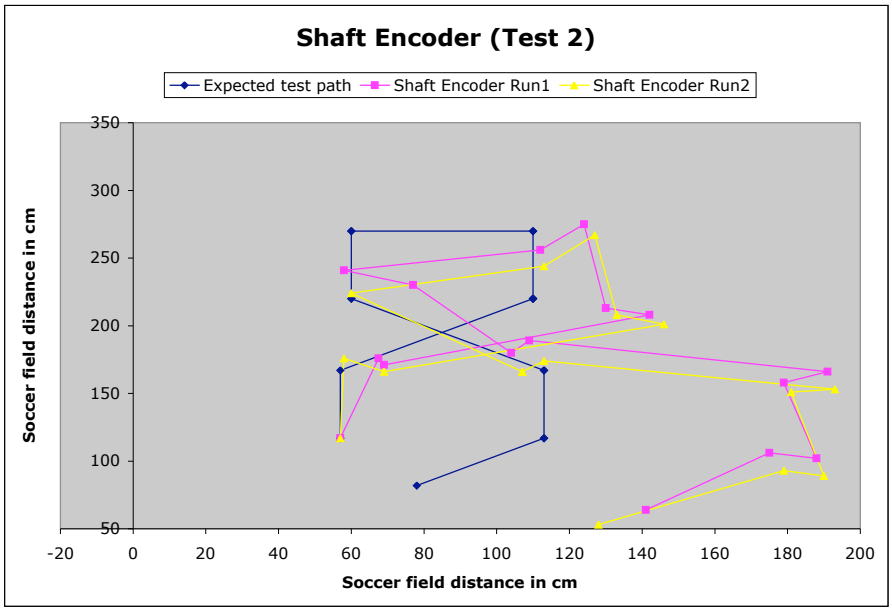


Figure 5.10: Test 2 KLT and Pioneer results.

5.6 Final Evaluation

This section will give a final evaluation of all of the systems tested, indicate which system performed best during the tests and give recommendations on how the designed visual odometry system could be improved to provide better overall results.

5.6.1 Visual Odometry and KLT System Short Comings

In Subsections 5.5.1, 5.5.2 and 5.5.3 it was found that during testing, the visual odometry system did not perform as initially thought. This was due to the problems of the camera position and unpredictable platform motion. These shortcomings will now be examined in more detail to try to explain the poor behaviour of the designed system

5.6.1.1 Camera Position

Initially the position with the camera pointing down was chosen because it was thought that allowing the robot to see the field directly in front of it would allow it to manoeuvre around any obstacles that it would encounter. The Robocup [3] soccer field was the domain of interest and the ability to see anything other than the field would not help the robot to determine its ego-motion. If the position of the camera had allowed the robot to see a far distance off of the field, the robot could have observed three dimensional objects that were not part of the ground plane (Z axis value not equal to zero). This would have meant that the camera calibration was invalid because the camera was calibrated for 2D surfaces (see Section 4.3). An invalid camera calibration would have resulted in an inaccurate conversion of the

detected line's image positions to their real world positions. This would have resulted in inaccurate ego-motion estimation.

During testing, it was discovered that having the camera pointing down resulted in the soccer field lines only being seen part of the time. The partial observation of the field lines meant that the line feedback could not be used at all times and that the robot could not update its motion model to more accurately represent its true motion. This had a negative effect on the robot because the majority of the time, it had to rely on motion model based dead reckoning. Because the platform had no other feedback source, the robot did not have any other method of measuring how far it had actually moved.

Two examples will now be introduced to explain the severity of this problem in more detail. Both of the following examples happened in all tests and test runs conducted in this thesis.

The first example occurred when the robot was moving forward as it traversed the soccer field. Initially the robot's motion model was set for the Move forward increment command to represent 2.5 cm of distance travelled. This meant that if no feedback was available after the robot was commanded to move forward, the robot would have thought that it had moved 2.5 cm. Using this statically initialized motion model actually increased the position error of the robot, given that this platform's motion is unpredictable (as explained later in Section 5.6.1.2). This was the best result that could be achieved because there were no other sources of position feedback. As the robot traversed the soccer field, it encountered the centre line. It could now see this field line and use the line's change in position over time to estimate its ego-motion and

update its motion model. This meant that the robot had traversed roughly half of the field before it had received any line feedback and that its position error grew each time it had moved. It was found at this point in the robot's traversal of the soccer field, that its real position was different than its commanded position. The robot could use the centre line for its line feedback for only a short time, as it continued its traversal of the soccer field, before this line moved outside of the camera's view. At this point, the robot now had to rely on its updated motion model to estimate how far it had moved after a command was issued. This also led to an incorrect ego-motion measurement given that the platform's motion was unpredictable (as explained later in Section 5.6.1.2) and that the line feedback was only available for a short period of time.

The second example occurred when the robot was turning around after it had traversed the soccer field. In this example the robot was commanded to turn 3.14 radians. Initially the robot's motion model was set for a Turn right increment command to represent a change in angle of -0.2 radians. As the robot turned, it only saw field lines intermittently or saw no lines at all. This depended on where the robot had ended up after its linear traversal of the field. Again this meant that the robot had to rely most of the time on its motion model to update its orientation. This resulted in a large orientation error because of the unpredictability of the platform's motion (as explained later in Section 5.6.1.2).

Using the information and results acquired from observing the robot's behaviour during the examples described above, it was realized that at least one field line must observed most of the time to take full advantage of the visual odometry algorithm.

To solve the problems associated with the camera pointing down, it was pointed in a more forward looking position, so that the robot could see the field lines most of the time.

During testing, the forward looking camera position was also found to have problems. When the camera was pointed in this orientation, the robot's field of view allowed it to see all of the field at once. This meant that if the robot was close to the end of the field, it would see a far distance off of the field. In this situation, the robot had the potential to observe 3D objects (tables, chairs, etc) that were not part of the ground plane. These 3D objects produced image lines that invalidated the calibration of the camera. The invalid camera calibration resulted in inaccurate image to real world coordinate conversions, which led to incorrect ego-motion measurements.

A second problem with the camera in the forward looking position occurred in the camera calibration. After the camera was calibrated in this position, each pixel in the upper part of the image represented a change in distance ranging from 1 cm to 6 cm. This meant that if the image or line data (due to quantization error) had an error of one pixel, the robot could think it had moved 6 cm or rotated by 40 degrees (measured via the ego-motion calculations) when no motion has occurred. In the worst case scenario, this error could be compounded resulting in an error much greater than 6 cm. In informal testing, errors of 10 cm or greater have been observed.

By performing all tests and test runs with the camera in the forward looking orientation, it was found that the problems associated with this orientation (as described above) led to the poor performance of the visual odometry system camera forward (VOCF). Two different methods were attempted to remedy this problem.

These methods were to use only the lower half of the image in calculating ego-motion and to point the camera facing down to see closer to the robot. Both of these methods resulted in the robot not being able to see the field at all times and yielded the same problems and results found with the visual odometry system camera down (VOCD).

5.6.1.2 Unpredictable Platform Behaviour

The platform used in this thesis was a custom built design as already described in Chapter 4. This platform utilized a single camera as its only sensor to measure its ego-motion when specific visual feedback was available. This meant the robot had no way of controlling its actual position and speed in the absence of the visual feedback. Using these constraints, the platform's drive system control software was designed and implemented. The control software was an open loop design that utilized a time based strategy to control each individual servo. This meant that a command of move forward would instruct each of the two servos to turn on for a specified amount of time.

The designed platform functioned adequately in informal testing and at the various robotic competitions. The problems associated with this platform were not noticed at this point because the robot could not be observed at all times during the competitions and most of the informal testing took place on a smooth surface with the robot's traversal limited to small distances.

After the start of the formal testing, the designed platform was found not to perform very well. While observing the motion of the robot during formal testing, it was found that each time a command was issued, a different amount of movement

or rotation took place. For example, some forward commands yielded motion that was very small (0.5 cm) and other forward commands yielded distances greater than 2.5 cm. The frequency of this difference in distances could not be predicted. The turning commands behaved the poorest on this platform and the range of motion was anywhere from less then 0.05 radians to greater than 0.3 radians. This meant that in the absence of feedback, the motion model was incorrect and that using it yielded very inaccurate ego-motion estimation.

In the presence of feedback, the lack of uniform motion was not a problem in controlling the robot because the robot would issue more motion commands to get the platform to move to the correct position setpoint. This lack of uniformity in the motion did negatively affect the motion model when feedback was used. For example, during a Run of Test 2, the robot was commanded to turn right by 1.57 radians and was using visual feedback to help it to complete its turn. After each turn increment, the robot had turned by only a very small angle (approximately 0.05 radians) resulting in the motion model being updated to more accurately reflect the motion of the robot. This was the exact behaviour that the system was supposed to exhibit. However, this did not work well for some of the other turns in the Test 2 Run because the motion model now modelled the robot turning a very small angle (0.05 radians) after a turn command was issued. At the next point in the test when the robot was commanded to turn right by 1.57 radians, the robot did not have any feedback, so the robot had to rely strictly on the motion model based dead reckoning to complete its turns. In this case, the robot's turns were actually 0.3 radians greater than the what the motion model had predicted. This meant the robot had exceeded

its setpoint by 1.57 radians by using only dead reckoning to complete its turn.

The platform unpredictability was a major hindrance on the performance of the visual odometry system. From observing the robot during all of its tests, the unpredictability for the most part was due to the timing of the motors and the speed which they are set. It is difficult to determine how long the motors are actually on, given that this is not a real-time operating system and each thread is not guaranteed to be executed every X number of milliseconds. It is also hard to determine how long it takes for the servo control board to receive and process the motor commands.

One other cause of the unpredictability was due to the construction of the robot's wheels. They were constructed out of very flexible plastic. It was observed that the entire robot was too heavy to be supported. This caused the robot's wheels to bend out, increasing the friction between the driving surface and the wheels. This bending of the wheels was negligible in the forward direction and it did not decrease the robot's performance. However, it was observed that the wheel bending increased significantly during turning. This caused the robot to turn less than it was commanded.

Stop, Move and Capture Effects On KLT

As explained in Chapter 4, the platform and the embedded system have evolved over the course of this research. In the first iteration of the platform with the initial Zaurus, the camera and the computing power limited the speed at which images could be captured and processed. The compact flash camera could not capture images very quickly (around one image per second) and the processing of each image took approximately 15 seconds depending on the quantity of data. These limitations made

it impossible to capture and process frames at a high rate. It was not possible to allow the robot to continuously move while it processed each new image. Therefore, it was decided that the robot would first capture an image, process it and then move a specified distance before the process would be repeated. This new control scheme worked well for the design of the visual odometry system. There were only a few lines on the soccer field and the motion of the robot was fairly small so the robot was able to track lines without many problems.

However for the KLT system, this caused a major problem (as already described at the beginning of Section 5.5). The problem was due to the fact that the robot moved too far between successive image captures. This caused the KLT system to misinterpret the features that it was tracking (feature aliasing) and the system was not able to track these features as they moved throughout the image. This resulted in incorrect ego-motion calculations for the KLT system, thereby causing the poor performance found in testing the KLT based ego-motion system. In order for the KLT system to be successful, large quantities of images needed to be taken during motion or the motion needed to be smaller between image captures.

5.6.2 Final System Evaluation And Recommendations

This subsection provides a final evaluation on the performance of each system and provides recommendations on what could be done to improve the performance of ego-motion estimation.

Final System Evaluation

All tests were executed twice for all systems except for the environmental debris test (EDT) which was executed only once in the Test 1 testbed. This was done in order to discover if the debris would negatively affect the systems' performance. It was found after the EDT run that the debris severely affected and limited the functionality of the visual odometry system and that it had no impact on the shaft encoder system. Therefore conducting more test runs in the current testbed would not lead to a better understanding of how the designed system would compensate for the introduced debris.

The test frequency of both Test 1 and Test 2 was chosen to discover the feasibility and obtain preliminary results of the designed visual odometry system. In practice, executing more test runs on physical robots is difficult and problematic. For example, battery life affects how long and how many tests can be performed. If the batteries are not closely monitored, they could run out of power before a test is completed. Another effect of the batteries is that as they are used, their power output diminishes. This will affect how fast the servos can turn, which will change the amount of distance travelled per system update. This introduces a situation that affects the test results as more tests are executed. This means that a test completed with fresh batteries

could yield different results than a test completed with depleted batteries.

The KLT system was not well suited for an embedded implementation in its current form because of its requirements for fast image processing and capture. This led to the conclusion that the KLT feature tracker in its current form is only good for off-line experiments and very slow moving mobile robots.

The performance of the visual odometry system showed that it had a lot of potential. It was able to successfully measure its ego-motion and move to different position setpoints when visual feedback was available. The problems with the position of the camera and the unpredictability of the platform motion revealed that more work and research needs to be conducted in order to discover the true capability of this system.

The shaft encoder system was based on a commercial robot and it was thought that its position error would be minimal over short distances. The system performed well in all tests but had problems estimating its position after a turn. The error in turning was not anticipated before the tests were conducted and this caused the robot to lose track of its actual position and orientation.

The position error of each system can be observed in Tables 5.1 and 5.2. These tables list the distance RMS position error of all systems at every test position. It is clear from analysing these tables that overall, the shaft encoder system had the smallest position error and that it performed the best overall.

Average distance RMS position error at each position in test 1

	VOCD	VOCF	KLT	Shaft Encoder
P0	0	0	0	0
P1	45.59	85.08	186.08	15.87
P2	15.63	106.23	189.8	19.93
P3	101.01	54.01	71.35	85.11
Sum of errors	**162.23**	**245.32**	**447.23**	**120.91**

Table 5.1: Distance RMS error at each position in Test 1. For all systems.

Average distance RMS position error at each position in test 2

	VOCD	VOCF	KLT	Shaft Encoder
P0	0	0	0	0
P1	12.81	36.76	39.98	11.44
P2	64.17	49.48	108.09	65.85
P3	17.7	40.21	105.13	37.44
P4	21.98	37.32	113.49	23.57
P5	43.08	40.48	145.82	16.06
P6	151.6	45.53	217.64	20.16
P7	160.65	55.49	187.8	37.53
P8	155.06	54.21	190.8	78.79
P9	141.3	58.47	148.23	65.53
P10	132.58	54.36	146.7	64.08
P11	124.83	90.74	107.34	79.61
P12	128.62	104.43	107.49	68.23
P13	116.42	144.54	102.57	79.21
P14	109.06	128.76	100.56	66.6
P15	81.89	137.12	91.53	61.66
Sum of errors	**1461.75**	**1077.9**	**1913.17**	**775.76**

Table 5.2: Distance RMS error at each position in Test 2. For all systems.

Recommendations

After developing the visual odometry system, performing the tests and conducting the system evaluations there are some recommendations that can be made.

The first is to make the motion of the visual odometry platform more predictable. This would entail the development of a repeatable motor controller and the development of stronger wheels. The new motor controller would reduce the amount of variability in the amount of time each motor is activated after a motion command has been issued. The use of stronger wheels on the platform would eliminate wheel bending and reduce friction during turning. These two changes would eliminate the unpredictability in the physical motion of the robot and would allow for the motion model to more accurately model and predict motion.

The second recommendation is to find an optimal camera position. This would allow the robot to see field lines for the majority of the time while keeping the real world distance values for each pixel in the image to be between 0.2 and 0.5 cm. If the robot could see the field for most or all of the time, it would be able to use visual ego-motion estimation thereby eliminating the need to use the dead reckoning based motion model. Keeping the distance of each pixel to the smallest value possible would reduce the position error introduced from image noise and or line data quantization error. This optimal camera position would increase the use of visual ego-motion estimation and reduce the error in the position estimation.

The third recommendation would be to use onboard hardware acceleration of the embedded system or to use an external digital signal processor to speed up image processing and mathematical calculations. The increase in speed of processing could

reduce the time needed to complete one iteration of the visual odometry and KLT calculations. This would allow the robot to move continuously while taking and processing images. The use of continuous movement would eliminate the problem of unpredictable platform motion. It would also allow the KLT system to be used in real-time, eliminating the feature aliasing problem associated with this system.

The final recommendation would be to combine all three systems tested in this thesis and utilize sensor fusion to come up with a robust ego-motion system. If successful, this robust ego-motion system could allow mobile robots to intelligently navigate themselves in real-world environments such as outdoor and Urban Search and Rescue domains.

Chapter 6

Conclusion

The purpose of this research was to develop an embedded visual odometry system that could be used on any mobile robot. I have developed a system that extracts common visual information found in images and allows a robot's ego-motion to be estimated. This system takes an image as input, extracts horizontal lines from the image and tracks these lines over time to calculate ego-motion.

In order to carry out this research, I have also developed a mobile robot platform that consists of the mechanical chassis, electromechanical components and embedded system. This platform has evolved over the course of the research and every iteration of the platform has incorporated changes that have improved the overall engineering design of the system.

The complete visual odometry system (including visual odometer and mobile platform) was tested in three different test set-ups. These tests were the autonomous traversal of a linear path with and without environmental debris and an autonomous traversal of a complex path (figure eight).

The results of these tests were compared to the the results obtained from performing the same tests on two separate systems that I implemented for this research. The two comparison systems were a commercial shaft-encoder based dead reckoning robot and the second was a KLT [1] based robot.

From the results obtained, it was found that the shaft-encoder based robot performed best overall, but still had significant error in the odometry estimation when it performed turns. The KLT [1] based robot did not perform well. It was discovered that in order for this type of system to be successful, the motion of the robot between successive image captures must be constrained to a small value. The visual odometry system's results were promising but failed to produce accurate results. This was due to the fact that in the initial downward facing camera position, the robot could only see the field lines a small fraction of the time and had to rely on pure non-feedback dead reckoning. To remedy this problem, the camera was put in a forward looking direction. This camera position allowed the robot to see the field lines at all times, but also introduced problems that degraded the system's performance. These problems were that 3D objects could be observed which would invalidate the camera's calibration and that the camera's calibration yielded pixel distance values (from 1 cm to 6 cm in the upper part of the image) that were susceptible to noise. The problems associated with this camera position led to ego-motion estimation with errors greater than 6 cm.

I am confident that using environmental lines that are found in images can aid in the overall estimation of robot ego-motion as evidenced by the preliminary test results from my research.

Bibliography

[1] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, August 1981, pp. 674–679.

[2] S. Schaerer, J. Baltes, and J. Anderson, "Practical ego-motion estimation for mobile robots," in *2004 IEEE Conference on Robotics, Automation and Mechatronics*, vol. 2, Singapore, December 2004, pp. 921– 926.

[3] (2004) Robocup Autonomous Robotic Soccer website. [Online]. Available: http://www.robocup.org

[4] (2004) Urban Search And Rescue website. [Online]. Available: http://robotarenas.nist.gov

[5] (2004) Darpa Grand Challenge website. [Online]. Available: http://www.darpa.mil/grandchallenge

[6] Z. Zhang, "Estimating motion and structure from correspondences between line segments between two perspective images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1129–1139, 1995.

[7] C. Taylor and D. Kriegman, "Structure and motion from line segments in multiple images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 11, pp. 1021–1032, 1995.

[8] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, 2001.

[9] O. Amidi, "An autonomous vision-guided helicopter," Ph.D. dissertation, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1996.

[10] R. Sim and G. Dudek, "Mobile robot localization from learned landmarks," in *IEEE/RJS conference on Intelligent Robots and Systems*, Victoria, BC, Canada, October 1998.

[11] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The International Journal of Robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[12] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Korea, May 2001, pp. 2051 – 2058.

[13] R. Sim, P. Elinas, M. Griffin, and J. J. Little, "Vision-based slam using the rao-blackwellised particle filter," in *IJCAI Workshop on Reasoning with Uncertainty in Robotics (RUR)*, Edinburgh, Scotland, July 2005.

[14] D. Herrero-Pérez, H. Martínez-Barberá, and A. Saffiotti, "Fuzzy self-localization

using natural features in the four-legged league," in *RoboCup 2004: Robot Soccer World Cup VIII*, ser. LNAI, D. Nardi, M. Riedmiller, and C. Sammut, Eds. Berlin, DE: Springer-Verlag, 2004.

[15] A. Milella and R. Siegwart, "Stereo-based ego-motion estimation using pixel tracking and iterative closest point." in *Proceedings of the Fourth IEEE International Conference on Computer Vision Systems*, New York City, USA, Jan 2006, p. 21.

[16] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," in *Proc. of the 2005 IEEE Conference on Systems, Man and Cybernetics*, Hawaii, USA, October 2005, pp. 54 – 62.

[17] J. Shi and C. Tomasi, "Good features to track," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, USA, June 1994, pp. 593 – 600.

[18] J. Campbell, R. Sukthankar, and I. Nourbakhsh, "Techniques for evaluating optical flow for visual odometry in extreme terrain," in *International Conference on Intelligent Robots and Systems (IROS)*, vol. 4, Sendai, Japan, September 2004, pp. 3704 – 3711.

[19] J. Campbell, R. Sukthankar, and I. Nourbakhsh, "Visual odometry using commodity optical flow." in *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*. Menlo Park, California: The AAAI Press, July 2004, pp. 1008–1009.

[20] J. Campbell, R. Sukthankar, and I. Nourbakhsh. (2004, August) Exploring vision as a basic navigational sensor. Campbell_VisNav_IRPOH2003.pdf. [Online]. Available: http://info.pittsburgh.intel-research.net/people/jasonc/

[21] (2004) Open Computer Vision Library website. [Online]. Available: http://www.intel.com/technology/computing/opencv/index.htm

[22] C. McCarthy and N. Barnes, "Performance of temporal filters for optical flow estimation in mobile robot corridor centring and visual odometry," in *Proceedings of the 2003 Australasian Conference on Robotics and Automation*, Brisbane, Australia, December 2003.

[23] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, vol. 1, Washington, DC, June 2004, pp. 652–659.

[24] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 756–777, 2004.

[25] K. Ali, C. Vanelli, J. Biesiadecki, M. Maimone, Y. Cheng, A. S. Martin, and J. Alexander, "Attitude and position estimation on the mars exploration rovers," in *Proc. of the 2005 IEEE Conference on Systems, Man and Cybernetics*, Hawaii, USA, October 2005, pp. 20 – 27.

[26] D. Helmick, Y. Cheng, D. Clouse, L. Matthies, and S. Roumeliotis, "Path following using visual odometry for a mars rover in high-slip environments," in *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, vol. 2, 2004, pp. 772–789.

[27] F. Iida, "Biologically inspired visual odometer for navigation of a flying robot," *Robotics and Autonomous Systems*, vol. 44, pp. 142–149, September 2003.

[28] S. Wilson, B. Lovell, A. Chang, and B. Masters, "Visual odometry for quantitative bronchoscopy using optical flow," in *APRS Workshop on Digital Image Computing*, Brisbane, Australia, February 2005, pp. 157–162.

[29] R. Gonzalez and R. Woods, *Digital Image Processing*, 2nd ed. Reading, MA: Addison-Wesley, 1992, pp. 414 – 428.

[30] P. Hough, "Machine analysis of bubble chamber pictures," in *International Conference on High Energy Accelerators and Instrumentation*, CERN, Geneva, September 1959, pp. 554 – 556.

[31] R. Duda and P. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[32] R. Tsai, "A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses," *Radiometry*, pp. 221–244, 1992.

[33] R. Wilson. (2005, May) Tsai Implementation. [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/rgw/www/TsaiCode.html

[34] B. Gerkey, R. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics*, Coimbra, Portugal, July 2003, pp. 317–323.

[35] S. Birchfield. (2005, July) KLT Implementation. [Online]. Available: http://www.ces.clemson.edu/~stb/klt/