University of Alberta

AUTONOMOUS OUTDOOR NAVIGATION AND GOAL FINDING

by

James Neufeld ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2008

Your file Votre référence
ISBN: 978-0-494-47375-7
Our file Notre référence
ISBN: 978-0-494-47375-7

Abstract

The problem of autonomous robotic geocaching, which involves locating a goal object in an unstructured outdoor environment given only its rough GPS position, is an interesting research challenge. The potential benefits of this research include applications to agriculture, search and rescue, surveying, transportation, and interstellar exploration. There are a number of attributes of the autonomous geocaching problem, which encompasses the problems of unstructured outdoor navigation and goal finding, that make it both interesting and challenging. These include the countless outdoor hazards which make obstacle detection difficult; large and interconnected obstacles inherent in outdoor environments (e.g. ditches, dense forest, buildings) which necessitate frequent backtracking and sophisticated path-planning; and the lack of additional information, such as satellite maps, GPS waypoints, or obstacle descriptions, available to the robot. These difficulties have limited previous work in outdoor navigation to the development of systems that rely on structural cues (e.g. road, paths, manually colored obstacles) to aid both obstacle detection and goal finding. This reliance limits these systems to a narrow range of environments.

This thesis presents a fully autonomous robotic system capable of solving the task of geocaching. The final robotic system, named "Kato", was constructed by outfitting a self-balancing Segway robotics platform with various sensors and computing resources. These sensors included a GPS receiver and inertial sensor for position estimation, a laser rangefinder for obstacle detection, and a camera for goal identification. The key challenges in solving the geocaching task include: constructing a map of the terrain as the robot navigates, using this map for planning and following a path to the GPS coordinate, and finally searching the local area for the goal object. The effectiveness of the system was tested in two different outdoor environments, and its performance was compared to that of a human expert teleoperating the robot. Results indicated that, in the test environments, the robotic system was able to navigate to, and detect, the goal object with a high level of dependability. Moreover, path planning and navigational (maximum speed) efficiency and obstacle avoidance was similar when the system was operated autonomously to when it was teleoperated by a human expert. Overall, this work makes strides towards the development of a cost-effective robotic system that can effectively operate in challenging real-world environments.

Acknowledgements

I would like to, first of all, thank the AICML and RLAI staff members Jason Roberts, Mike Sokolski, and Steven Walsh for their many hours of work helping to develop the Kato robotic system. Also, a thank you to the many graduate students that have contributed to this project including Adam Milstein, Cam Upright, Brad Joyce, and the class of CMPUT 608. I thank Dr. Rich Sutton for his valuable advice and support throughout my MSc. work and of course my supervisor Dr. Michael Bowling for his brilliant advice and outstanding devotion his students.

Finally, I would like to thank my family for their support throughout my thesis work and I am particularly grateful for my fiance Thea's, and my sister Stephanie's, help in editing this thesis document.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem

Producing an autonomous robot capable of navigating in an outdoor unstructured environment is a challenging research task. In this context, *unstructured* implies that there are no added elements that make it easier for an autonomous robot to detect obstacles or find a path to its goal (i.e. no roads, obvious obstacles, or waypoints). This property, in combination with the various types of outdoor hazards, is primarily what makes the unstructured outdoor navigation problem challenging. The major aspects of this problem are: detecting obstacles from sensors data, planning a path to the goal while avoiding these obstacles, and finally, following the path efficiently.

1.2 Applications

Autonomous robotic systems capable of safe and efficient navigation in complex domains have a wide variety of potential applications. These applications include: autonomous crop harvesters, mining trucks, transport vehicles, surveying robots, search and rescue drones, and planetary rovers. In many cases, autonomous robots can also be used in place of a human operator to execute tasks at a lower cost, as robotic systems do not require daily wages or other worker benefits. Another invaluable application is the ability for robotic systems to operate in domains that are not safe for humans, such as battlefields, toxic environments, or outer space.

1.3 Previous Work

Despite the potential benefits autonomous navigation systems offer, there is still significant research required before these applications can be realized. Due to the difficulty of operating in an unknown outdoor environment, past research has typically focused on constructing systems that operate in constrained, or *structured*, environments. An example of a structured navigation task is the DARPA Grand Challenge [5], which was a racing competition where waypoints were provided and a road or path was known to exist between such waypoints. Although impressive advancements have been made in such environments, these methods rarely extend to domains where the assumed structure does not exist. The lack of transferability occurs because the environmental cues that these robots depend on are either not frequent enough in the unstructured environment, or are combined with new and undetectable hazards. However, despite the failure of robots designed for structured environments to operate successfully in unstructured environments, many of the isolated components developed for these systems can be directly or indirectly applied to systems designed for unstructured environments.

Although research into autonomous navigation for unstructured outdoor environments has been limited, there have been some notable advancements in the field. However, such work is of limited value as it is not cost-effective due to the physical components required, which are difficult to reproduce because of labour and monetary constraints. An excellent example of this is the CMU PerceptOR robot which relied on numerous complex and expensive sensors, as well as an autonomous helicopter drone, to extend sensor range [21]. While this particular system has demonstrated an impressive ability to tackle difficult outdoor terrain [23] the aforementioned costs related to this system prevent the technology from being applied in many scenarios.

1.4 Thesis Summary

Currently there is need for a simple, cost-effective, robotic system capable of robust navigation in unstructured outdoor environments. A major component of this thesis involved the design of a system that fills this niche. An additional component of this thesis addresses the previous unstudied problem of vision-based goal finding. This task involves reasoning about the possible goal location, producing a search path that maximizes the chance of finding this goal, and identifying the target object from camera images. The culmination of this thesis work, therefore, involved developing a system ("Kato") for solving the entire problem of autonomous *geocaching*[32], which essentially involves locating an object hidden in an unknown environment when given only a rough GPS coordinate.

1.4.1 Contributions

This thesis work contributes to the overall outdoor autonomous robotics research field by developing methods for 1) summarizing laser rangefinder data into a map representation that can be used for identifying obstacles in an outdoor environment; 2) estimating the position and orientation of a mobile robot using both position and terrain sensor information; 3) efficiently computing a high-resolution path to a long distance goal location; 4) producing smooth robotic motion along this path through a challenging environment; and 5) locally searching for, and visually identifying, a target object. Additionally, this thesis presents evidence that these methods in combination with the unique, and relatively inexpensive, sensor configuration and robotic apparatus are suitable for this outdoor navigation task. Given the significant costs in evaluating complex robotic systems in outdoor environments this is a significant contribution. This work is also presented in a concise conference publication [29].

1.5 Outline

This thesis document is organized into chapters, three of which address the three key issues of autonomous geocaching: perception, navigation, and local search. Chapter 4 (Perception) details how the robot estimates its current position and produces a map of the terrain from sensory data as it moves through the environment. Chapter 5 (Navigation) explains how this map is used to first plan a path to the target GPS location and then produce control commands that will cause the robot to follow this path efficiently. In Chapter 6 (Local Search) the final search component, which involves determining what areas to search, navigating to those areas, and visually identifying the goal object, is described. In addition to these chapters, Chapter 2 outlines the number of different methodologies and robot configurations that have been applied to the outdoor navigation problem in recent times. Chapter 3 details the specific hardware configuration of this robot including the robotic apparatus and sensors used. Also, experimental results for the developed geocaching system are presented in Chapter 7. Finally, in Chapter 8 the major limitations of the system are discussed and the thesis work is concluded.

Chapter 2

Background

In this chapter, the current state of research in outdoor autonomous navigation is outlined. A significant focus is placed on the systems developed for recent robotics challenges, which, because of their success in attracting significant monetary and research investments, showcase the most popular methods for autonomous outdoor navigation. Additionally, the competitive nature of robot competitions helps ensure the testing procedures are rigorous and well controlled. The different choices for robotics platforms are detailed as are the different methods for three key aspects of autonomous navigation: localization, mapping, and path-planning.

2.1 Major DARPA Projects

In the last several years the Defense Advanced Research Projects Agency (DARPA) has funded three major outdoor robotics challenges: the 2004 and 2005 Grand Challenges [7, 8], the PerceptOR program from 2000 until 2003 [23], and the Learning Applied to Ground Robots (LAGR) program started in 2005 and currently ongoing [9]. Figure 2.1 shows robots from each of these three programs. These challenges have not only spawned a great deal of research in the field, but have also employed rigorous testing procedures. This is an important aspect because its is difficult to evaluate and compare outdoor mobile robots due to the time and monetary costs involved and, consequently, many works lack such rigorous testing.

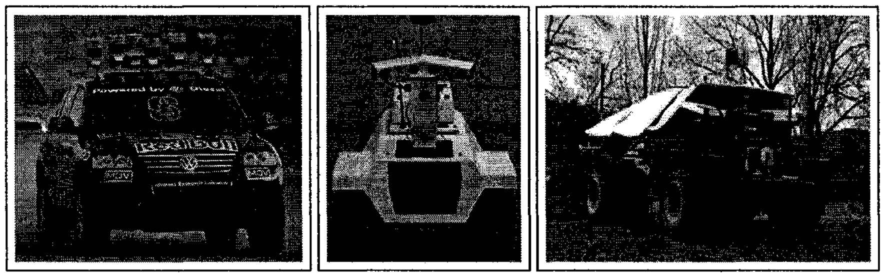


Figure 2.1: On the left is an image of "Stanley" winner of the 2005 Darpa Grand Challenge [41], a LAGR robot is shown in the middle image [6], and on the right is CMU's PerceptOR robot [21]. Images courtesy of DARPA.

2.1.1 Grand Challenges I & II

The DARPA Grand Challenge (DGC) was a racing competition for autonomous systems over rough desert roads. The objective of this challenge was to design a fully autonomous vehicle that could navigate, at high speeds, over several hundred kilometers of rough desert track using a platform and sensors of the teams choice. The track was provided to the competitors in the form of several thousand GPS waypoints and consisted mainly of structured desert roads. The first competition featured a first place prize of \$1M and took place March 13, 2004. However, this race was not successful as none of the 15 contestants were able to complete more than 5% of the 229 km long course. The challenge was repeated on October 8 of the following year, and this time 23 participants were selected to compete for the first place prize of \$2M. Of these 23 robots, 5 were able to successfully finish the 212 km course. The first place finisher was Stanford's "Stanley" robot [41] (shown in Figure 2.1) followed by CMU's two entries: "Sandstorm" and "H1ghlander" [44], Gray Insurance Company's entry "KAT-5" [43], and finally by the Oshkosh "TerraMax" vehicle [3]. Because of their success these five systems serve as excellent examples of the current state of research in the field of autonomous outdoor navigation.

2.1.2 PerceptOR

The DARPA PerceptOR program was a smaller scale project than the DGC; there was no prize money and only 3 teams participated. Nevertheless, this work is highly related to this thesis as it focused on navigation through undeveloped areas and competitors were not racing along a structured course, although they were still provided with multiple GPS waypoints. Additionally, the robots competing in this challenge were subjected to extensive testing that took place over 130 km of terrain. Given the high cost of experimenting with mobile robots this level of evaluation is rare and thus provides useful insight into the effectiveness of various navigation techniques. CMU's PerceptOR entry is shown in the rightmost image in Figure 2.1.

2.1.3 LAGR

The LAGR project is an outdoor navigation challenge that aimed to foster research in machine learning techniques for visual robot navigation. This project started in 2005 and is currently ongoing with 8 active competitors. What is unique about this competition is that the robotic apparatus is fixed for all teams and only includes two binocular vision units, infrared cameras, and a bump sensor. The middle image in Figure 2.1 shows the LAGR robot. Due to the constrained sensor suite, the LAGR robots have not yet progressed to the point where they can operate effectively in unstructured outdoor domains. For this reason the test courses for this project often include a significant amount of added structure to aid visual navigation. However, the current lack of effective vision based navigation systems, combined with the high potential of using cameras for navigation, indicate that the LAGR project has the potential of producing highly applicable technology.

2.2 Robotic Platforms

A number of different robotic platforms have been used successfully in outdoor navigation tasks. Such platforms are used to provide both mobility and carrying capacity for robotic systems. These platforms differ in size, carrying capacity, speed, acceleration, ruggedness, and maneuverability. Due to this level of diversity, the choice of platform is usually dependent on the characteristics of the task being solved. For example, a racing task like the DGC will require a significantly different robotic platform than a search and rescue task.

Most teams in the DGC elected to use production level (i.e. mass manufactured) vehicles as a primary platform [5]; as did the CMU's NAVLAB II system [40]. This is mainly due to the fact that the kingpin steering mechanism, that all these vehicles share, is well suited for high speed applications. Additionally, these systems have been in common usage for over 75 years and thus are relatively low cost, readily available, and dependable. Also, all three of the robots that competed in the DARPA PerceptOR challenge used production level all terrain vehicles (ATVs) as the primary platform [23]. These ATVs were chosen because of their significant carrying capacity, availability, rugged engineering, and dependability. However, these platforms also use a kingpin steering mechanism which significantly limits the maneuverability of the robot.

Differential drive systems are more commonly utilized for smaller scale applications as they provide increased maneuverability. These smaller and more maneuverable robots are often able to navigate to locations where a larger system would not fit. Additionally, the ability of these systems to turn in place makes them much easier to control autonomously. The Segway RMP uses a differential drive [35] as does the DARPA LAGR robot [9].

In environments where ruggedness is a critical factor, tracked robots are often used. Tracked robots can navigate rough terrain more effectively because they have more drive area in contact with the ground at any given time, which acts to increase traction and prevent the robot from sinking in soft ground. In order for a tracked robot to turn, an approach similar to a differential drive is used. However, the extended length of the tracks cause them to skid, making it difficult to estimate the position of the robot from odometry encoders, as well as requiring more energy. Researchers at the Jet Propulsion Laboratory in California have recently designed a tracked robot, "Urbie", for autonomous outdoor navigation [26].

Some other, less commonly used, robotic platforms include legged robots such as CMU's "Ambler" robot [36], which is designed for extremely rough terrain, or NASA's "Snakebot" [28] which uses a biologically inspired slithering locomotion to navigate in challenging environments. Although, these platforms are capable of navigating in very rough terrain their low top speed and high development costs have prevented them from being widely used.

2.3 Localization

In order to fuse sensory information together over time, as well as plan a path to the goal, it is essential that an autonomous robot maintain an estimate of its position and orientation. In the robotics literature this problem is referred to as the localization problem [42]. This is a very active area of research in mobile robotics, although less so in outdoor applications since the introduction of GPS. This section details some of the more commonly used sensors and methods for measuring the position or motion of a mobile robot.

2.3.1 Common Localization Sensors

Odometry Encoders

Odometry encoders are used to measure the change in a robot's wheel rotation which can be combined with the measured wheel diameter to estimate motion of the robot. The technology behind these sensors is relatively simple and therefore they tend to be cost-effective, accurate, and reliable. For these reasons, most outdoor robotic systems use odometry readings in some capacity. However, while these sensors return very accurate wheel rotations, external factors, such as wheel slippage, can make it difficult to accurately determine the true robot motion.

Inertial sensors

Inertial sensors, sensors often employed in autonomous robotic systems, are used to measure accelerations in 3D position and velocities in 3D rotation. These sensors often combine information from both gyroscopes and accelerometers to produce these measures. Due to the complexity of these devices, accurate inertial sensors are relatively expensive but provide more accurate short range position information than any other sensor that is currently available. A position estimate can be computed by doubly integrating the measured accelerations over time. While the orientation of the sensor can be maintained by integrating the measured rotational velocities over time. However due to this constant integration, like the odometry estimate, this estimate is prone to accumulating errors.

Global Sensors

Global Positioning System (GPS) sensors and magnetic compasses are position sensors that are limited to outdoor applications. What is unique about these sensors is that their measurements do not accumulate error as the sensor moves. This has helped set the field of outdoor navigation apart from indoor navigation, at least in terms of how the localization problem is solved. GPS sensors triangulate their position using satellite signals and are typically accurate to 8-20 meters, depending on the features of the environment. Also, the use of GPS sensors in outdoor robotics has increased substantially since the USA Air Force allowed civilian devices to receive a non-degraded signal in year 2000 [31]. Magnetic compasses, on the other hand, use the earth's magnetic field to calculate a heading, and therefore the accuracy of these sensors depends heavily on any interfering magnetic fields. Also, since GPS is effective at calculating headings, many systems do not require the use of magnetic compasses.

2.3.2 Dead Reckoning

Dead reckoning is the simplest form of robot localization and refers to pose estimation based entirely on sensor information internal to the robot. Of the previously listed position sensors, only odometry encoders and inertial sensors qualify as "internal to the robot". The main factor regarding a dead reckoning localization approach is the accumulation of error in the position estimate.

Because of the long traversal distances and high odometry error typical of outdoor robotics tasks, this method of localization is rarely used. Also, GPS sensors and compasses have essentially made the dead reckoning approach obsolete for all but a narrow class of outdoor environments. Interestingly, the CMU NAVLAB II robot [40] was developed before these sensors became widely available and therefore relied on the dead reckoning approach for localization. The designers of this system used odometry encoders and an inertial sensor to produce a "two-dimensional position estimate accurate to roughly 1% distance traveled and a heading estimate accurate to a few degrees per hour" [40]. To account for this accumulating error, this system was forced to include several modifications to the mapping and planning subsystems.

2.3.3 Reference-Based Localization

Reference-based localization systems eliminate accumulating position errors by referencing the external environment. Because this advantage comes with little to no cost, almost all of the more recent outdoor navigation systems use this form of localization. Commonly used external references include: GPS signals, compass readings, known landmarks, and local terrain measurements. Also, some robotics tasks constrain the availability of these sensors to advance research in specific domains. For example, planetary rovers do not usually include a GPS sensor because GPS signals are only available on our planet. The localization system designed for NASA/JPL's "FIDO" rover, for example, uses the position of the sun as an external reference to reduce accumulating position error [2].

However, since the majority of outdoor robotic applications take place where GPS signals are available, most systems elect to use them. Two effective methods of robotic localization using GPS signals in combination with other localization sensors, are provided below. The first technique, employed by CMU's PerceptOR robot [21], maintains both a local and a global position estimate. This system employs dead reckoning (using odometry and inertial sensor measurements) to maintain the local position estimate. This local estimate is used to determine the position of the laser sensors so they can be used to construct a locally correct obstacle map. The global position is estimated directly from the GPS sensor measurements and is used to produce a local position for the supplied global waypoints that mark the desired path. The advantages of this approach are that it is straightforward to implement and large jumps in the GPS position do not introduce artifacts into the terrain map. One limitation of this approach is that the robot's global position cannot be estimated to a greater degree of accuracy than what the GPS sensor provides.

Another, more popular, approach to robot localization in an outdoor environment involves combining redundant sensor readings using a Kalman Filter (KF) [10]. There are hundreds of technical documents on how KFs can be used to combine sensor readings for mobile robots and the reader is therefore referred to the standard text in the field [42] for a comprehensive overview of various KF techniques. In short, the technique combines redundant measurements from multiple sensors to produce a position estimate that is more accurate than one produced by any sensor in isolation. Also, there are a number of widely available off-the-shelf position sensors that use a Kalman filter to combine GPS, odometry, and inertial measurements in hardware. Such devices were used by the majority of the final contestants in the 2005 DGC [5]. Kalman filters, however, take a substantial amount of tuning before they work properly and a naive implementation would be highly sensitive to GPS position jumps [41].

2.4 Environmental Mapping

All outdoor navigation systems have some method for sensing terrain and labeling it with a cost measure. The cost measure is used to avoid terrain that poses a hazard to the robot or slows traversal time. Since non-airborne mobile robots cannot alter their elevation these cost measures are usually projected onto a flat 2D map that is sent off to a planning subsystem to produce a path. This section explains the various ways in which this is done.

2.4.1 Common Terrain Sensors

Laser Rangefinder Sensors

Of the available terrain sensors, laser rangefinder (or LIDAR) sensors are the most commonly used for high performance outdoor navigation applications. This is evidenced by the fact that they were used as the primary terrain sensor by all of the final contestants in the DARPA Grand Challenges [7, 5], and the DARPA PerceptOR program [23]. Laser rangefinder sensors measure the distance to objects by rapidly projecting laser pulses and timing their return. Because each laser pulse provides only a single depth measurement, these sensors are usually designed to send out pulses in a sweeping motion using a rotating mirror. Many systems use multiple laser sensors in combination to increase the small field of view typical of laser sensors. These sensors can be mounted rigidly at different angles in a "push-broom" configuration like the 5 laser rangefinders mounted on the "Stanley" DGC robot visible in Figure 2.1. Also, Pan-tilt or rotating mounts are commonly used to increase the viewable area; for example, the CMU PreceptOR robot [21] has a single laser sensor on a continually rotating mount above the robot and another on a mount that continually pans left to right. The widespread use of laser rangefinders is largely due to the fact that they return highly accurate terrain measurements that do not require sophisticated techniques to interpret.

Cameras

After laser rangefinder sensors, cameras are the next most commonly used terrain sensor in outdoor navigation applications. Providing there is enough ambient light in the environment, cameras are a rich source of terrain information. However, unlike a laser sensor, camera information can be difficult to interpret and often necessitates the use of sophisticated machine vision methods. The high potential of these sensors, once machine vision methods are refined, as well as low cost and small size, contribute to their appeal for outdoor robot navigation tasks.

Cameras can be used to measure terrain geometry in a variety of ways. Depth from stereo is the most widely used method for measuring terrain geometry because of its improved robustness and viewable area compared to other vision-based approaches. The DARPA LAGR robots [9] combine the depth from stereo readings from two separate binocular stereo rigs to increase robustness even further. Trinocular vision systems are also a common off-the-shelf approach to extracting geometry from camera images. The TerraMax robot that finished fifth in the 2005 DGC, for instance, used three trinocular camera rigs as part of the mapping system [3]. Computer vision techniques such as shape from shading, structure from motion, range from focus, and range from texture can also be used to determine terrain geometry [17, 13]. However, these techniques do not typically have the robustness necessary to be used as a primary method for terrain measurements. Research investigating the application of machine learning techniques to extract distance from monocular images using texture features has also been done [34]. This approach was successfully implemented on a small scale mobile robot designed to navigate at high speeds in a somewhat structured outdoor environment; a monocular camera was the sole terrain sensor [27].

Radar

Radar sensors do not provide very accurate or dependable terrain measurements and for that reason are usually used as a secondary terrain sensor, if they are used at all. The robotic systems that have used radar sensors generally did so to detect very large obstacles at a range greater than available for a laser sensor or to penetrate terrain artifacts, like dust [44].

Short Range Sensors

Many robotic systems employ short range sensors to detect obstacles that have been missed by the primary terrain sensor. These sensors are not typically used as a primary sensor because their short range would limit the top speed of the robot. Commonly used short range sensors include bump sensors, infrared camera sensors, sonar sensors, or motor torque sensors. Because most of these sensors are cost-effective and have low power consumption, they can be placed strategically around the robot to detect obstacles right before they are contacted.

2.4.2 Geometry-Based Mapping

Geometrically-based classifiers determine the cost of the terrain by considering its geometric properties. These methods require the use of a sensor that is capable of extracting this geometry, common sensor choices include laser rangefinders, cameras, sonar, and radar sensors. Many systems combine information from multiple sensors to produce a more accurate or more dense terrain map [44]; this approach is known as sensor fusion [19].

Geometry-Based Classifications

In outdoor robotic systems, estimating height changes in the load bearing surface is the most widely used method for determining the cost of a given area. The long standing applicability of this approach has lead to the development of high level formalizations for multiple sensor types [20]. Under these formulations the height of the load bearing surface is represented by a discretized elevation map, referred to as a 2.5D representation. Under this representation, obstacles can be easily, and efficiently, detected by comparing the height differences between neighbouring cells. However, difficulties arise in the determination of the true load bearing surface from sensor depth readings.

There are no fool-proof methods for accurately estimating the height of the load bearing surface; therefore the majority of systems assume the sensor depth estimates lie directly on the surface. For highly vegetated terrain this approach is ineffective as the system will likely describe all of the terrain as unsafe and will declare that there is no path to the goal. An alternative approach, which was developed for use with the CMU autonomous tractor system [46, 45], accounts for this through the use of a machine learned terrain classifier which is able estimate the true load bearing surface from laser rangefinder data. The key property that this approach exploits is that vegetation will occasionally allow laser scans to pass through whereas the load bearing surface typically will not.

Geometric terrain features can also be used to directly determine a cost for a section of terrain without making any assumptions about the load bearing surface. The Applied Perception team in the DARPA LAGR program developed a neural network classifier to determine cost from specific terrain features [15]. This classifier inputs a set of 8 different easily calculated geometric features (e.g. the mean, variance, max, and min of sensory measurements), into a neural network model which was tuned offline with a large set of previously obtained training data. Alternatively, the "Urbie" robot used a hand designed rule-based classifier to create a function from geometric features to cost [26]. These approaches arguably reduce approximation errors by avoiding the intermediate step of converting into a 2.5D representation.

2.4.3 Image Feature Based Mapping

A number of methods have also been developed for extracting terrain cost values directly from images without measuring the geometry. Because image data is difficult to interpret, the more successful approaches have used machine learning techniques to develop the cost function. For example, the DGC winning robot, "Stanley", used an online self-supervised learning approach to identify the surface of the road in camera images [24]. This relatively new learning procedure, also known as sensor bootstrapping, uses recent sensor data to provide labeled training examples for use with another sensor. In the case of the Stanley robot, laser rangefinder data was used to provide recent labellings (safe or unsafe) to parts of the current forward facing camera image. This learning approach allowed the simple color-based classifier to adapt very quickly to changes in the color of the road surface which compensated for its inability to fully capture the complexities of the relationship between camera images and terrain safety. Essentially, this approach used the camera to extend the range of the laser sensor by exploiting the fact that the road surface was likely to remain a consistent color in the short term. This allowed the robot to safely travel at higher speeds than it was able to with laser data alone.

Other systems have employed the self-supervised learning scheme to develop image-based terrain classifiers with a focus on short term accuracy. One approach used the robot's local terrain sensors to label examples in widely available satellite images [37]. These images were subsequently used to recommend terrain costs over a substantially larger distance, thus shortening the overall length of the robot's path. This approach has been effectively applied to the far field vision classification subsystems in the LAGR robots [15, 33, 11].

2.5 Planning

Robotic systems generally divide the planning problem into two components: the global planning stage, which usually works at a lower resolution over larger distances, and the local planning stage, which takes smaller scale robot motion into account. However, some tasks, such as the DGC race, do not require a global planning system because GPS waypoints are provided beforehand. For this reason the path planning algorithms designed for these systems are focused on generating smooth, precise, short distance paths.

2.5.1 Global Planning

The most common global path-planning algorithm for mobile robots is Dynamic A* (or D*) [39]. The continued popularity of this approach is likely because it was specifically designed to take advantage of the unique characteristics of the robot navigation problem in unknown terrain. The approach works by maintaining a cost-to-goal map, which requires only local updates as new sensor data is received. The CMU NAVLAB II robot [40] used the original D* search algorithm as a global planner as do most of the LAGR robots. Also, since the original D* publications, several new adaptations have been introduced to improve the algorithm for specific scenarios. Field-D* was introduced to alleviate the sub optimalities introduced by discretizing the search space [12]. Also, the D* Lite algorithm was introduced to improve the computational efficiency of the original algorithm as well as simplify the implementation [22].

An alternative approach to the D* search algorithm is to use a forward planner in a variable resolution grid like a quad tree. This approach has been successfully employed on a retrofitted autonomous jeep in an outdoor environment [47].

2.5.2 Local Planning

In order to take smooth and efficient paths around small obstacles, a robotic system must account for the complexities in the robot's dynamics. For complex steering configurations, such as the kingpin steering mechanism, this can be very challenging. Given the computational restrictions inherent in autonomous robotic systems, approximations to these dynamics are typically employed. The approach taken by CMU's NAVLAB, PreceptOR robot, and Stanford's Stanley robot was to preset several smooth robot trajectories ahead of time and choose between them at run time [41, 21]. These trajectories are chosen by first following along each one and filtering out any that intersect an obstacle. The system selects from the remaining trajectories by choosing the one that results in the robot proximity to the goal being minimized after execution. The chosen trajectory is followed using either pre-programmed control commands or by using a simplified trajectory following controller.

Alternatively, some systems plan a path ignoring the dynamics of the robot and use a path-smoothing method to improve the efficiency. Some common path smoothing techniques include spline-fitting [41] and waypoint following [6].

2.6 Summary

The field of autonomous outdoor navigation has begun to develop a reasonably well rounded body of research. Through the recent DARPA competitions several system have been developed for navigation at high speeds, through dense forest, and using primarily vision. The trends in these competitions are useful in determining which technologies have been used effectively and which ones will likely be effective in the future. For instance, the current dominance of the laser finder in high performance systems is clear, however, the camera has become increasingly more used. As more sophisticated computer vision and machine learning methods are developed, this trend is likely to continue. Also, the focus on structured environments is likely to turn to unstructured environments as more effective navigation methods are developed.

Chapter 3

Hardware

This section discusses the physical components of the Kato geocaching system. This includes the underlying robotic platform that provides both sensing, carrying capacity, and mobility to the system as well as the various sensors used for both measuring the robot's movement and sensing the terrain. These include GPS, inertial, camera, and laser rangefinder sensors. Protocols used for communicating with the main platform and sensors, and the means of powering them, are also discussed. The final portion of this chapter details the computing systems used to process sensor information and ultimately send drive commands to the robot.

3.1 Segway RMP

The Segway Robotics Mobility Platform (RMP) is the main underlying platform for the system. This platform consists of a drive system, control box, internal sensors, and aluminum chassis. What is possibly most interesting about this platform is that it has only two wheels and thus stays upright by continually using drive forces to stay balanced. To maintain its balance the RMP drives either forward or backward so that the bottom of the robot will stay under the center of mass. The embedded software in the RMP's control box exclusively maintains the robot's balance so there is no need to write custom software. Interestingly, the platform uses this continuous balancing strategy to move the robot forward and backward as instructed. This is accomplished by first allowing the center of mass to tilt a certain level in the desired direction. The robot then drives the wheels in the tilt direction to maintain its balance which moves the entire platform. The RMP has a top speed of roughly 4.0 m/s, can accelerate at 2.0 m/s^2, can carry approximately 60kg, and has enough battery to sustain up to 4 hours of continuous navigation [35].

This two wheeled balancing approach does however limit the stopping speed of the robot since it must accelerate in the direction of travel to recover the tilt before it can stop. Also, a balancing robot is much more prone to unrecoverable crashes as hitting an obstacle can cause it to lose its balance and fall over. However, the embedded software on the RMP is surprisingly effective at maintaining the robot's balance when it is acted on by external forces. As a result, the platform is

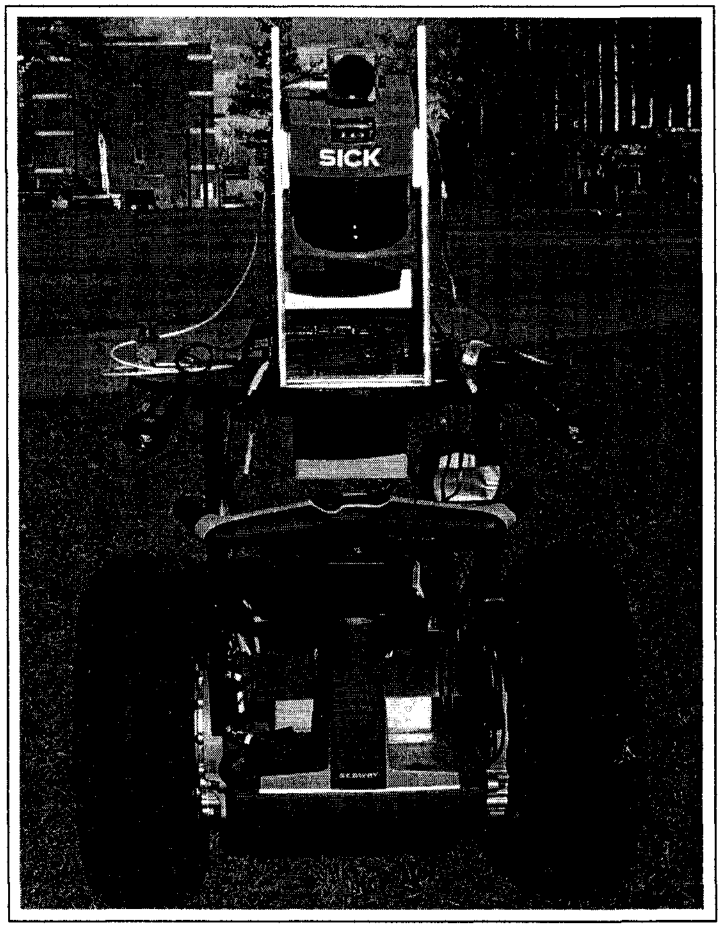


Figure 3.1: A photograph of the Kato geocaching system.

capable of handling most collisions with solid objects robustly, without losing its balance or having to make aggressive corrections. The two wheeled balancing strategy has a significant advantage over a standard non-balancing robot: the ability to turn in place without requiring additional room. In the geocaching setting this attribute is extremely useful as dead ends frequently necessitate backtracking in tightly constrained spaces. Also, the ability to turn in place allows the robot to get to any position without ever having to drive in reverse. This both simplifies the control algorithm and ensures a forward facing sensor is sufficient for obstacle detection.

The control box on the RMP receives commands and sends sensory information through a CAN interface. While there are a number of commands that can be used to control various aspects of the platform only forward velocity and angular velocity commands are actually used. The RMP receives these commands at a frequency of 100 Hz and attempts to meet them while maintaining the robot's balance. The robot also returns sensory measurements at the same rate: these measurements include left and right wheel odometry counts, pitch (forward rotation), roll (left-right rotation), and battery level. The wheel counts are determined from internal odometry encoders and are accurate to +/- 0.064 of a degree [35]. The pitch and roll are measured by the platform's two internal gyroscopes and thus their accuracy is dependent on the roughness of the terrain. However, during normal outdoor operation these values were found to be accurate to approximately +/-0.1 degrees. Also the RMP control box will, as a safety precaution, immediately power down the robot if the CAN interface loses its connection either by wires being pulled loose or by closing the emergency stop circuit.

The RMP platform was modified so that it would be better suited to handle rough outdoor terrain. These modifications included two large roll bars visible in both Figures 3.1 and 3.2. These bars are designed to prevent damage to the sensors in case the robot runs into an obstacle or falls over. The original thin plastic fenders and indoor wheels were also replaced in favor of all-terrain tires. These tires both increase traction and allow the robot to drive over larger height changes.

3.2 Sensors

The sensors included in the RMP platform do not provide enough information to solve the geocaching problem. Therefore, this robot is further outfitted with a GPS receiver, inertial measurement unit (IMU), laser rangefinder, and a camera. These sensors are highlighted in Figure 3.2.

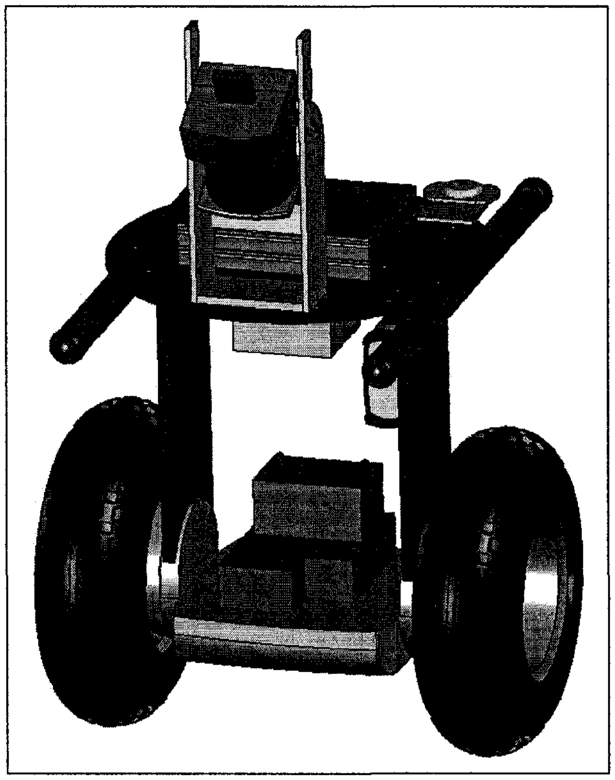


Figure 3.2: The locations and sizes of the hardware included in this system are shown. Components in this simplified model are color coded for easy identification as follows: the camera is shown in red, the laser rangefinder in blue, the GPS antenna + IMU + Novatel integrator box are all yellow, the 12 volt battery, which powers these sensors, is shown in green, the laptop computers are shown in orange, and finally the RMP control box is shown in bright pink. This control box is slightly different then the one shown in Figure 3.1 because it has since been upgraded to reduce its size.

3.2.1 GPS + IMU Sensor

In addition to the position sensors on the RMP platform the Kato geocaching system also employs a Novatel ProPac G2–Plus sensor. This sophisticated sensor combines measurements from both a GPS receiver and a Honeywell HG1700 inertial measurement unit (IMU) to produce a position estimate that is much more accurate than a standalone GPS receiver. The GPS receiver periodically triangulates a global position from satellite signals and can generally return a position estimate accurate up to +/-8 meters. Because this estimate is always recalculated from scratch, this error measure does not accumulate over time. The IMU sensor, on the other hand, calculates its position by measuring accelerations in the 3 spacial dimensions as well as velocities in the 3 rotational dimensions. The position is maintained incrementally by integrating these position estimates over time. Although this sensor is extremely accurate for short range movements, the constant integration causes errors to accumulate as the sensor moves. Because these two measurements have advantages and disadvantages that are essentially opposite, the sensor is able to combine them, usually with a Kalman filter, to produce a more accurate overall estimate. The GPS position was found to be accurate to approximately +/-0.3 meters in normal conditions. The sensor is also able to provide an altitude estimate accurate to +/-1.0 meters and rotational estimates accurate to +/-0.01 degrees.

3.2.2 Laser Rangefinder

To measure the terrain in front of the robot a SICK LMS-200 laser rangefinder is used. The laser is rigidly mounted at the highest point on the robot at pitched forward 27 degrees. In this "push-broom" configuration this sensor scans a section of terrain approximately 2 meters in front of the robot and 5 meters wide, assuming the terrain is flat and the robot is upright. The LMS-200 is configured to send out 401 individual laser pulses at 0.25°increments over a 100 degree range. Each of these distance measurements are accurate to approximately +/-0.5 cm and the sensor competes 19 full sweeps per second. The laser communicates with the on board computers through a USB to serial interface.

3.2.3 Camera

For the purpose of identifying the goal object, a low-cost Unibrain FireWire color camera is used. This camera is only turned on during the local search phase of the task, as explained in Chapter 6. A polarizing filter was placed over the camera to reduce both overexposure and specularities caused by reflected light. This camera is configured to capture 320x240 resolution color images at a 3.75 Hz framerate. This sensor has a 40 degree horizontal field of view and a 30 degree vertical field of view. Also, the camera is mounted on top of the laser sensor and pitched forward 30 degrees, the reasoning behind this choice is also provided in Chapter 6.

3.2.4 Power

The laser, IMU+GPS, and camera are all powered from a single 12 volt lead acid battery. The IMU+GPS sensors both require 12 volts so they are wired directly while the laser sensor is connected through an off-the-shelf 12 to 24 volt converter. The battery can power these sensors for approximately 6 hours on a full charge.

3.3 Computers

The computational resources for this system are provided by two on-board laptop computers and one additional off-board console computer. All of these computers are IBM ThinkPad X-series laptops running Gentoo Linux. The two on-board laptops are an 800 MHz X31 laptop with 512 MB RAM and a 2.0 GHz X60 with 1 GB RAM. These laptop computers communicate with each other through a wireless 802.11g *ad hoc* network. The older X31 laptop is used exclusively to interface with the sensors and make the information available over the network. The faster X60 laptop is used to run the mapping, navigation, and local search procedures where the majority of the processing takes place. This laptop is also hooked directly to the camera so that the images would not need to be sent over wireless. The off-board computer is the same older X31 model laptop and is used to start and monitor processes as well as interface with the joystick for teleoperation.

Chapter 4

Perception

Receiving input from sensors, and organizing that input into a representation of the environment which can be used for path-planning is a fundamental aspect of the geocaching task. This chapter explains the various methods and procedures used by Kato to construct such a representation.

The explanation of how this system constructs a model of the environment from sensory information (the perception problem) is addressed in two parts: the localization problem and the mapping problem. Localization relates to the problem of estimating the robot's position and orientation (pose) in the environment. Estimating the robot's pose is necessary for both planning a path to the goal and for integrating relative terrain measurements, taken at different positions, into a continuous map. Consequently, any errors in this estimate will effect the performance of both the path-planning and mapping subsystems. Section 4.2 outlines how the Kato Geocaching system addresses the localization problem with the objective of minimizing this error.

The mapping problem relates to the construction of a map that can be used to detect and avoid unsafe areas of terrain. To construct this map, a laser rangefinder is used to scan the terrain in front of the robot as it navigates. The current estimate of the robot's pose is used to combine sequential laser measurements into a congruent map of the terrain. The geometric features of this map are then used to classify sections of terrain as unsafe. The details of this mapping system are presented in Section 4.1 and it is assumed that the localization estimate is available. These subsystems are presented in seemingly reverse order because part of the localization estimate, namely the elevation of the robot, depends on information from the terrain map.

4.1 Mapping

In order for the robotic system to detect and avoid dangerous terrain, the system must have a set definition of what properties make terrain unsafe. In this thesis, dangerous, or *untraversable*, terrain is defined as any part of the environment that the robot is physically unable to traverse without being rendered immobile (e.g. getting stuck or falling over). Since the list of examples of such terrain, in an outdoor environment, is inexhaustible (e.g. shrubs, mud, buildings, trees, curbs, or potholes) it is difficult to come up with simple definition of unsafe terrain. However, since many common obstacles in an outdoor environment can be identified by a change in height, the effect of this property on traversability was investigated with a series of "crash" tests. These tests were conducted by teleoperating the base Segway RMP over terrain with varying height changes at different angles and speeds. A summary of how slope affects traversability was generated from these tests and is defined by the following special cases: objects with infinite slope (solid blocks) cannot be traversed if they are higher than 8 cm; objects with 45°slope (small hills) cannot be traversed if they are wider than 12 cm; and objects that are longer than 30cm (large hills) cannot be traversed if their slope exceeds 32°. Also, it was determined that the RMP was able to traverse downsloping terrain with much less difficulty than the equivalent upsloping terrain. Section 4.1.1 provides an explanation of how the system interpolates between these special cases to produce a terrain classification scheme that is simple, robust, and reasonably exclusive.

To measure height changes of the terrain in front of the robot, a laser rangefinder sensor is used. This sensor was chosen because it provides highly accurate terrain information and is easily interpreted. Also, the sensor operates at a relatively high frame rate which allows the robot to safely navigate at a higher speed. Since other outdoor navigation systems have also used multi-view camera sensors [9, 3] these were also considered. These sensors are appealing because they are able to return depth information over a large area and are readily available. However, the performance of stereo vision systems is highly variable across different terrain types and thus were not deemed as effective a terrain sensor as the laser rangefinder.

3D Point Cloud

The result of a single 401-ray sweep from the laser sensor is a set of points in a polar coordinate system. Before these points are converted into 3D Cartesian coordinates they are filtered to remove measurements most likely caused by light refractions or very small airborne objects (e.g. snowflakes, pollen, or insects). To filter out the scans resulting from these environmental influences, the system removed scans where *all* of the following conditions are met:

- The two neighboring scans are both valid.
- The difference in length between the scan in question and each of it's neighbors is above a set threshold (τ_1).
- These differences are in the same direction (i.e. they have the same sign).

The pseudocode for this process is shown in Algorithm 1 as the **FilterScans** procedure. The filtering process is based on the principal that since the scans are separated by only a quarter of a degree, any obstacle that poses a danger to the robot will be large enough to be detected by at least two scans. During tests conducted for this thesis, the distance threshold was set to $\tau_1 = 20$ cm. Additionally,

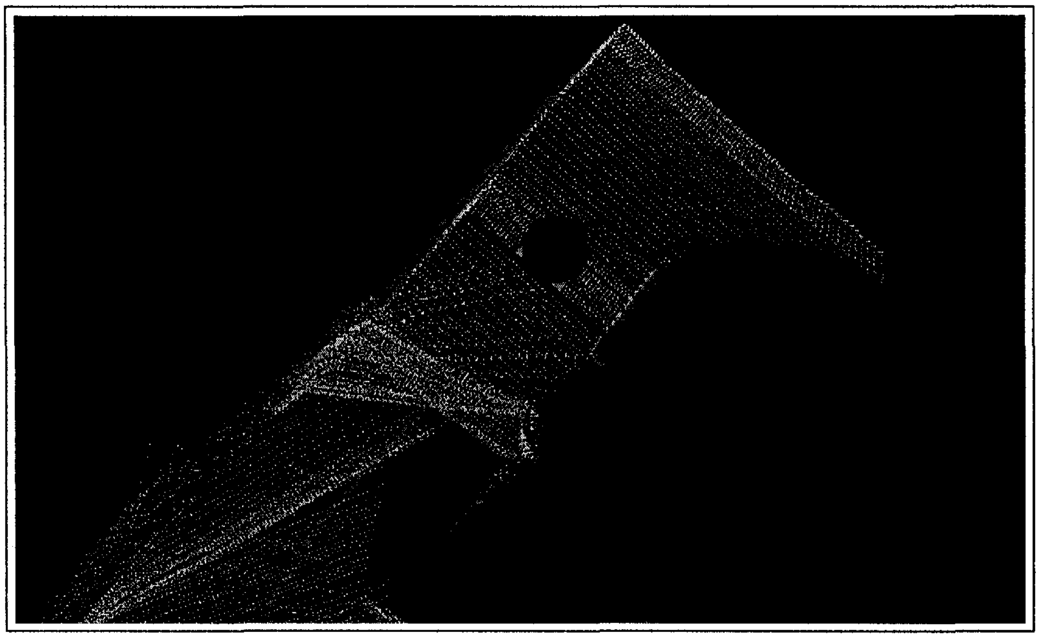


Figure 4.1: This image shows a top down view of the 3D point cloud produced during a test run. As cells differ from the current height of the robot (measured at the bottom of the wheels) they turn from green, to yellow, and eventually red.

scans that were shorter than 20 cm or longer than 4.6 m were removed because they were likely the result of measurement errors or corrupted by localization error.

Since the exact position of this sensor with respect to the robot is known, and the position of the robot with respect to the global environment is known (as discussed in Section 4.2), these measurements can be converted into 3D points in the global reference frame. Conversion to the world reference frame is necessary to allow scans taken at different times and locations to form a unified 3D point cloud, which represents the contour of the terrain. To execute this conversion each laser measurement is first converted from polar coordinates into Cartesian coordinates. Each of these points is then multiplied by the transformation matrix representing the position of the world center with respect to the laser. This transformation matrix is computed using the forward kinematics functionality in the ROBOOP open source robotics library [14]. These steps are taken in the **ConvertScans** procedure in Algorithm 1. After this procedure is completed, the resulting 3D points are added to the current terrain map, along with previous readings. A rendering of a unified point cloud in a real environment is shown in Figure 4.1.

A limitation of the laser mapping process is that the 3D point cloud only contains measurements to the first object in the environment that the laser beams come in contact with. While this provided near certainty that an object is present at that 3D location, it is not possible to determine if the point lies on the load bearing surface. Vegetation, for example, is a common terrain feature that will occlude the true ground plane. This occlusion is a concern because it can not only make safe terrain appear unsafe, which can lead to inefficient paths, but also make unsafe terrain appear as safe, which can lead to the robot crashing. While researchers have developed methods for filtering out scans that likely correspond to specific types of vegetation [46] these approaches require assumptions that are not true in all outdoor environments. Ultimately, the inability to directly measure the ground plane is a limitation of the sensor modality and currently cannot be effectively addressed. The consequences of this limitation can be minimized by sacrificing potential efficiency and assuming that all the 3D points correspond to solid obstacles. The loss in efficiency results from the robot navigating around such terrain as tufts of grass or small twigs that could have been driven over safely.

Probabilistic Terrain Representation

Storing the 3D point cloud produced by the laser for even a short time requires a substantial amount of memory (one minute worth of readings is 5.2 MB). For this reason the point cloud information is summarized using a discretized probabilistic representation. In this representation, the terrain is divided into a lattice of square cells, each with the same width, where each cell contains a mean and variance over the height of the 3D points that fall within its boundaries. The width of these cells is an important parameter because as the width increases, it begins to smooth out local height variations and the system is less able to detect small, but potentially dangerous, obstacles. Conversely, as the width of these cells is reduced, measurement errors and small terrain variances become much more pronounced. Results from various trials indicated that a cell width of 12.5 cm effectively balances smoothing out errors and maintaining sufficient detail. The mean and variance of a cell was maintained incrementally using 3 sums: the number of samples (n), a sum of each sample's value $S_1 = \sum_i X_i$, and the sum of each sample's value squared $S_2 = \sum_i X_i^2$. Equation 4.1 shows how the mean and variance are computed from these sums.

$$\begin{aligned} Mean(X) &= E(X) = S_1/n \\ Var(X) &= E(X^2) - E(X)^2 \\ &= S_2/n - (S_1/n)^2 \end{aligned} \tag{4.1}$$

4.1.1 Classification

The probabilistic terrain representation, which is created through the aforementioned methodology, is used to identify untraversable terrain according to our definitions. The three special cases determined by the crash tests illustrated how height changes in the load bearing surface can affect the traversability of the terrain. Ideally, the robotic system would classify this terrain by comparing the height differences in the load bearing surface between adjacent cells. However, due to the limitations of the laser range finder, discussed in Section 4.1, it is not possible to determine the true height of the load bearing surface. Additionally, due to the discretization in the probabilistic terrain representation, height variations within a cell are likely even for solid terrain like pavement.

There are several approaches that address the problem of estimating the height of the load bearing surface from the probabilistic map. To determine which approach was the most effective, different

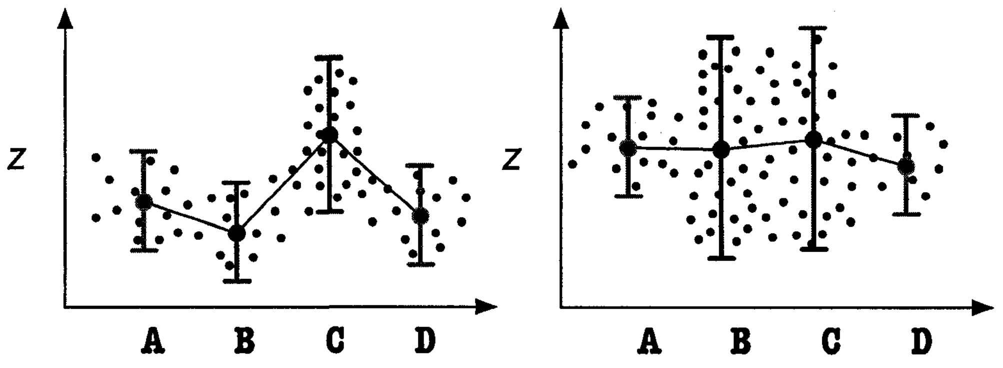


Figure 4.2: Cross-sections of two example probabilistic grids constructed from 3D point clouds are shown. In the diagrams the 3D laser points are drawn as small circles, the grid cells (A, B, C, & D) are marked by a mean (small circle) and scaled standard deviation bars. In the left diagram cells B & C are marked as unsafe because the difference between their means exceeds threshold. In the right diagram cells B & C are classified as unsafe because their variance exceeds threshold.

estimates were implemented and tested in environments representative of typical outdoor scenarios. The different estimates used in the trials included using the max of the z values, using the mean of the z values, and using the mean +/- one standard deviation. The terrain grids produced by these methods were evaluated by hand with additional information provided by a video of the trial runs. It was determined that using the mean value for a cell was the most reliable technique, mainly due to its resistance to outliers which the other techniques were far more sensitive to.

Developing a terrain classifier that will identify unsafe areas is relatively straightforward once the estimate for the height of the load bearing surface is computed. The terrain classifier employed by the Kato geocaching system considers the height of an individual cell compared to the height of its neighbours. If the magnitude of the height difference between the selected cell and any of its neighbours exceeded the set threshold (τ_3), the cell is classified as untraversable. For all of the tests conducted for this thesis, a 5x5 square neighbourhood area was used and the threshold was set at 8.3 cm. This classifier is able to address all of the special cases developed during the crash tests and was computationally inexpensive to employ. It captures vertical obstacles over 8 cm tall, and 45°obstacles wider than 12 cm, by comparing to immediate neighbours. By comparing neighbours two cell widths apart, the classifier marked terrain that exceeded 33°slope over 25 cm. The classifier parameters did not exactly match the special cases because they are set to be more conservative (err to the side of safety) to better handle measurement and localization errors.

In addition to sloped terrain, rough terrain can be difficult for the robot to traverse effectively. In order to avoid traversing over rough terrain, the classification subsystem will also label a cell as unsafe when the measured variance exceeds a set threshold. This classification can also potentially capture unsafe terrain where the load bearing surface estimate is incorrect. For the tests conducted in this thesis work a variance threshold of ($\tau_2 = 0.01$) was used. Figure 4.2 shows examples of both the slope and the variance classification schemes in action and the **isTraversable** procedure in

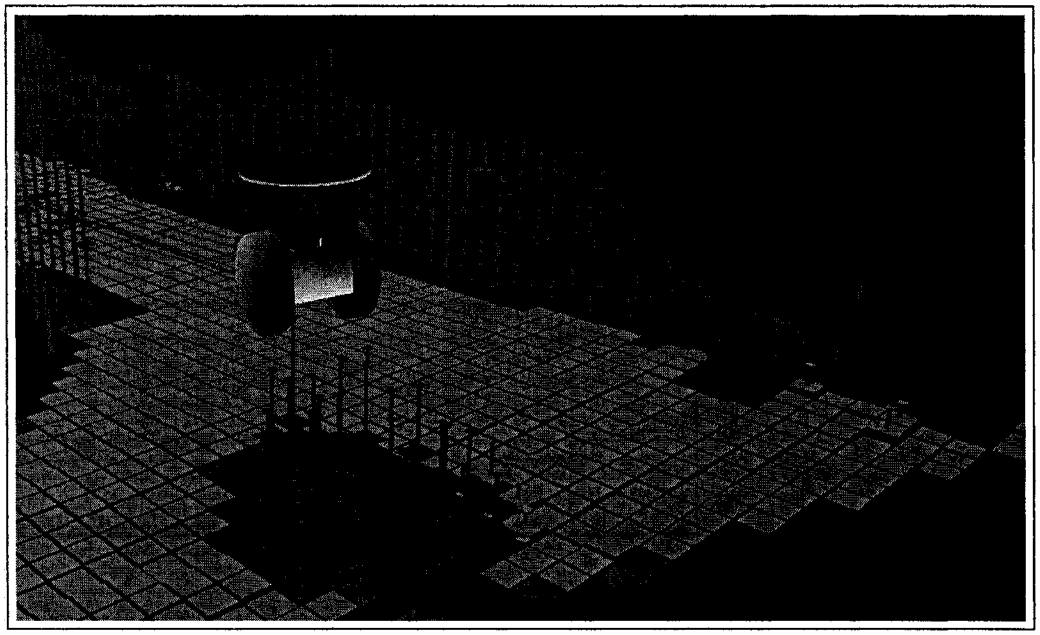


Figure 4.3: The probabilistic terrain representation during a real test run is shown. The tiles represent the mean value for a cell and the vertical bars represent the variance. A green cell has been classified as traversable and a red cell has been classified as untraversable by either high variance or slope. When a variance bar is blue it's value is below the threshold, if its red then the value is above threshold.

Algorithm 1 shows the pseudocode.

Once all of the new cells for a laser frame have been properly classified the mapping component is finished. The new classifications are then passed to the path planning subsystem, which updates its own map and produces a new path.

4.1.2 Addressing Localization Error

Several modifications were added to the terrain grid and classification scheme to address accumulating error in the localization estimate. The source of this accumulating error is explained in Section 4.2. The key attribute of these modifications was to avoid the comparison of laser measurements that were taken at significantly different times. This is because any height discrepancies are just as likely to be the result of localization error as they are changes in the actual surface. The first modification was to remove old values in the probabilistic grid when new data is received for a given cell. This was implemented by marking each cell with a timestamp when new data is received. Then, any time another batch of data is received for a cell this timestamp is checked, if the cell's data is older than a set threshold the values were reset before the new data was inserted. The second modification that was introduced involved avoiding comparison of neighbouring cells in the classification step if their timestamps were sufficiently different. During the testing stages a threshold of 2 seconds was used for replacing old laser measurements and 8 seconds was used for ignoring neighbours in the classification stage.

4.1.3 Implementational Details

The pseudocode for the entire mapping and classification system is shown in Algorithm 1. This code explains the method in which the subsystem processes distance measurements received from the laser sensor (`lVals`) and a pose estimate (`Pose`) and subsequently updates the map. Upon receiving this data, the algorithm first filters out the scans that correspond to measurement errors or small objects in the **FilterScans** procedure. The filtered readings are then converted to the world coordinate system in the **ConvertScans** procedure by first converting them from polar coordinates into Cartesian coordinates. From there each of the points are transformed into the world coordinate system using the transformation produced by the kinematics library and the current pose on line 8. At this point the 3D point cloud for the current laser frame has been produced and is used to update the probabilistic grid one point at a time.

The "for" loop on line 17 iterates over each of the 3D points and updates the grid cells for which they correspond. To reduce access times, the values for the probabilistic grid are stored in a hash table using the (x, y) location as the key. However, since the grid is discretized, integers are used for the (x, y) location of a cell; lines 18 and 19 illustrate how these integers are calculated for a single point. Once the key has been calculated, it is used to reference the grid cell and update the values accordingly. The newly updated grid cell, along with its neighbours (G_{neigh}), are then added to a hash table with the objective of noting which cells need to be reclassified. A hash table is used here to automatically remove duplicate entries.

The algorithm finishes by iterating through the hash table and recalculating the mean and variance for each identified cell prior to sending it for reclassification. In this document, the loop is simplified for brevity but the actual implementation uses two loops to ensure all of the variances and means are updated before any classifications are done. These classifications are computed using the **isTraversable** procedure, which first verifies the variance on a cell and then compares the cell's height to that of its neighbours. If the magnitude of the variance or height difference is above the set threshold then the cell is marked as untraversable.

4.2 Localization

The following section explains the method in which the pose of the robot, in the world reference frame, is estimated from sensor readings. The pose of the robot is described as a 6 dimensional vector, $(x, y, z, \theta, \gamma, \phi)$, containing 3 spatial and 3 rotational dimensions. The x and y variables represent the location of the robot on the map while the z variable represents its height or elevation. The rotational variable θ represents the current rotation around the z-axis and is also referred to as the heading of the robot. γ is the forward/backward rotation (pitch) and ϕ represents the rotation from side to side (roll). The sensors that are used to estimate these variables include wheel odometry, IMU+GPS readings, internal RMP gyroscopes, as well as the laser sensor.

Algorithm 1 : Mapping and Classification

```
procedure FilterScans(lVals)
    for $i = 2 : len(\text{lVals}) - 1$
      if (isnan(lVals($i-1 : i+1$))) continue;
      $d_1 \leftarrow \text{lVals}(i-1) - \text{lVals}(i)$
      $d_2 \leftarrow \text{lVals}(i+1) - \text{lVals}(i)$
      if $(|d_1| > \tau_1)$ & $(|d_2| > \tau_1)$ & $(\text{sign}(d_1) = \text{sign}(d_2))$
        lVals($i$) $\leftarrow NAN$
    end

procedure ConvertScans(lVals, Pose)
    $T \leftarrow$ kinematics.laser(Pose)
    for $i = 1 : len(\text{lVals})$
      $P_x(i) \leftarrow \text{lVals}(i) * cos(40 + i/4)$
      $P_y(i) \leftarrow \text{lVals}(i) * sin(40 + i/4)$
      $P_z(i) \leftarrow 0$
      $P(i) \leftarrow T * P(i)$
    end

procedure UpdateMap(lVals, Pose)
    FilterScans(lVals)
    ConvertScans(lVals, Pose)
    for $i = 1 : len(\text{lVals})$
      $G_x \leftarrow round(P_x(i)/\text{resolution})$
      $G_y \leftarrow round(P_y(i)/\text{resolution})$
      ref $\leftarrow$ GridHash[G]
      ref.$s1$ += $P_z(i)$
      ref.$s2$ += $P_z(i)^2$
      ref.$s3$++
      updateHash[$G + G_{neigh}$] $\leftarrow$ true
    end
    for ref $\in$ updateHash
      $\text{ref}_{avg} \leftarrow \text{ref}.s1/\text{ref}.s3$
      $\text{ref}_{var} \leftarrow \text{ref}.s2/\text{ref}.s3 - (\text{ref}.s1/\text{ref}.s3)^2$
      $\text{ref}_{trav} \leftarrow$ isTraversable(ref)
    end

procedure isTraversable(ref)
    if $(\text{ref}_{var} > \tau_2)$ return false
    for $(N \in \text{ref}_{neigh})$
      if $(|\text{ref}_{avg} - N_{avg}| > \tau_3)$ return false
    end
    return true
```

4.2.1 Estimating (θ, γ, ϕ)

The three rotational variables are the easiest to estimate because the internal RMP sensors and the IMU+GPS sensor provide accurate absolute measurements for them. The pitch and roll values are provided by both of these sensors to a comparable degree of accuracy. However, only the measurements returned from the RMP's internal gyroscopes are used because they are generated at a faster rate. These values are crucial to the accuracy of the terrain map because the errors are magnified when the laser measurements are converted into the world reference frame. For instance, a $1°$error in the pitch estimate can lead to a 4 cm error in the height estimate within the terrain map. Also, a $1°$error in the roll estimate can cause up to a 6 cm error within the terrain map. Fortunately, the RMP's sensors provide measurements with an absolute error of approximately +/- $0.1°$. The heading is read directly from the IMU+GPS sensor which generates this value with an absolute error of approximately +/- $0.01°$. These error values, which were determined empirically by observing their values while teleoperating the robot, are low enough to allow the terrain map to effectively represent true height changes as evidenced by its test performance presented in Chapter 7.

4.2.2 Estimating (x, y)

The three spatial variables (x, y, z) can be read directly from the IMU+GPS sensor, however the high error associated with these values limits their applicability for map building. These errors occur because the sensor is tuned to produce values that have a low absolute error, which is the error in position at any given time regardless of motion, rather than a low relative error, which is the error in the change of position. For example, the position returned from a handheld GPS sensor will change rapidly due to noise in the signal, despite its stationary position. However, the magnitude of this error will not increase as the sensor is moved throughout the environment, assuming the signal remains constant. Because of this, the GPS unit is said to have a high relative error but low, or at least constant, absolute error. Conversely, a position estimate that is maintained by continually integrating measurements, such as odometry or IMU readings, can be highly accurate over short distances but will become less accurate as the sensor moves due to the accumulation of error; this estimate has low relative error and high absolute error.

It is important to distinguish between these two errors because they have different effects on the overall performance of the system. The mapping subsystem, for instance, is highly sensitive to relative error because it compares small height changes as the sensor makes small movements. If the robot's position has high relative error, the measurements between successive scans become meaningless. The planning system, on the other hand, is more sensitive to absolute error because it operates at a much larger scale. As position errors accumulate, the map of the environment begins to warp and the optimality of the path is compromised. Additionally, as the robot's position estimate drifts, the position of the fixed goal, relative to the robot, becomes outdated.

To estimate the (x, y) position of the robot, with low relative error, the Kato geocaching sys-

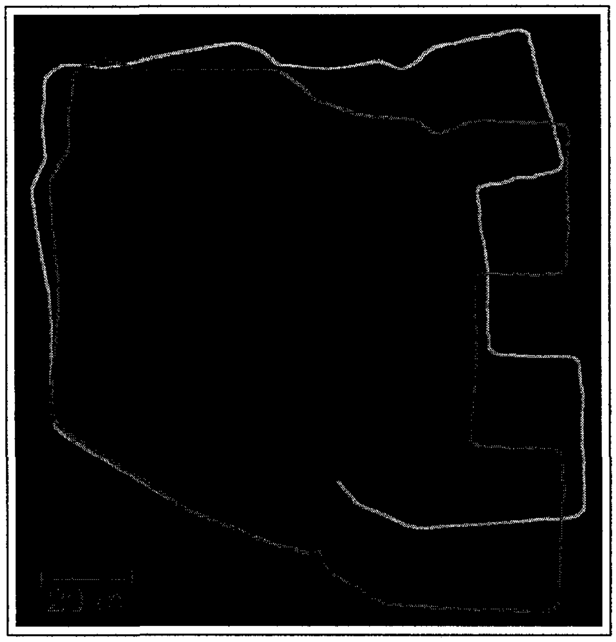


Figure 4.4: The paths produced with odometry alone (green) and with odometry combined with the GPS+IMU heading infomation (purple) are shown. Both these paths were generated from the same sensor log file captured while teleoperating the robot around an outdoor environment and ending in the exact same location as it started.

tem incrementally updates its location estimate using wheel odometry and the heading information supplied by the IMU+GPS sensor. The wheel odometry returned by the RMP is used to produce a highly accurate measurement of the distance the robot has traveled. This simple geometric calculation was computed using the returned wheel rotations and the measured tire circumference to determine how far each wheel has moved on the ground. The distances calculated for each wheel are averaged to provide the distance traveled by the entire robot. Given a previous estimate for the position of the robot, the current heading, and the calculated distance, the new position can be easily computed.

Environmental factors, such as variable tire inflation and wheel slippage, cause the accumulation of position estimate errors as the robot moves. To determine the accumulation rate of these errors, five test runs were conducted by teleoperating the robot through different outdoor environments and returning it to the starting location. To determine the relative error, the euclidean distance between the estimated start and end position was divided by the total distance traveled. From these runs, it was determined that the (x, y) position estimate was accurate to approximately 1% of the distance traveled, on average. Figure 4.4 shows the performance of this position estimate, compared to an odometry-only estimate, on one of the test runs.

The geocaching task often requires navigating up to several kilometers and, as such, even a 1% error can cause the position estimate to be off by tens of meters by the time the goal location is reached. To compensate for the accumulating position error, the goal location is periodically updated using its known GPS location and the current GPS location of the robot. Since the GPS reading does not accumulate error over time it will always reset the goal location to a position within a known absolute error. This is similar to the strategy employed by the CMU PerceptOR robot [21].

4.2.3 Estimating z

The elevation of the robot (z-value) is the most difficult parameter to estimate because no sensors, other than the GPS+IMU, directly provide a measurement of it. However, the values returned by the GPS+IMU sensor are almost entirely useless as they have an absolute error of roughly +/- 1.0 m. Due to the lack of good sensors, earlier versions of this system were forced to simply assume that the z value of the robot was always constant at 0.0 m. This assumption worked quite well for the majority of the terrain encountered but there were specific scenarios that caused the system to fail. Not surprisingly, these scenarios all occurred when the robot was traversing up or down sloped terrain. Figure 4.5 shows how the lack of a z estimate affects the terrain map for two example scenarios. The main point these figures illustrate is that once the robot's z value starts to change, it directly subtracts that change from the terrain map. In some cases, like the one depicted in the right image, this constant subtraction causes the robot to underestimate the severity of the slope resulting in a crash. These issues highlighted the need for a procedure to estimate the elevation of the robot.

In order to obtain an estimate for the z position of the robot, an approach that uses the terrain grid produced by the laser was developed. This approach relies on the fact that the robot will always obtain a fairly accurate estimate for the height of the load bearing surface before it drives over it. Since the position of the contact points on this surface can be calculated from the robot's position it is relatively straightforward to estimate the height of the entire robot. This approach, however, is prone to accumulating errors which can cause positive feedback leading to a "stair" effect. This effect appears when the height estimate drifts over time and then is corrected very rapidly by moving onto more recently scanned terrain. This causes the robot to believe it has jumped up or down, which in turn, causes a sharp height change in the terrain that is currently being scanned by the laser. When the robot navigates over this artifact it again incorrectly updates the height estimate creating another artifact, and the pattern continues. Not surprisingly this "stair" effect can lead to incorrect terrain classifications having any number of detrimental effects. Additionally, this effect could be seen again on a smaller scale when the robot navigated from one discretized cell to another. The height difference between these cells would cause an abrupt change in z leading to various terrain artifacts. To reduce the occurrences of these artifacts a number of modifications to the original approach were introduced. First, instead of averaging between only two contact points at the base of each tire a total of 12 points were averaged. These points were spread evenly over the base of each tire in a 20x10 cm rectangular area. This modification essentially smoothed the height changes as the robot moved along the height grid. To address the larger scale effect the change in height was bounded in two separate places. First, each of the 12 readings was capped at a maximum change of 4 cm from

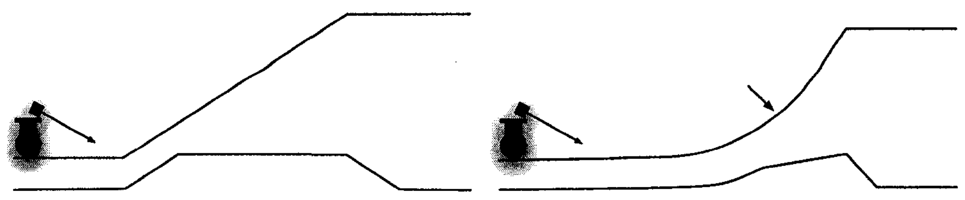


Figure 4.5: Two example hill scenarios and how they are perceived by the robot when the z value of the robot is assumed to be zero at all times are shown. The top line shows the contour of the true ground and the bottom line shows the robot's perceived environment map. The small arrow on the right diagram shows the point at which the robot would no longer be able to climb the hill.

the current z value for the robot. Second, the z change for the entire robot was capped at a maximum speed of 11 cm per second. This setting allows the robot to move up or down at a maximum slope of 28°when traveling at its maximum speed of 0.4 m/s which is sufficient to model all traversable hills.

The performance of the localization estimate was evaluated using a number of different outdoor test runs using both the autonomous controller and a human operator. Each of these runs were conducted with the z value set to zero and with it estimated from the terrain map. The first tests focused on small scale scenarios where the importance of the z estimate was already known. The first scenario was a large hill with increasing slope very similar to the one depicted in the right image in Figure 4.5. The terrain grid obtained by teleoperating the robot straight up this hill is shown in Figure 4.6. What is interesting about this particular figure is that the robot turns sideways at the top of the hill. Because the robot has a highly accurate estimate of the roll angle the true slope of the hill is shown by the terrain map after this turn. The misalignment shown in the middle figure shows just how bad the terrain estimate is when z is assumed to be zero. Additionally, autonomous tests were conducted on this same hill to determine how the estimate, or lack of an estimate, affects the behavior of the robot. Not surprisingly when the elevation was assumed to be zero the robot was not able to measure the extent of the slope and kept driving up the hill until the RMP fell over. This test was repeated for 2 other hand picked scenarios with similar results.

The second experiment measured how the errors accumulate in this estimate as the robot was teleoperated around in different terrain for approximately 10 minutes before returning to the original start location. The final estimate for z was then used to determine how far off the estimate had drifted. Although the error is dependent on the type of terrain and the number of hills in the environment, it was discovered that the z estimate is accurate to roughly 4.4% of the distance traveled on grass and 1.3% on pavement. With such low numbers these errors are rather insignificant considering the fact that, like the errors in roll and pitch angles, these errors can only affect classification between cells scanned within a short period of time. In summary, this z estimate allows the robot to correctly avoid several dangerous situations without introducing any major drawbacks.

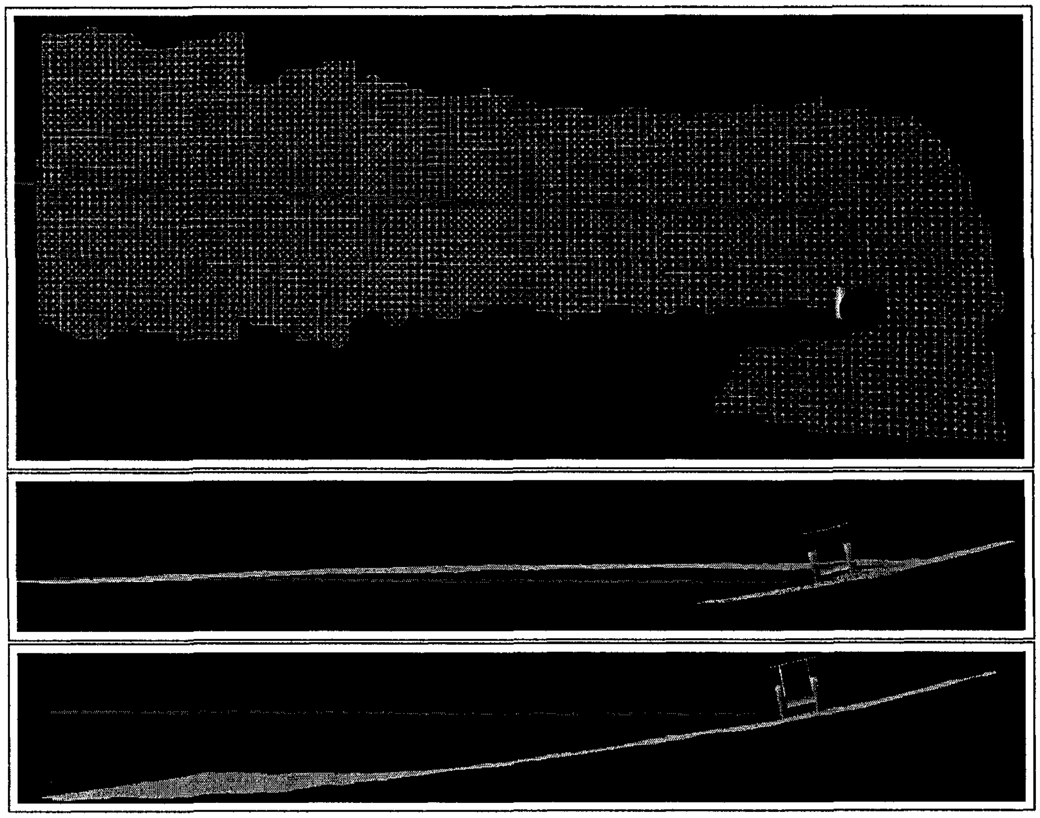


Figure 4.6: The significant difference estimating z can make on the terrain map is shown here. These three images are obtained from teleoperating the robot straight up a hill then turning right. The top image is a birds eye view of the terrain map, the purple line indicates the robot's trail. The other two images show the same grid except from the side, note that the entire trail is drawn at the current z-value of the robot. The middle image shows the grid obtained when the z was assumed to be zero and the bottom image is obtained when z was estimated.

Chapter 5

Navigation

This chapter explains the strategies and algorithms employed to produce a safe and efficient navigation subsystem for the Kato geocaching system. The objective of the subsystem is to take the information provided by the perception subsystem and issue low-level drive commands to the RMP that will cause it to reach a target location as efficiently as possible. The discussion of how this system searches for the actual goal object after reaching this location is left until Chapter 6. For this chapter it is assumed that there is a single GPS goal location, provided by the perception subsystem. Mobile robot navigation, even for highly-structured environments, is difficult due to the continuous search spaces and robot dynamics involved. In the geocaching context, the unstructured nature of the terrain and the sheer size of the environment increase the computational and strategic complexity of this task. This chapter explores the natural split of the problem by first explaining how the system produces a path to the goal (path-planning), and then explaining the mechanisms used for following the path (control).

The isolated problem of path-planning in the complex geocaching domain poses a number of specific challenges which require the employment of specialized techniques. Such challenges include the necessity to produce very long-distance paths at a resolution high enough to allow for precise local navigation around, and between, obstacles. In addition, the system must be able to efficiently alter the current path in order to adapt to the constant stream of new robot positions and map information. The problem is further complicated by the limited computational resources available on-board the robot. In order to produce an efficient planning system that can overcome these challenges, the exploitable structures of this specific task are explored. Section 5.1 outlines which specific properties of this problem were exploited and how this reduced the overall computation required for planning.

Once the planning subsystem has produced a path to the goal (i.e. a GPS location), the information is transferred to the control subsystem which is tasked with following the path as safely and efficiently as possible. To accomplish this task, the control system analyzes the information contained in the environment map, the robot pose, and the path, to produce angular and forward velocity commands. These commands are then passed on to the embedded software on the RMP, which attempts to meet the velocities while also keeping the robot balanced. In Section 5.2 it is explained how the final control process is implemented by examining it through the development stages; that is, a simple controller is used to start with and a number of incremental improvements are explained.

5.1 Planning

The path-planning subsystem developed for this robot addresses the specific problems associated with geocaching by extending current state of the art search methods. One specific challenge faced in robotic geocaching is that the paths must be planned over distances of several kilometres and still be accurate enough to plan through corridors less than a metre wide. The combination of these two factors greatly increases the overall size of the search graph, which can pose problems for traditional methods. Additionally, because the robot is constantly changing position, new obstacles are continually being detected. Because of this, the path-planner must be able to produce new paths several times per second in order to navigate safely and efficiently. Considering the limited computational resources available on-board the robot, traditional search methods, such as A* [16], are not computationally efficient enough to be of use; because they make no special attempt to reduce the size of the search space. To develop a planning system with the level of efficiency required for this task, specific properties of this problem were exploited to decrease planning costs. These specific attributes were: 1) the distance that the robot moves between planning stages, and 2) the location of the map updates in relation to the robot. Fortunately, a class of algorithms that take advantage of these two properties has already been developed. This class was formed out of the first algorithm of its kind called Dynamic A* (or D*) [39]. This system uses a recently-developed and lightweight version of this algorithm called D* Lite [22].

5.1.1 D* Lite

D* Lite is specifically designed for efficient path-planning on mobile robots in an unknown environment. This algorithm is able to improve the efficiency of the search, in part, because it plans backwards; that is, it plans from the goal location to the robot's location. By planning in this direction, new map updates that occur near the robot's location only affect the end portion of the path, leaving the majority of it unchanged. This would be irrelevant if A* search was used, as A* regenerates the entire path from scratch. The D* Lite algorithm takes advantage of this property by keeping the calculated costs from previous searches and using them when it plans a new path. Because the unaffected section of the path remains optimal, the algorithm only needs to plan starting from as far back as the changes have altered the path. To briefly put this into the context of the entire problem: the laser scans 2 m ahead of the robot and the goal can easily be 1000 m or more away. In this case roughly 99.6% of the cost map does not need to be recalculated when a small obstacle alters the path. Furthermore, by maintaining a cost map in between searches, this algorithm eliminates the costs involved in updating the position of the robot. Since this operation does not affect the cost map, a new path can be produced in linear time.

To discuss the D* Lite algorithm in more detail, the specifics of the geocaching problem will temporarily be put aside and the focus placed on planning in an arbitrary search graph. This search graph is defined by a set of vertices which are connected by a set of directed edges. Each edge is assigned a positive real-value edge cost; the cost between vertices v_1 and v_2 is denoted as $c(v_1, v_2)$. Additionally, two functions are defined over vertices in regard to these edges: Succ(u) returns the set of nodes that can be reached from u, and Pred(u) which returns the set of nodes that u can be reached from.

$$\text{rhs}(u) = \begin{cases} 0 & \text{if } u \text{ is start state} \\ \min_{s' \in \text{Succ}(u)}(c(u, s') + g(s')) & \text{otherwise} \end{cases} \tag{5.1}$$

Additionally, the D* Lite algorithm maintains two cost values for every vertex in the graph: $g(v)$ and $rhs(v)$. The g-value is the current estimate of distance to the goal and rhs is a one-step look ahead based off neighboring g-values as defined in the equation (5.1). Using these values, two types of vertices are defined: consistent vertices, where $g(v) = rhs(v)$, and inconsistent vertices, where $g(v) \neq rhs(v)$. This is a useful distinction; if a vertex is inconsistent then a map update must have caused its value to change, and therefore, it is necessary for the search to update the g-value for the vertex. Additionally, this algorithm uses a heuristic function to recommend node expansions. This function provides the best case cost between two vertices u and v (usually between a vertex and the goal) and is denoted $h(u, v)$.

In order to keep track of the inconsistent vertices in the graph, so they can be updated, this algorithm uses a priority queue. This queue, denoted as U, is sorted so that the vertices that have the lowest cost + heuristic value are on top (ties are broken using cost alone). This is accomplished by setting the priority of each vertex inserted into the queue using a *key* value defined on line 1 of the pseudocode in Algorithm 2. The *key* comparison function ($\dot{<}$) is used to rank *keys* in the queue and it is defined in equation (5.2). This priority queue also implements the following functions: *TopKey()* which returns the *key* at the top of the queue or $[\infty, \infty]$ if the queue is empty, *Pop()* which removes and returns the element at the top of the queue, *Insert(u, k)* which adds a new element u to the queue with priority k, and *Remove(u)* which finds and removes element u from the queue.

$$\begin{aligned} k(u) &= [k_1(u); k_2(u)] \\ k(u) \dot{<} k(v) \quad &\triangleq \quad \begin{cases} k_2(u) < k_2(v), & \text{if } k_1(u) = k_1(v) \\ k_1(u) < k_1(v), & \text{otherwise} \end{cases} \end{aligned} \tag{5.2}$$

When the D* Lite search algorithm is first initialized it sets the g and rhs values to infinity for each vertex in the graph. This initialization ensures that when the rhs value for a cell changes it will be marked as inconsistent and the search will detect and update it. The algorithm then inserts the goal state into the priority queue to provide a starting point for the search. These steps are executed in the **Initialize**() procedure shown in Algorithm 2. Once initialized, D* Lite enters its

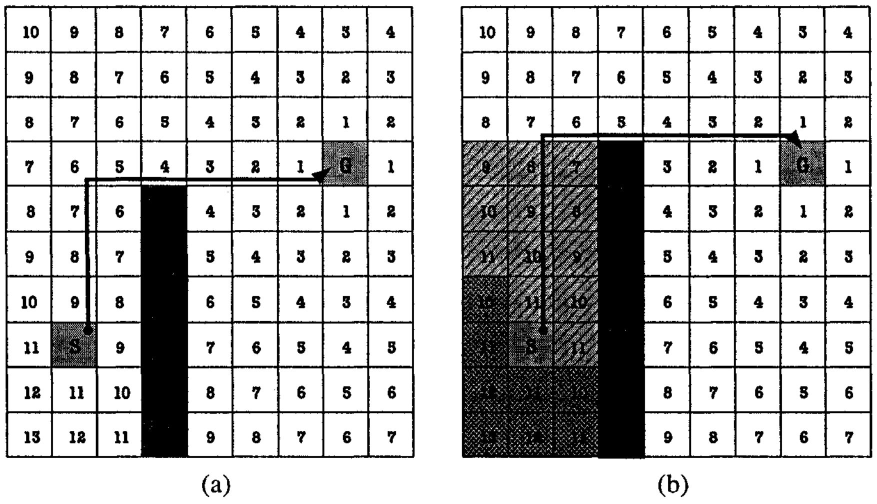


Figure 5.1: A graphic representation of the method used by D* to update the search graph when new map updates occur. Both diagrams show a 4-way connected search grid with a start and goal location. The solid dark red squares represent untraversable cells and thus do not have a cost value. The gray striped cells in (b) have changed the cost due to the new map change and have been updated by the search. The gray checkered cells have also changed cost but are not updated by the search because the start state has already been reached.

main loop, which repeatedly updates the cost map until all the inconsistent vertices with a lower *key* value than the start state, and the start state itself, have been updated. This stopping condition guarantees that all the vertices that lie along the optimal path to the goal are updated, assuming the heuristic is admissible. This main loop is contained in the **ComputeShortestPath**() procedure shown in Algorithm 2.

Each iteration of the main loop will update the vertex at the top of the priority queue in one of three ways. First, if the vertex's *key* is outdated (line 13), it is recalculated and inserted back into the queue. Otherwise, if new map updates have reduced the cost for a vertex (line 15), the node is then made consistent by setting its g-value equal to its rhs-value. All of the vertex's predecessor vertices are also considered for update, since this update may have caused their rhs-values to change as well. This simple procedure is the primary way in which the algorithm propagates the cost changes through the graph. If neither of the first two conditions are met, then the cost of the vertex must have increased. In this case, the vertex's g-value is set to infinity and it, along with its predecessors, are considered for update. A cell is "considered for update" by sending it to the **UpdateVertex**() procedure. This procedure first checks if the vertex is inconsistent by recalculating the rhs-value and comparing it to the g-value. If the vertex is indeed inconsistent it is added to the priority queue to be updated.

Once all of the necessary cost values have been updated, the loop will terminate and the cost map can be used to produce a path to the goal. Figure 5.1 shows an example of how D* Lite updates the cost map in a simple 4-way gridworld. The rest of the pseudocode in the **Main**() procedure describes the production of a path from the cost map and updates any edge changes.

Algorithm 2 : D* Lite [22]

```
procedure CalculateKey(s)
1:  return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$

procedure Initialize()
2:  $U \leftarrow \emptyset$
3:  $k_m \leftarrow 0$
4:  $\forall_{s \in S}\ rhs(s) \leftarrow g(s) \leftarrow \infty$
5:  $rhs(s_{goal}) \leftarrow 0$
6:  U.Insert($s_{goal}$, CalculateKey($s_{goal}$))

procedure UpdateVertex(u)
7:  if $(u \neq s_{goal})$ $rhs(u) \leftarrow \min_{s' \in \mathrm{Succ}(u)}(c(u, s') + g(s'))$
8:  U.Remove($u$)
9:  if $(g(u) \neq rhs(u))$ U.Insert($u$, CalculateKey($u$))

procedure ComputeShortestPath()
10:  while (U.TopKey() $\dot{<}$ CalculateKey($s_{start}$) OR $rhs(s_{start}) \neq g(s_{start})$)
11:    $k_{old} \leftarrow$ U.TopKey()
12:    $u \leftarrow$ U.Pop()
13:    if ($k_{old}$ $\dot{<}$ CalculateKey($u$))
14:      U.Insert($u$, CalculateKey($u$))
15:    else if $(g(u) > rhs(u))$
16:      $g(u) \leftarrow rhs(u)$
17:      $\forall_{s \in \mathrm{Pred}(u)}$ UpdateVertex($s$)
18:    else
19:      $g(u) \leftarrow \infty$
20:      $\forall_{s \in \mathrm{Pred}(u) \cup \{u\}}$ UpdateVertex($s$)

procedure Main()
21:  $s_{last} \leftarrow s_{start}$
22:  Initialize()
23:  ComputeShortestPath()
24:  while $(s_{start} \neq s_{goal})$
25:    if $(g(s_{start}) = \infty)$ return 0  // no path to goal.
26:    $s_{start} \leftarrow \arg\min_{s' \in \mathrm{Succ}(s_{start})}(c(s_{start}, s') + g(s'))$
27:    move to $s_{start}$
28:    scan graph for changed edge costs
29:    if any edge costs changed
30:      $k_m \leftarrow k_m + h(s_{last}, s_{start})$
31:      $s_{last} \leftarrow s_{start}$
32:      for all edges $(u, v)$ with changed cost
33:        update the edge cost $c(u, v)$
34:        UpdateVertex($u$)
35:      ComputeShortestPath()
```

5.1.2 Modifications

To improve the computational efficiency of the planning system, as well as the quality of the path produced, a number of modifications were made to the D* Lite algorithm.

On-Demand Initialization

The first of these improvements involved changing the method in which the cost map was initialized. In the basic implementation, the algorithm generates the majority of the cost map on the very first planning run. This initialization method unnecessarily uses a large amount of computation power and memory as there are no obstacles in the map and the g and rhs values simply equal the heuristic distance to goal. Instead of generating the cost map by actually executing the search, the final implementation of this algorithm computes the g and rhs values for unaltered vertices on-demand using the heuristic function. This modification requires that the search store cost values for vertices only if their values differ from the heuristic function. Considering the large distance between the start and the goal in the geocaching setting, the amount of memory this modification saves substantially reduces the load on the system in addition to improving the start up time.

Lazy Delete

In the D* Lite pseudocode it is assumed that the priority queue supports the *Remove()* function. While this function is intuitively simple, it is not a procedure that is typically supported for an efficient priority queue implementation. For this reason, the functionality had to be implemented using the operations that *are* supported for a priority queue: *Insert()* and *Pop()*. The *Remove()* function was initially implemented by continually popping elements off of the queue and pushing them onto a stack data structure until, either the element to be removed was found, or the end of the queue was reached. At this point, all of the previously popped elements were individually popped off the stack and inserted back into the priority queue. After this function was implemented, a number of tests were conducted to evaluate the algorithm's performance using the GNU C++ Profiler. It was discovered that a large majority of the processing time for the entire algorithm was spent removing vertices from the queue. This is to be expected, as every time a new element is added to the queue the algorithm first attempts to remove any potential duplicates. Since the majority of the time there are no duplicates, the remove function will iterate through the entire queue before it returns. To improve on this naive implementation, a hash table was used to keep track of what elements are currently in the queue. In this implementation, when a new cell was added to the queue it was also inserted into the hash table. The algorithm was then able to verify the presence of a cell in the queue in near-constant time by checking this hash table. To remove a cell from the priority queue, a "lazy" delete strategy was employed; where the cell was only removed from the hash table when the *Remove()* function was called. An element was "lazily" removed from the priority queue by checking if it had been deleted from the hash table when it was popped. If it had been deleted, the

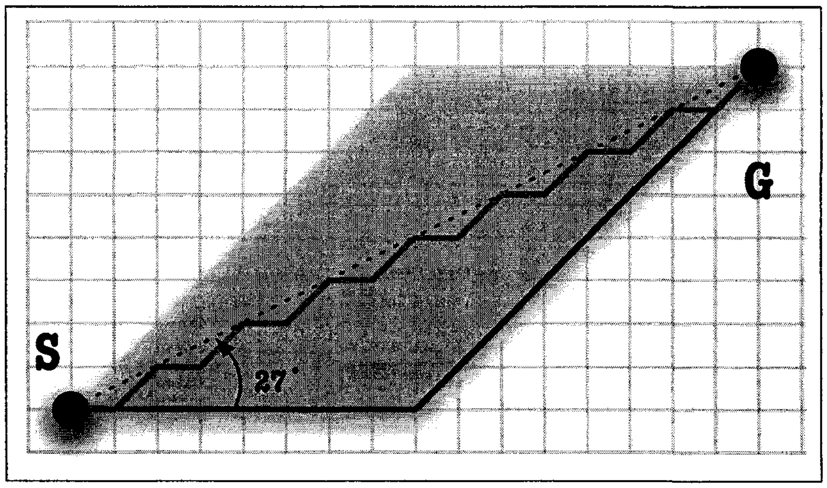


Figure 5.2: The difference between the paths produced by the original D* Lite algorithm (red line) and the modified version (green line). The parallelogram region shows the space covered by all of the 8-way connected paths of optimal length. While both of these paths have the same 8-way connected cost of 19.3 ($8 * (\sqrt{2} + 1)$), the green path is much shorter when actually traversed by the robot. The true shortest path is drawn as the dotted straight line and has a cost of 17.9 ($\sqrt{5 * 8^2}$).

element was skipped and the next element was considered instead. This remove method was found to decrease the average computation time of the entire search algorithm approximately 10-fold, and used only slightly more memory.

Moving Goal Position

In addition to modifying D* Lite to be more computationally efficient, the algorithm and the search graph were also modified to handle specific characteristics of the geocaching problem. One of these characteristics was the periodic update of the goal location to account for accumulating localization errors. To change the position of the goal, the entire search was reinitialized, the new goal location was set, and all of the obstacles from the traversability map were re-added. This procedure was surprisingly efficient as most of the obstacles lie behind the robot and therefore do not affect the search. This procedure also had the added benefit of clearing out the deleted entries in the open list, which reduced the amount of additional memory the "lazy" delete modification used.

Improving Path Distance

To decrease the real world distance of the paths, the final implementation used an 8-way connected search graph rather than the standard 4-way connected graph, as illustrated in Figure 5.1. The edge costs in this grid were set to the true Euclidean distance between the vertices: 1 for axis-aligned movements and $\sqrt{2}$ for diagonal movements. Also, instead of setting the edge cost to infinity for untraversable vertices, they were simply removed from the graph. Although the 8-way path produced by the planner is better, as it allowed more direct paths, it is still a discretization and thus cannot describe the truly optimal path. In fact, an optimal 8-way connected path can be up to 8% longer

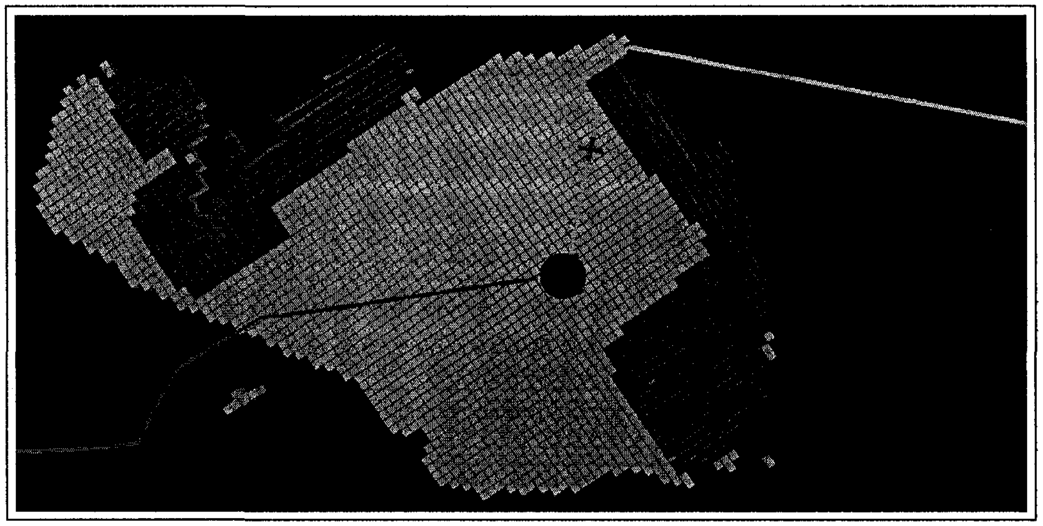


Figure 5.3: The path and terrain grid for part of a test run. The robot's trail is shown in purple, the robot is the gray circle, and the planned path is yellow. The purple X on the path is the current waypoint. The terrain grid is shown as a set of tiles, green is traversable, red is untraversable, and blue cells are in the expanded region around the untraversable cells.

than the true shortest path ($\frac{(\sqrt{2}+1)}{\sqrt{5}}$) , a significant distance considering the size of the search area. This situation is particularly noticeable when there are no obstacles in the way of the goal; instead of driving directly towards the goal, the robot may drive at an angle up to 27°off of the straight line path. This issue was addressed by modifying the greedy search, that produces the path from the cost map, to break ties by expanding the vertex that was closest to a straight line between the start and goal positions. To calculate which vertex is closer to this straight line, the algorithm calculates the Euclidean distance between the vertex and the start location as well as that between the vertex and the goal location. These two values are then summed and the vertex with the minimum sum is closer to the straight line to goal. Figure 5.2 shows the difference between the path produced by the original planner and the one produced by the modified planner.

Expanded Obstacles

The basic D* Lite algorithm assumes that the agent occupies a single cell at any one time. However, since the search grid is set to the same 12.5 cm resolution as the traversability map, and the robot is roughly 80 cm wide, this assumption is clearly violated. If this violation is ignored, the planner will produce paths that cut far too close to obstacles and the robot may run into them. To address this issue, the standard technique of expanding obstacles by marking all cells in a set radius as unsafe is used. For the results obtained in this thesis, a square radius of 3 cells was used. This radius is actually larger than required, so the path can be smoothed safely, as described in the next section. This increased radius allowed the navigation system to produce safe, high-resolution paths without complicating the actual search algorithm. To implement this modification, each vertex retained a count of the number of untraversable cells within a 3-cell distance. Each time a new untraversable cell was added or removed from the search grid, the counts were incremented or decremented accordingly. This expanded region is clearly visible as the blue area in Figure 5.3.

5.2 Control

Once the path-planning subsystem has produced a path to the goal, the control subsystem is tasked with following the path in a safe and timely manner. This involves considering the current path, terrain map, and robot pose, and then sending low-level drive commands to the robot that will cause it to follow the path. The RMP receives these low-level commands in the form of angular and forward velocity values. The embedded software on the platform attempts to satisfy these commands under the constraint of keeping the robot balanced. To calculate the required velocities, a point-based *proportional* controller is used. This simple controller first provides an angular velocity until the robot is facing the target location. As the robot begins to face the correct direction the controller provides an increasing forward velocity, resulting in smooth acceleration toward the target point. The magnitude of the velocities supplied are a function of the distance between the desired pose and the current pose; which is what makes this a *proportional* controller. This attribute also causes the robot to slow down accordingly as it approaches the target point, which prevents it from overshooting. Additionally, each of these velocities are bound by a maximum value to ensure the robot behaves in a reasonably safe manner. The maximum value for the forward velocity is determined by the range at which the laser rangefinder is able to detect obstacles.

With this controller, the robot can follow along a given path by continually choosing the closest point along the path and navigating to that point. However, the frequent heading corrections required by this approach lead to a slow, rigid movement. In order to improve the efficiency of the robot's motion, the control system smooths the path using a *waypoint*-following approach. This is similar to the naive approach except that instead of choosing the closest point along the path, it chooses the farthest *visible* point along the path. A point was initially determined to be *visible* if the line between it and the robot's location did not intersect any part of the expanded region around unsafe cells. This modification substantially improved the navigational speed, at least when there where no obstacles near the path, as the robot drove directly toward the goal at maximum speed. However, whenever the path planner was forced to navigate around an obstacle, it consistently planned a path that was as close as possible to the object, as this is optimizes path length. In these cases, the system was unable to choose a *waypoint* that was more than one cell width away because all of the other points along the path were slightly occluded by the expanded obstacle. In order to produce smooth motion in these cases, the definition of a *visible* point was relaxed so that the robot was able to cut corners. A point was redefined as *visible* when the line between it and the robot's position did not intersect any cell that had **more than one** untraversable cell within a 3-cell square radius. Because this radius is a full cell width larger than required for safely navigating around obstacles, cutting off corners in this fashion posed no risk to the robot. The diagram in Figure 5.4 highlights the effect of

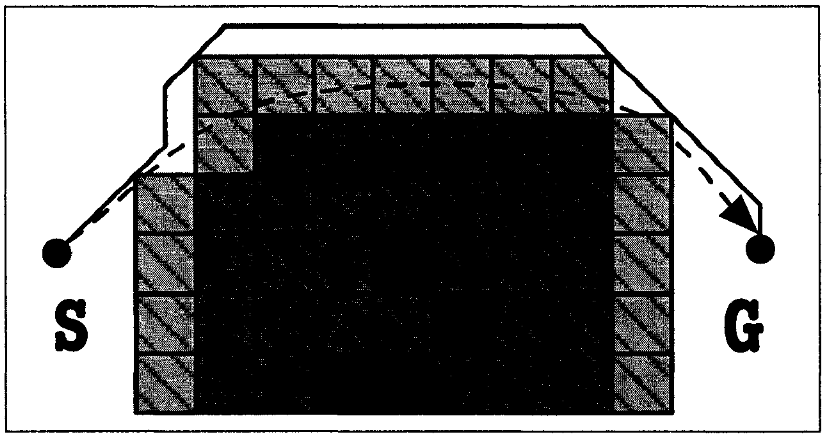


Figure 5.4: The differences between the original path produced by the D* Lite planner (solid line) and the resulting smooth path taken by the robot (dashed line). The solid dark cells have been marked as untraversable by the terrain classifier. The dark striped cells have more than one untraversable cell within the 3-cell radius. The lighter striped cells have only one untraversable cell within a 3-cell radius. The start position is indicated by the 'S' and the goal by the 'G'.

the modified path smoother. The way this modification was implemented, however, made it possible for the robot to cut too close to an obstacle that only occupies a single cell. But, because obstacles almost always occur in clusters naturally, and the slope classifier will always mark at least 2 cells as untraversable, this potential problem did not result in a single crash in any of the trial runs.

During navigation, the robot's constant movement results in new obstacles being continually added to the map. This causes the current path to become outdated and safe navigation requires that a new one be produced. Deciding when to produce a new path is a difficult problem because frequent planning is not only computationally expensive, but can cause the robot to oscillate between two different paths that have near identical cost. This oscillation is a critical issue because it can result in the robot indefinitely rotating back and forth in a small area. Conversely, planning infrequently can also be problematic, as the robot could navigate into a new obstacle in-between planning steps. To make the decision of when to replan, the system uses an interrupt-driven approach. The control system will continue to navigate along the current path until **any** of the following three conditions force a replan:

- The robot is closer to the current waypoint than to the path's starting point.
- A set amount of time has elapsed since the last replan.
- The current waypoint is no longer *visible*, due to robot motion or newly detected obstacles.

Under this replanning scheme the robot did not exhibit oscillating behavior because it remains on its current path unless there is a good reason to deviate. Additionally, this scheme reduced the demand on computational resources by avoiding redundant planning.

Chapter 6

Local Goal Search

This chapter describes the method in which the robotic system searches the local area, and visually identifies the goal object, after arriving at the target GPS location. The local search component is a necessary element because the GPS coordinate, provided as part of the geocaching problem, is assumed to have some degree of error. In the human geocaching setting, the goal object is generally identified by the fact that it does not fit with the rest of the environment; for example, a stuffed animal lying alone in a wooded area. However, this particular form of object recognition is highly complex and relies heavily on prior knowledge which unnecessarily complicates the problem. In order to simplify this visual goal identification, the robot is provided with a visual description of the goal object beforehand. For the experiments conducted in this thesis, the distinctive color and shape of the goal object were provided as the visual description. To detect the presence of the goal object in an image, a simple color-threshold classifier was employed.

While visual object recognition is an important component, much of the effort involved in solving the local search problem is related to effectively and efficiently moving through the environment until the goal object is identified. The first aspect of this navigation task is computing a path that will minimize the expected time required to locate the goal object. The calculation involves accounting for both uncertainties in the environment and the complex dynamics inherent in robot control. The complexity of this planning problem, combined with the limited computational power of the robot, necessitates the focus to be on the most simple and computationally efficient solutions. In addition to planning a path, the system must address all of the same perception and navigation issues discussed in Chapters 4 & 5. These include: producing control commands, updating the environment map, and deciding when to replan. Also, the system must continually assess the camera images to determine the presence of the goal object and eliminate regions that have already been scanned by the camera but do not contain the goal object. Section 6.4 provides an explanation of how the challenges were addressed to produce an effective goal finding system.

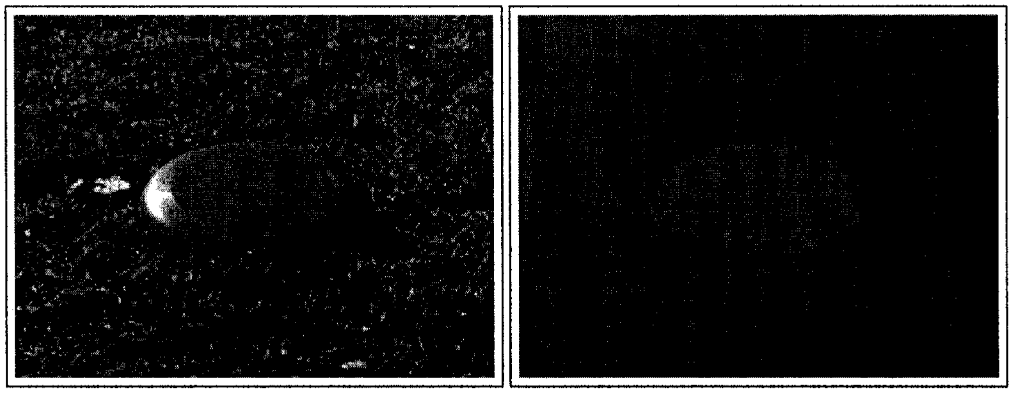


Figure 6.1: The goal object used for testing this system is shown in the left image. The thresholded image produced by the CMVision library is shown on the right.

6.1 Image Based Object Recognition

This section describes the problems associated with visual object detection in an outdoor environment and the mechanisms in which the Kato geocaching system addresses these problems. Visual object detection in an outdoor environment is problematic because such environments feature a range of colors and shapes that could be confused with the goal object. Also, uncontrollable environmental factors, such as overcast skies and shadows, can affect the ambient and specular lighting conditions which, in turn, can alter the appearance of the target object. Again, since robotic geocaching is a real time application, and computational resources are limited, the complex and highly specialized techniques developed for robust object recognition [30, 25] are too computationally demanding to be used. Ultimately, to make this problem tractable, a goal object that had an easily distinguishable color and shape (a red disc 28 cm in diameter) was selected.

To detect this goal object a modified version of the CMVision [4] color threshold library was employed. This software was originally designed for use with Robocup soccer: a real time robotics challenge that relies heavily on visual cues for navigation and localization. The software detects known objects by segmenting out the colors that fall within a given range and then measuring the size and shape of the resulting color blob. If the measured geometry resembles the goal object's known geometry, then a positive match is returned. This form of object recognition is robust, computationally efficient, and relatively straightforward to implement. The required color of the goal object was obtained by photographing the goal object under different lighting conditions, and manually selecting the matching pixels. The shape of the object was assumed to be an ellipsoid and the size (in pixels) was also chosen from sample images. A picture of the goal object, as well the thresholded image, are shown in Figure 6.1. This classification method was efficient enough to process images arriving at a rate of 3.75 Hz with minimal CPU usage.

6.2 Intelligent Local Navigation

Assuming it is possible to identify the goal object when it is in the camera's field of view, the next step is efficiently and effectively allowing the camera to scan the local area. This planning problem is significantly more complex than the problem of navigating to a single location, as discussed in Chapter 5. The added complexity results from the uncertainty surrounding the location of the goal and the introduction of the camera. The uncertainty regarding the goal location creates complications because it changes the definition of an optimal path. The search algorithm is no longer concerned with minimizing the traversal time to a single goal location but rather how likely it is to find the goal object along the way. In fact, the final endpoint is unlikely to even contain the goal because an optimal search path will be more focused on scanning areas with a higher probability of containing the goal. This specific problem has been largely overlooked by the research community and thus no solutions other than the brute force approach have been developed. The brute force approach, which involves enumerating all possible paths through the space, is intractable because the number of paths increases as an exponential function of the number of search cells. Additionally, the complexity of this problem is magnified by the fact that a camera is used to "scan" vertices, which forces the algorithm to compute a complex set of equations in order to determine the set of successor states for an action. This essentially amounts to calculating what areas of the terrain will be scanned by the camera, provided the robot moves to a given position. However, because the shape of the terrain is unknown, it is not possible to identify which part of the map the camera has in its view. Consequently, until the robot scans the terrain with the laser rangefinder it is not possible to calculate whether or not the object has been seen by the camera. The combination of these factors increases the state space to the point where the optimal path is too computationally challenging to compute and the solution must be approximated.

To develop a search method that fits within the aforementioned computational restrictions, it is necessary to simplify the problem of what the camera actually sees as the robot moves. To do this, the camera was placed on the robot so that the field of view would intersect the laser rangefinder's field of view. Figure 6.2 shows a precise model of the robot including the placement and field of view for the laser and camera sensors. Under this configuration, any terrain that is scanned by the middle third of the laser can immediately be marked as having been viewed by the camera. Additionally, because these sensors are configured to scan terrain prior to the robot traveling over it, the robot can search a given location by simply navigating to it.

At this point it is possible to produce a complete search algorithm by continually choosing any unseen point and searching it. This approach is also easily implemented as the subsystems necessary for effectively navigating to a static location have already been developed, as explained in Chapter 5. However, to improve the overall efficiency of the search a probability distribution is defined by assigning each cell in the discrete search graph a probability of containing the goal, denoted as $Pr(x, y)$. This allows the system to focus the search on the areas of the map that have a

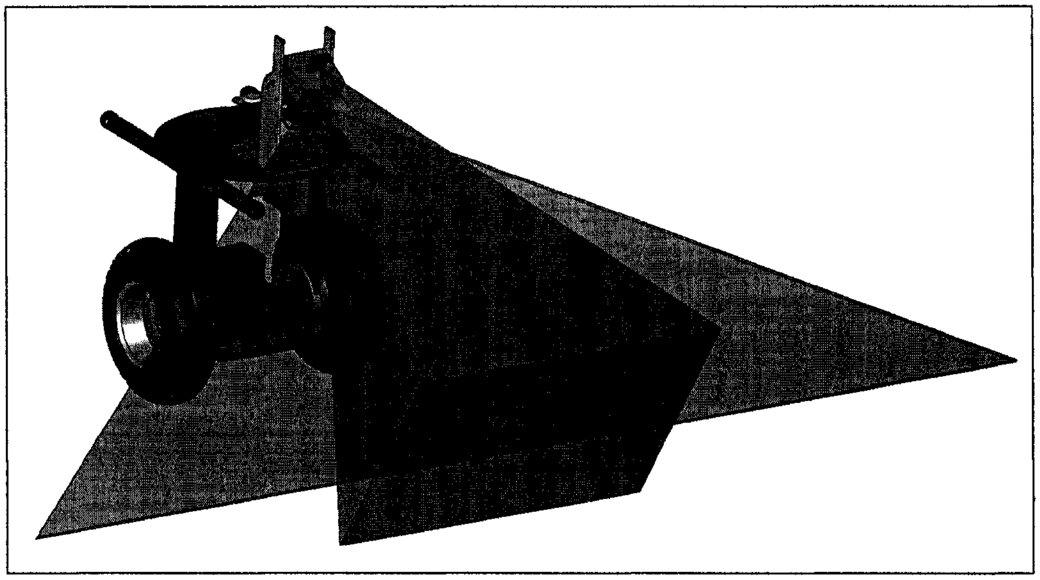


Figure 6.2: A to-scale rending of the Kato system is shown here. The field of views for the laser sensor (blue) and the camera (red) are designed to intersect so that the robot can easily identify what terrain has been scanned by the camera.

higher payoff, this distribution can be initialized arbitrarily but, in this thesis work, it was initialized using a normal distribution centered around the GPS goal location. Additionally, because travel time also factors into the efficiency of the search, a preference for points that are nearer to the robot was also introduced. To combine the information contained in the probability distribution with the preference for close points the following parametrized scoring function was used:

$$score(x,y) = Pr(x,y) * \frac{C}{C + dist(x,y)^{\alpha}}, \tag{6.1}$$

In this equation, $dist(x,y)$ is the length of the shortest obstacle free path to the cell, as determined by the path planner. The constant parameters C and α are used to trade off how much the system prefers closer points over points with higher probability. For all of the testing conducted, the constants used were: $C = 50$, $\alpha = 2.6$ and the variance of the normal distribution was set to $\sigma = 8.0$. These parameters were set by observing the robot's search pattern in test scenarios. Also, to avoid having to score an infinite amount of points, the search space was bounded by a set (8 metre) radius around the GPS goal location. Pseudocode for this final local search method is shown in Algorithm 3. Also, Figure 6.3 shows how the costs change as the robot navigates around the search area as well as a typical search path in a flat area.

6.3 Implementational Details

This section provides a more detailed explanation of the local search system's implementation. The pseudocode in Algorithm 3 is meant only to demonstrate the high level concepts involved and is referenced throughout this section. To begin, the search algorithm initialized the search grid with

Algorithm 3 : Local Goal Search

```
Procedure isValid(x,y)
1 :  if (dist(RobotPos,target) < DistThresh) return false
2 :  if (isSeen(target)) return false
3 :  if (!isTraversable(target)) return false
4 :  return true

Thread 1
5 :  while (!GoalFound)
6 :    GoalFound ← CMVCheckImage()

Thread 2
7 :  while (!GoalFound)
8 :    target ← $\arg\max_{x,y} score(x, y)$
9 :    while (isValid(target) AND !GoalFound)
10 :      Navigate toward target, marking scanned cells as seen.

Procedure Main()
11 :  GoalFound ← $false$
12 :  join(Thread 1, Thread 2)
```

the probability values that correspond to the predefined distribution. This was accomplished by iterating over all of the cells within 8 metres of the GPS goal and setting each cell's probability according to a normal density function. After the initialization step, the algorithm enters the loop that continually chooses, then navigates to, the cell with the highest score. The most challenging part of this stage was to calculate the distance to each cell, in a computationally efficient manner, so that it could be accounted for in the computation of the score function. The distance values were computed based on path planning rather than euclidean distance. This method was used for two main reasons: firstly, if a cell is surrounded by obstacles which cause it to be unreachable, it should not be selected; secondly, in the case where the search area is separated by a very wide object (e.g. wall or curb) the resulting inefficiencies from continually navigating around it must be taken into account.

To calculate the distance value for a cell, the algorithm used a modified version of the A* search algorithm. This modified algorithm was configured to keep the open and closed lists when the target changes, so that it could reuse the work done by previous searches. When the robot's position alters, the open and closed lists are cleared since their values are no longer correct. In order to account for the additional costs involved with changing direction, the distances were calculated starting from 1 metre in front of the robot. This planning location is shown in Figure 6.3 as a light blue X, the other light blue X marks the cell with the maximum score. Also, because the camera and the laser rangefinder cannot sense the terrain too close to the robot, the search does not consider cells that are within 1.2 m of the robot. This unconsidered region around the robot is clearly visible in the middle image in Figure 6.3. Once a cell has been chosen, a path to the cell is planned using standard A*

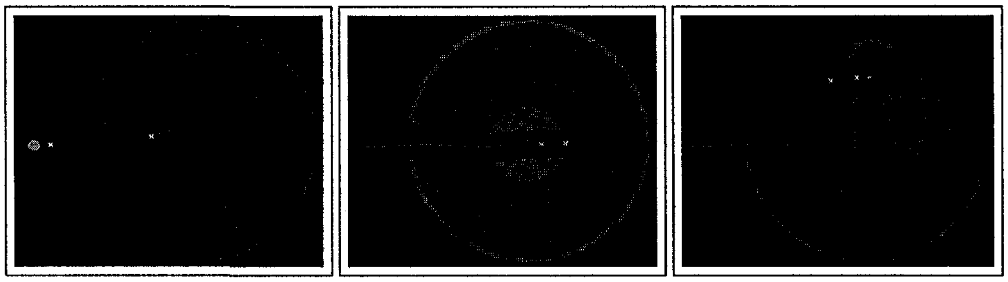


Figure 6.3: The progress of the local search procedure in flat obstacle-free terrain is show. From left to right the images show the progress of the robot at approximately 5 seconds, 15 seconds, and 3 minutes. The robot is drawn as the small gray circle and the path it has taken is shown as the solid purple line. The search area is shown as a collection of red, blue, and pink cells. The more red a cell is the higher its score, and the more blue the lower the score. The pink cells have a score that is zero either because of floating point precision or because the cell is deemed too close to the robot. As cells are viewed by the camera they are removed from the search grid.

search, and the original controller and waypoint subsystems are used. The robot will continue to navigate toward the selected cell until any of the following conditions are met:

- The cell is seen by the camera.
- The cell is marked as an obstacle or unreachable.
- The robot has moved to within 1.2 metres of the cell.

These conditions are also shown in the **isValid()** procedure in Algorithm 3. When any of these conditions are met the search point is rejected and a new one is chosen according to the scoring function. The navigation loop is also terminated if goal object is detected by the camera.

6.4 Evaluation

Because the local goal search process is somewhat of an *ad hoc* solution it becomes necessary to provide empirical evidence regarding its performance. To evaluate the local search subsystem 15 test runs were conducted in environments with ranging difficulties. The performance metric used in this evaluation included the number of goal finding successes as well as the amount of time taken to find the goal object. All the tests were conducted in an outdoor park environment and a different start and goal location were used for each run. The tests were equally divided between three scenarios, each with a different level of difficulty. The level of difficulty was determined by the percentage of obstacles covering the search area; these levels were approximately 1%, 25%, and 50%. These obstacles make the search more challenging because they necessitate more complex path planning as well as occlude the goal object. For each of the 15 tests the robot was initially placed 10 metres from the GPS location and the goal object was placed in a random location within an 8 metre radius of the provided GPS coordinate. The average search times and number of successes for each environment,

Environment	average time	success rate
$\approx$ 1% obstacles	69 seconds	5/5 (100%)
$\approx$ 25% obstacles	68 seconds	5/5 (100%)
$\approx$ 50% obstacles	298 seconds	4/5 (80%)
Overall	143 seconds	14/15 (93%)

Table 6.1: Results for the local goal search tests.

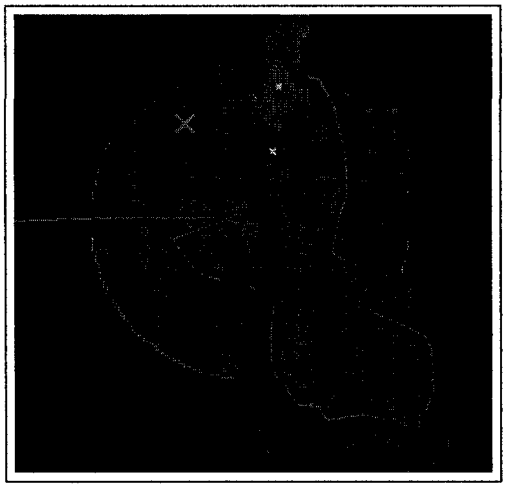


Figure 6.4: The local search behavior for a single run in the $\approx$ 50% obstacle environment is shown. The obstacles detected by the laser are drawn in red while the local search scoring function is shown as small blue-red tiles. The actual location of the goal is marked with the green X and the robot is shown as a gray circle. This image was captured at the point where the robot identified the goal object.

as well as the overall performance, are shown in Table 6.1. Also, an example test run in a $\approx$ 50% obstacle environment is shown in Figure 6.4.

The most positive result of these trials was the low average search time. Considering an environment with 50% obstacles is quite rare, it is fair to say that the local search system will usually find the goal in under 2 minutes. The low search times are mainly due to the fact that the goal object is rarely occluded and thus can be seen from almost anywhere in the search space. In fact, in several tests the robot was able to identify the goal at distances up to 10 metres. In addition to the low search times the system achieved an overall success rate of 93%. In the one failed test the visual object recognition incorrectly classified a patch of red leaves as the goal object. This failure highlights the need for a more sophisticated method of object recognition.

Chapter 7

Experimental Results

To evaluate the performance of the Kato geocaching system, several full system tests were conducted in two distinct outdoor environments. A full system test involved placing the robot at a random location in the environment, with the goal object several hundred metres away. The test was complete when the robot safely navigated to the given GPS location, searched the local area, and identified the goal object. To add randomness into the location of the goal object, its GPS coordinate was recorded with a handheld GPS receiver, which had an error of approximately 8 metres. During each of the test runs a number of parameters were recorded, including: the total time elapsed, the distance traveled, the path taken, and the number of successes (locating the goal object). Section 7.2 presents these values as well as some anecdotal results.

The Kato geocaching system's performance was compared to the performance of a human expert teleoperating the robot. To allow meaningful comparison, the human operators were shielded from the environment and were only given a visualization of the robot's sensor readings. The objective of this testing was to allow human operators to assume the higher level perception and planning aspects of the system. This was to ascertain whether a human would be able to plan a more efficient path to the goal, identify obstacles more effectively, or conduct a more intelligent local search procedure compared to the programmed system. The results of this comparison are presented in Section 7.3.

7.1 Environments

Two distinctly different outdoor environments were used for the full system tests. Photographs of these environments are shown in Figure 7.2. The first test environment was a lightly wooded park near the University of Alberta campus. This park spanned a maximum distance of about 850 metres and consisted of two large forested areas separated by a small access road. Some common obstacles in this environment included: picnic tables, trees, small shrubs, fallen trees, barbecues, large berms, and curbs. Additionally, the access road which dissected the park was bordered by a curb on both sides, which could only be crossed safely by the robot in four places. The crossing points were fairly evenly spaced along the curb, with each access point being less than 1.5 metres

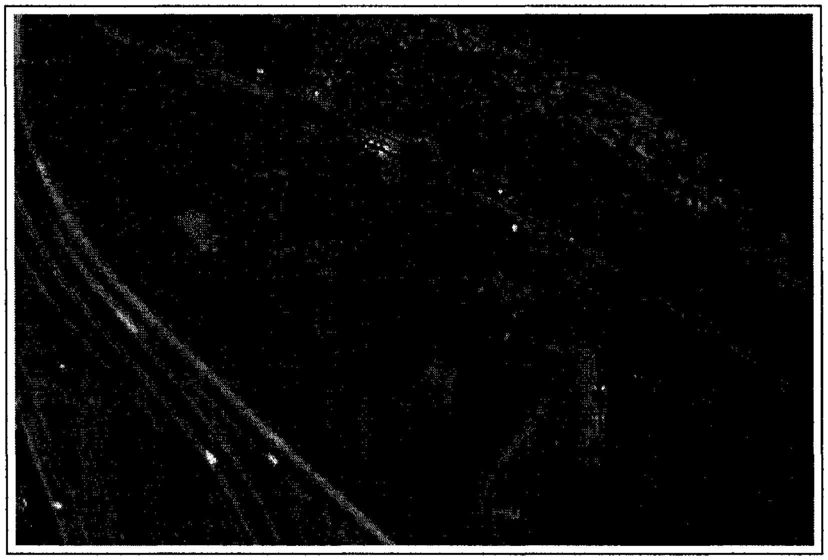


Figure 7.1: A satellite image of the park environment, the curbs are marked out in red and the crossings highlighted with blue ellipses. Image courtesy of Google Maps^{TM}.

wide. Correctly classifying, and navigating through, the crossing points was a critical challenge in this test environment; misclassifying a crossing as unsafe would result in the robot having to navigate approximately an average of 350 metres before encountering another crossing. These crossing are highlighted, along with the curbs, in the satellite image of the park shown in Figure 7.1. The presence of mixed vegetation in the test environment also added to the complexity of navigation by producing variable lighting conditions, which complicated the visual identification of the goal object. However, the actual park terrain, which did not include the curbs, was not as challenging to navigate as a completely undeveloped area because the grass and other vegetation was trimmed and obstacles were relatively sparse.

The second environment used to evaluate the Kato geocaching system's performance was the 2 km wide, semi-urban, University of Alberta campus. This environment featured a number of natural and man made obstacles including: trees, bicycle racks, garbage cans, benches, wheel chair ramps, curbs, buildings, flower gardens, chain link fences, and hills. The main challenge of navigating in this environment was finding a path to the goal around a large complex of buildings, fences, and gardens. These obstacles were similar to the curbs in the park environment in that they were impassable but different in that they were typically much larger, contained more inlets, and were more difficult to detect with the laser. These environmental characteristics resulted in the robot frequently backtracking in order to reach the goal location. Again, like the park environment, the majority of the vegetation in this environment was trimmed, which allowed the robot to completely avoid dense vegetation and still find a path to goal.

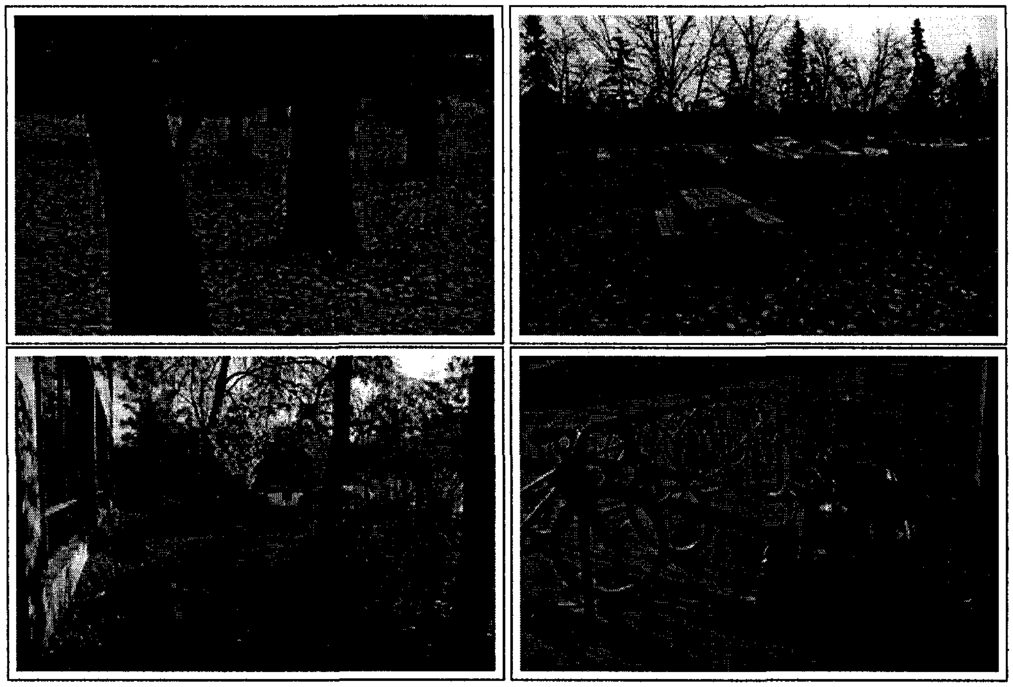


Figure 7.2: Images of the two test environments are shown; the top two images are taken in the park and environment and the bottom two images are from the campus environment.

7.2 Trials

In total, 17 different full trial runs were conducted in the test environments. The semi-urban campus environment was used for 2 long distance runs and the park environment was used for the other 15 runs. Each trial began by placing the robot at a random location in the test environment and providing it with the GPS coordinate of the goal object. To minimize the potential of the researcher to bias the trial through the selection of the start and end locations, they were chosen from a low detail map ahead of time. This prevented the researcher from accurately discerning the difficulty of the run while still being able to place the robot and object in valid locations (i.e. not on top of a hazard). Additionally, to ensure the robot was provided a randomized coordinate, a hand held GPS receiver, which has an average measurement error of +/-8 m, was used to record the final position of the goal object.

During each test run, the following parameters were recorded: path taken, distance traveled, total time, goal finding success, and the number of interventions required. An intervention was defined as any time the human testers overtook control of the robot to prevent it from navigating into an unidentified obstacle or to get the robot out of a situation where it was no longer able to move. Also, all of the parameters remained constant throughout testing with the exception of the terrain grid based z estimate. Of the 17 tests, 9 were conducted without the use of this estimate. This was done to independently evaluate the affect of this parameter; which is discussed in Section 7.2.2. The results recorded for all 17 trial runs, split between the two environments, are summarized in Table

	Park	Campus	Total
Trials	15	2	17
Successes	14	2	16
Goal Finding Fails	1	0	1
Interventions	2	2	4
Cumulative Distance (m)	4165	2342	6507
Straight Line Distance (m)	3434 (82%)	1060 (45%)	4494 (69%)
Cumulative Time (min)	216	131	347
Average Speed (m/s)	0.321	0.298	0.313

Table 7.1: Results for the autonomous system on the trial runs.

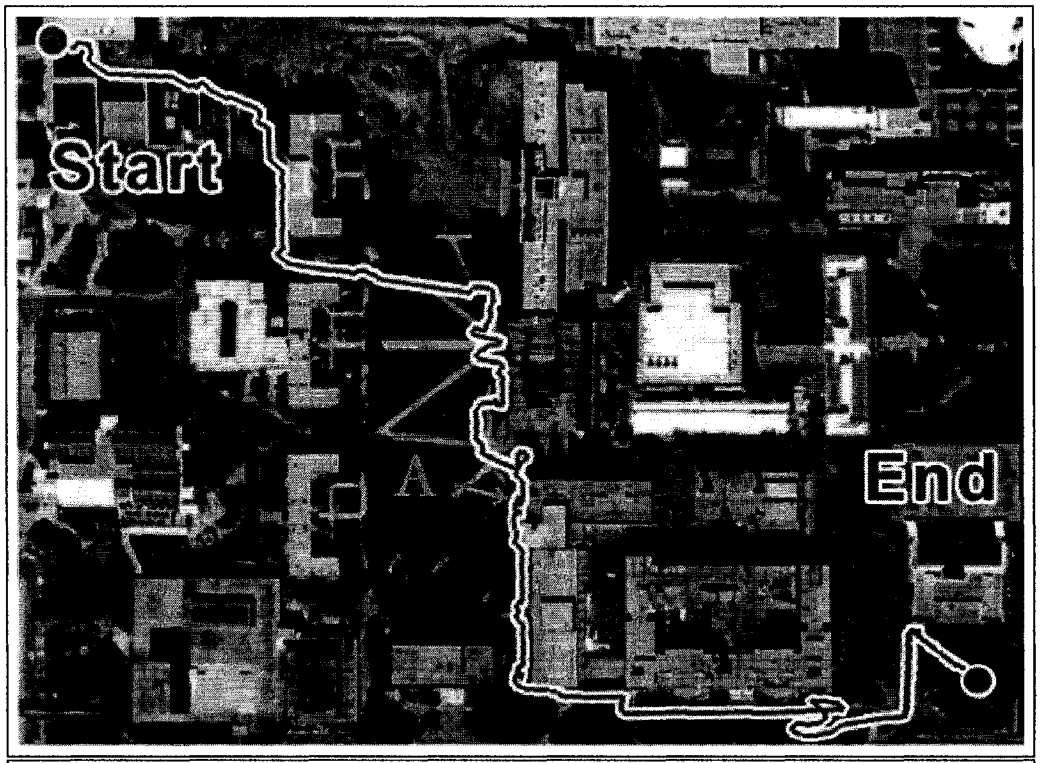
7.1.

7.2.1 Analysis

The overall goal finding success rate of the Kato geocaching system was high. The robot was able to successfully navigate to and identify the goal object 16 out of 17 times (94%). The single goal finding failure resulted from the local search system mistakenly classifying a patch of red berries as the goal object. This result is consistent with the results obtained during the independent local search tests, which were presented in Chapter 6. During the navigation portion of the trials, only 4 interventions were required over the entire 6.5 km. This is impressive given the difficulty of the terrain, which is indicated, in part, by the difference between the cumulative traversal distance and the straight line distance. This measure provides insight how the obstacles in the environment affected the path taken by the system. Additionally, the path-planning and control subsystems performed efficiently during the test runs as an average speed of 0.313 m/s, or 78% of the maximum speed, was maintained. The paths shown in Figure 7.3 illustrate the solid performance of the planning and control systems and illustrate the complexity of the campus environment.

The high accuracy of the terrain classifier was illustrated by both the minimal number of interventions and the system's ability to correctly categorize critical sections of terrain. For instance, the small breaks in the curb, that encircled the two park areas, were navigated through in 9 separate instances during the 15 test runs without a single misclassification. Also, in the campus environment, the system was able to successfully classify a wheelchair access ramp shown in Figure 7.4 as traversable. This was a difficult section of terrain to correctly classify because it is less than 2 metres wide, included a sharp change in direction, and had a significant downward slope. Furthermore, correctly classifying this ramp was a crucial to the performance of the system because, had it been misclassified, the next shortest path would have been over 1 km longer.

Of the four interventions necessary during the 17 full system trials, two occurred in the park environment and were a direct consequence of assuming a constant robot height; these interventions are discussed in Section 7.2.2. The other two interventions occurred in the campus environment. The locations of each of these two interventions are marked by green arrows in Figure 7.3. The first of these, labeled A in the figure, was the result of the robot becoming suspended on a tree while



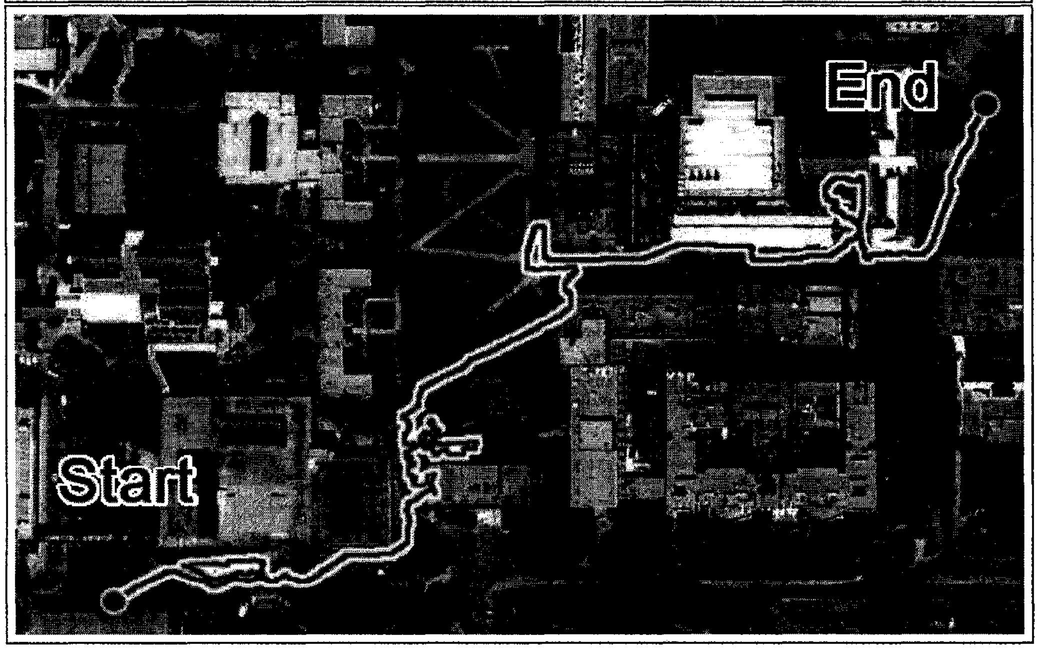


Figure 7.3: Each image in this figure shows the path taken by the robot in the campus environment. The green arrows mark the locations of the human interventions. The path in the top image was a total distance of 1.1 km and took 66 minutes to complete. The Path in the bottom image was 1.2 m and took a total of 65 minutes to complete. Map images courtesy of Google Maps^{TM}

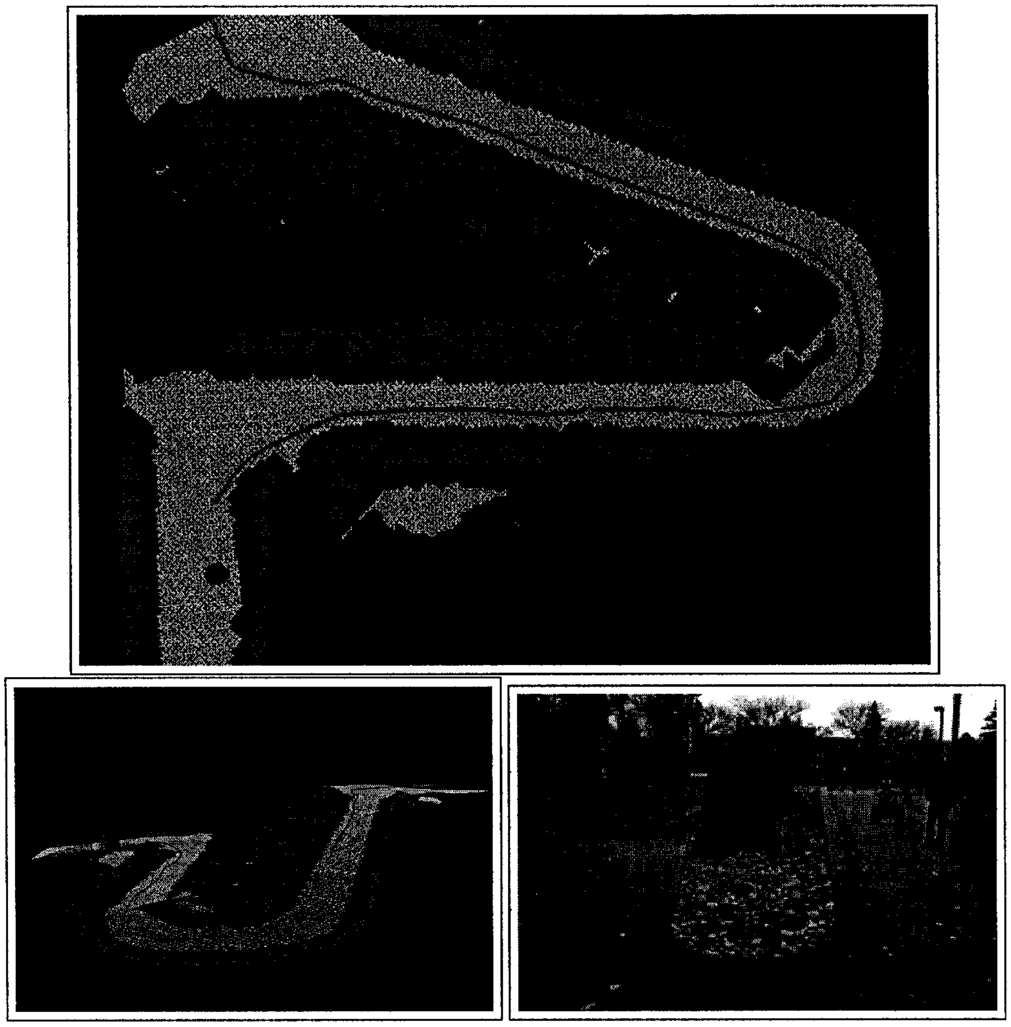


Figure 7.4: A rendering of the terrain map and a photograph for the bike ramp section of the second campus run (labeled a 'C' in Figure 7.3) are shown. The top image is a top-down view of the terrain grid, untraversable terrain is red, the buffer zone is shown in blue, and the drivable areas are shown in green. The bottom left image shown the same grid from a South facing view. A photograph from the same direction is shown in the bottom right.

attempting to regain its balance after traversing over a small tree root. This situation occurred when the robot was required to provide additional torque to drive over an obstacle which resulted in the RMP platform leaning forward to counterbalance the forces applied to the wheels. Subsequently, once the object was overcome, the robot was forced to accelerate quickly in order to regain its balance and was not able avoid a tree in its path. This effect was observed several times during testing but only once did the robot become immovable. The second intervention, labeled B in the figure, was caused by the robot navigating into a thick branch that did not enter the laser's field of view until it was too late to stop. In this case, the testers stopped the robot to prevent it from damaging the laser sensor and restarted the robot about a metre back.

7.2.2 Effects of z-estimate

Table 7.2 summarizes the results of the 15 full system tests conducted in the park environment. This table is split between runs where the system attempted to estimate the elevation of the robot and runs where the elevation was assumed to be zero at all times. The most significant feature of these results were the 2 interventions that resulted because of the assumption of a constant elevation. This assumption ultimately resulted in an underestimation of the slope of a hill which would have otherwise been classified as untraversable. More specifically, because the robot's change in height was subtracted from the terrain grid, the slope was underestimated and the robot attempted to drive up a hill that was not traversable. See Chapter 4 for a detailed discussion on the root of this problem. In both instances the robot was not able to summit the hill and the human testers manually moved the robot over the hill and restarted the system. Three more trials runs were conducted on the same hill region with the terrain based z-estimate turned on. In these cases, the robot was able to correctly label the terrain as untraversable and was able to find a safe path to the goal without intervention. Aside from the interventions, the z-estimate did not significantly affect the behavior of the robot; the average speed was within a margin of error and no differences where observed by the testers.

	$z = 0$	z-estimated
Trials	9	6
Interventions	2	0
Cumulative Distance (m)	3370	1554
Straight line Distance (m)	2093 (62%)	1341 (86%)
Cumulative Time (min)	179	79
Average Speed (m/s)	0.314	0.326

Table 7.2: Results with and without the z-estimate, for the park environment only.

7.3 Comparison to Human Teleoperation

Five of the autonomous trial runs were rerun with humans teleoperating the robot. To standardize the comparison, and ensure that the decision making ability was parameter tested, both the human ex-

	Robot	Humans
Trials	5	5
Successes	5	5
Goal Finding Fails	0	0
Interventions	0	3
Cumulative Distance (m)	1517	1541
Cumulative Time (min)	76.9	83.2
Average Speed (m/s)	0.329	0.309

Table 7.3: Results for the human expert trials.

perts and the autonomous system received the same information about the environment. To achieve this, the human controllers were not permitted any visual or tactile contact with the environment or given any prior knowledge about the test site. Instead, the only environmental data the human operators had access to was a 3D visualization of the robot's sensor information. The visualization provided information on the position of the robot and the GPS goal, the 3D point cloud produced by the laser sensor, and the system's automated terrain classifications. The terrain classifications were provided to the human controllers because the point cloud representation was, at times, difficult to interpret. However, the human operators were encouraged to rely on the point cloud information as much as possible, due to possible inaccuracies in the terrain classifications. This was done to more effectively evaluate the ability of the human testers to perform laser-based obstacle detection. Control commands were provided to the robot using a off-the-shelf 2-joystick gamepad. To conduct the localized goal search, testers were told the GPS goal was off by a maximum of 8 m and were instructed to navigate around until the visual goal detector found the goal. The automatic goal detection subcomponent was used to prevent the human testers from unfairly using the images to navigate.

Also, to limit the use of knowledge acquired from previous tests, two different human experts were used during the trials. Each human tester completed a short practice run to ensure they were familiar with the interface and controllers. The results for both the human testers and the autonomous system are shown in Table 7.3.

The difference in the cumulative distances traveled for both of the test groups (human teleportation and autonomous robotic control) was within an expected margin of error. This suggests that the autonomous robot and the human teleoperaters preferred similar paths and classified many obstacles similarly. The autonomous system achieved a faster average speed than the human controllers. Although this difference can be attributed to the additional time required for the humans to infer the shape of the terrain from the point cloud data. The major discrepancy between the two tests groups was the number of interventions; the autonomous system did not require a single intervention over the five test runs while the human controllers required three. In all three of these cases, the human controller thought the terrain appeared safe, overruled the automatic classifier, and encountered an obstacle. While the policy of overruling the terrain classifier did not pay off in these cases, in

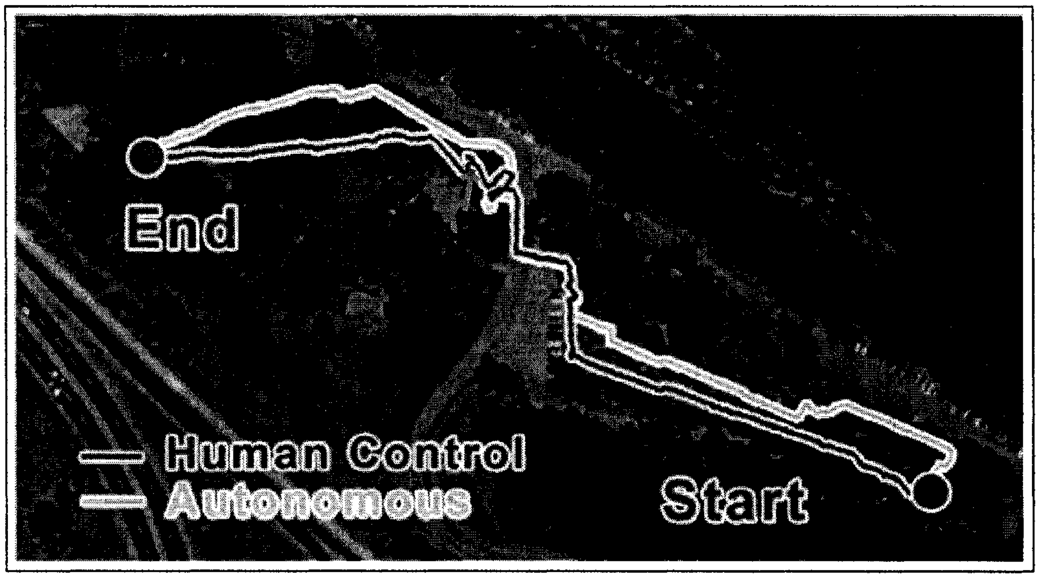


Figure 7.5: This image shows the path taken by both the autonomous system and the human controller for the same task. The green arrow marks the location where an intervention was necessary for the human controller. Map image courtesy of Google Maps^{TM}.

several instances the human controllers were able to navigate through very narrow gaps between obstacles, where the autonomous system would not have. This allowed the human controllers to reduce the overall distance traveled. The paths in Figure 7.5 illustrate the method in which the human controllers were able to find a shortcut where the autonomous system did not. However, in this example, the shortcut was through rougher terrain, which resulted in a lower average speed and an intervention. The human controllers were more effective at recognizing traversable vegetation (e.g. grass) from the point cloud compared to the autonomous system and thus did not waste time navigating around it. From these results it is clear that the human obstacle detection was different from the autonomous system's, but not necessarily more or less accurate. In general, it can be concluded that the autonomous system performed on a level similar to, or slightly above, the human controllers, which is impressive considering the additional prior knowledge about commonalities in outdoor environments that the human controllers had at their disposal.

7.3.1 Comparison To Other Systems

This section compares the results collected for the Kato geocaching system with those collected for similar outdoor navigation systems. Of note, these systems employ different sensors and robotic platforms and have slightly different objectives and testing environments and, as such, any direct comparisons are limited. However, general comparisons can be made by contrasting the number of interventions and the average speed of the robots, while accounting for differences in the robotic platforms and the environments.

LAGR

The results of the test runs for the Kato geocaching system can be compared to the results obtained during the 13 DARPA LAGR Phase I test runs conducted from March 2005 to May 2006 [18]. The DARPA tests were similar to the Kato geocaching system in that the robot was placed in an outdoor environment, provided with a GPS goal location, and required to navigate autonomously to the goal. However, due to the LAGR project's focus on online learning applied to far field vision, the testers introduced a significant amount of structure to the test environments in an attempt to simplify the learning problem. The decision to add structure was also likely a result of the low success rate in early unstructured environments; specifically, only 3 out of 27 runs were successful in this type of environment. In any event, the courses were made easier by adding objects such as metre high orange fences, straw bales, and colored shale to mark paths. Even though this additional structure aided the participating robots in locating the goal, the goal finding performance was poor. In this simplified environment, the DARPA robots achieved an average success rate of only 59%. In comparison, the success rate obtained for the Kato geocaching system in a less structured, and therefore more difficult, environment was 93%. The discrepancies between the DARPA and the present system are likely due to the disadvantages caused by constraining the available sensors on the LAGR robot. More specifically, the LAGR robots used stereo vision as a primary terrain sensor which, in comparison with laser rangefinder sensors, is less reliable.

NAVLAB II

The performance of the Kato geocaching system can also be compared to the CMU NAVLAB II system, which was similar in that it utilized a laser rangefinder in the same "push-broom" configuration [40] and was tested in a similar outdoor environment. However, the NAVLAB II system was not as rigorously evaluated; the majority of the published results for this system were anecdotal. These anecdotal results indicated that the system was not very robust, with the major issue being that the robot was unable to turn around (backtrack) without human intervention. The authors did, however, publish the full results for a single 1410 metre long run, in which the CMU NAVLAB II robot required 6 human interventions. Three of these interventions were the result of the robot's reduced mobility due to its kingpin steering system, while the other three interventions were the result of terrain misclassifications.

PerceptOR

The robotic systems that took part in the final DARPA PerceptOR project [23] provide the most useful comparison to the Kato geocaching system due to the similarities in test environments and sensors. However, the differences between the structure of the systems used in the PerceptOR project and the system presented in the thesis are notable. In general, the robots competing in the PerceptOR project were much more sophisticated; these robots used multiple laser sensors mounted on motorized pan tilt units, multi-view image systems, high end video cameras, and, in the case of CMU's entry, an external autonomous aerial drone equipped with a camera and laser sensor to extend sensory range [21]. Also, the test environments in the DARPA PerceptOR competition were selected from undeveloped, outdoor, forested environments and therefore necessitated the use of vegetation detecting software. A series of global waypoints marking the desired path were also provided to the competitors ahead of time.

The participating PerceptOR robots are the most extensively evaluated unstructured outdoor navigation systems to date. The trial runs conducted during the final tests summed to a total distance of 130 km over a 110 hour length of time in 6 different environments. However, because not all of the robotic systems were tested in all of these environments, only the results obtained for the most tested, and highest performing, system, the CMU robot, are considered here. This system was able to navigate a total distance of 81 km over a 3843 hour period and achieved an average speed of 0.35 m/s [21]. Over the course of these runs the system required 121 "hard" interventions where the robot did not detect a dangerous situation and required an emergency stop. Additionally, this system had to stop and ask the human operator for assistance a total of 519 times; these situations would classify as interventions based on the definition used for the geocaching tests. On the same test runs the human teleoperators, which used video to navigate, were able to achieve up to three times faster average speeds with less than one tenth the number of interventions. This result is notable because the human teleoperators in the geocaching system tests were not able to achieve a higher level of performance as the autonomous system. However, this discrepancy could be attributed to the fact that the human teleoperaters were able to use video to navigate in the PerceptOR tests, whereas they were forced to use laser information in the geocaching tests.

As mentioned previously, differences in the test environments limit direct comparison between the CMU PerceptOR robot test results and the geocaching system. Although the high number of interventions, in respect to the distance traveled as well as in comparison to the human operators, indicates the PerceptOR robot is not particularly robust in its test environments. The Kato geocaching system had significantly fewer interventions per kilometre traveled (0.55) than the CMU PerceptOR robot (7.9) and indicates that the system was able to function more autonomously in an outdoor environment. However, future study in a more complex outdoor environment would be required to draw a definitive conclusion.

Chapter 8

Limitations and Conclusions

This final chapter reviews the major limitations of the system and presents modifications that could be used to address them. Most of these limitations are in regard to robust obstacle detection as it is both difficult and vital to the performance of the system. This chapter also provides some concluding remarks on the performance of the geocaching system and how its development contributes to the field of autonomous outdoor navigation.

8.1 Limitations

8.1.1 Laser Mount

The most significant limitations of the Kato geocaching system arguably result from the employment of a single, rigidly mounted, laser rangefinder as the only terrain sensor. With this "push-broom" configuration, the system is not always able detect obstacles early enough to avoid them. Off-ground obstacles like tree branches, fences, and picnic tables are good examples of obstacles the robot has difficultly avoiding. In fact, one of the two unaddressed interventions in the full system tests was due to this limitation. Figure 8.1 illustrates scenarios in which off-ground obstacles are first detected, and it is clear that it is often dependent on their height and point of contact with the load bearing surface.

Another difficulty with this sensor configuration becomes apparent when the pitch of the robot differs from the slope of the terrain. This occurs frequently as the RMP constantly alters its pitch in order to maintain its balance. In such scenarios the altered pitch of the laser sensor prevents the system from effectively mapping the terrain. This effect is most evident in situations where the robot is traversing down-sloped terrain. In these cases, the robot will remain relatively upright while the terrain in front of it continues to slope downward. Consequently, the laser scans the ground at an extended distance and is therefore more susceptible to errors in the localization estimate and is less able to detect obstacles.

To alleviate the problems resulting from a rigidly mounted laser rangefinder sensor, the sensor suite could be reconfigured to increase the viewable range. This can be done by converting the rigid

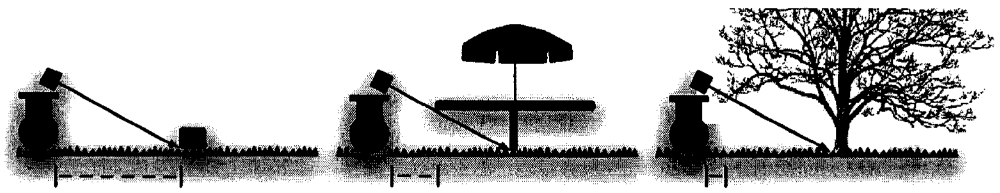


Figure 8.1: Three examples of how off-ground obstacles are detected at different distances ahead of the robot are illustrated. The dashed bracket below the ground indicates the distance at which these obstacles are first seen.

laser mount into a motorized pan-tilt mount that would alter the pitch of the sensor as necessary. Additionally, adding terrain sensors, such as multi-view cameras or laser rangefinders, would increase the viewable range. With an increased field of view the system would be more able to effectively detect off-ground obstacles and measure down-sloped terrain.

8.1.2 Inaccurate Ground Plane Estimate

The Kato geocaching system estimates the height of the load bearing surface by averaging laser readings over a discrete area. This estimate can be incorrect when terrain features occlude the true load bearing surface (e.g. snow, grass, twigs, leaves, water, dust). In these situations, the robot's inability to accurately classify the terrain can lead to inefficient paths and/or crashes. However, because some of these terrain features exhibit exploitable characteristics, for instance grass appears porous when scanned with a laser sensor, improvements to the load bearing surface estimate can be achieved. For example, these features have been successfully used by other systems to improve the load bearing surface estimate [46]. Additionally, some active sensors, such as radar, have been shown to have increased ability to penetrate occluding terrain and thus provide additional information to the system, which can then be used to improve the load bearing surface estimate [44]. However, these approaches are limited in that semi-transparency does not always mean that the obstacle is not load bearing, for example, a chain-link fence. Ultimately, robust methods for addressing this limitation have yet to be developed, which makes estimating the height of a load bearing surface an active area of research.

8.1.3 Limited Sensor Range

Another significant limitation of the Kato geocaching system is the limited range of the terrain sensor, which results in a lower top speed and less efficient paths than could be achieved with a increased range. A lower top speed results because the robot, for safety reasons, will only travel at speeds where the stopping distance does not exceed the look-ahead of the terrain sensor. Less efficient paths result because a robot with a shorter sensor must plan paths with reduced information about the environment (compared to a robot with an increased sensor range). For example, when Kato enters a cul-de-sac, it explores the entire region before backtracking. If the robot was able to determine the traversability of terrain at extended distances, it could avoid many of these scenarios.

A significant amount of research has focused on improving the sensory range of laser-based mapping systems and many of these techniques could be applied to this system. For example, using a camera sensor as an early warning system improved the top speed of the DGC winning "Stanley" robot [41]. Using available satellite imagery has also resulted in the increased navigation efficiency for a mobile robot in similar outdoor environments [37]. Again, adding additional terrain sensors or mounting the current laser rangefinder on a motorized pan-tilt mount would also increase the sensory range.

8.1.4 Binary Terrain Costs

Another limitation of the Kato geocaching system is its inability to favor smooth terrain over rough terrain because, once classified as safe, all terrain has the same cost. Navigating through rougher terrain is less desirable because it applies greater friction and is more likely to contain obstacles, both of which decrease the average speed of the robot. Rough terrain can also result in excessive motion (bouncing) of the robotic platform, which can lead to increased localization and mapping errors.

To address this concern, the classification subsystem could be configured to avoid rougher terrain by lowering the height and variance thresholds. This could, however, cause the robot to behave inefficiently in cases where navigating over a small patch of rough terrain is worthwhile. A more effective approach would be introducing a preference for smoother terrain by adding additional cost measures based on terrain roughness. This could be accomplished by assigning cost as a function of a cell's variance and neighbouring height changes. Also, self-supervised machine learning methods have been successfully applied to estimate the roughness of the terrain with both color camera images [1] and laser rangefinder measurements [38]. Under this type of cost-based classification scheme, the robot would prefer smoother terrain, but could choose to navigate over rougher terrain when it is efficient to do so.

8.2 Conclusion

In summary, the objective of this thesis project was to develop a fully autonomous robotic system that is able to consistently and efficiently solve the outdoor geocaching problem. A simple, cost-effective robotic system capable of effectively solving this problem could be the foundation for enumerable application areas, such as autonomous search and rescue, surveying, transportation, and interstellar exploration. Despite progress in the development of autonomous robots through targeted research initiatives, there is still significant research required before they become useful for practical applications.

The challenges involved in the development of an autonomous robotic system include aspects of localization, mapping, path-planning, robot control, and visual object recognition. To meet these challenges, a highly maneuverable self-balancing Segway RMP was outfitted with various sensors and modifications to allow for fully autonomous navigation. Information from these multiple sensors was combined to address the localization problem and produce a position estimate with low relative error. The position estimate was used to fuse multiple terrain measurements, provided by the laser rangefinder, into a congruent map of the environment. The map summarized the laser measurements in a compact representation that was sufficient for obstacle detection; these detected obstacles were avoided by use of a sophisticated and computationally efficient path-planning subsystem. A waypoint-based control system was used to produce smooth motion along the planned path through rough outdoor terrain. Finally, this system utilized a novel approach for searching a target area to visually identifying the goal object.

The methods developed for the robotic navigation system, including the unique combination of sensors and robotic apparatus, were evaluated in multiple outdoor scenarios. Results were compared to those obtained through human teleoperation of the robotic system and for systems developed during other robotics studies. Overall, there was convincing evidence that Kato is able to effectively solve the geocaching task and is able to do so as effectively as a human or any previously developed systems, at least in the test environments. Moreover, this thesis project has resulted in an increased understanding of the various challenges that must be overcome in order to act intelligently in an outdoor environment.

Bibliography

[1] A. Angelova, L. Matthies, D. Helmick, and P. Perona. Dimensionality reduction using automatic supervision for vision-based terrain learning. In *Proceedings of Robotics: Science and Systems*, 2007.

[2] E. T. Baumgartner, H. Aghazarian, and A. Trebi-Ollennu. Rover localization results for the FIDO Rover. In *Sensor Fusion and Decentralized Control in Autonomous Robotic Systems*, 2001.

[3] Deborah Braid, Alberto Broggi, , and Gary Schmiedel. The TerraMax autononomous vehicle. In *Journal of Field Robotics*, 2006.

[4] J. Bruce, M. Veloso, and T. Balch. Fast and inexpensive color image segmentation for interactive robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 00)*, 2000.

[5] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The 2005 DARPA Grand Challenge*. Springer, 2007.

[6] National Robotics Engineering Center. Learning applied to ground robots (LAGR) - robot design, 2008. http://www.rec.ri.cmu.edu/projects/lagr/index.htm.

[7] DARPA. Grand challenge I, 2004. http://www.darpa.mil/grandchallenge04/.

[8] DARPA. Grand challenge II, 2005. http://www.darpa.mil/grandchallenge05/.

[9] DARPA. Learning applied to ground robots (LAGR), 2008. http://www.darpa.mil/ipto/programs/lagr/lagr.asp.

[10] H. L. Durrant-Whyte and I. J. Cox. Dynamic map building for an autonomous mobile robot. In *IEEE International Workshop on Intelligent Robots and Systems (IROS 90)*, 1990.

[11] A. Erkan, R. Hadsell, P. Sermanet, J. Ben, U. Muller, and Y. LeCun. Adaptive long range vision in unstructured terrain. In *IEEE International Conference on Intelligent Robots and Systems (IROS 07)*, 2007.

[12] D. Ferguson and A. Stentz. Field D*: An interpolation-based path planner and replanner. In *International Symposium on Robotics Research*, 2005.

[13] D. Forsyth and J. Ponce. *Computer Vision, A Modern Approach*. Prentice Hall, 2003.

[14] Richard Gourdeau. ROBOOP: a robotics object oriented package in C++, 2007.

[15] Michael Happold, Mark Ollis, and Nik Johnson. Enhancing supervised terrain classification with predictive unsupervised learning. In *Robotics Science and Systems*, 2006.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths in graphs. In *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[17] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[18] L. D. Jackel, Eric Krotkov, Michael Perschbacher, Jim Pippine, and Chad Sullivan. The DARPA LAGR program: Goals, challenges, methodology, and phase I results. In *Journal of Field Robotics*, 2006.

[19] M. Kam, Z. Xiaoxun, and P. Kalata. Sensor fusion for mobile robot navigation. In *Proceedings of the IEEE*, 1997.

[20] A. Kelly and A. Stentz. An approach to rough terrain autonomous mobility. In *International Conference on Mobile Planetary Robots*, 1997.

[21] Alonzo Kelly, Omead Amidi, Mike Happold, Herman Herman, Tom Pilarski, Pete Rander, Anthony Stentz, Nick Vallidis, and Randy Warner. Toward reliable off road autonomous vehicles operating in challenging environments. In *International Journal of Robotics Research*, 2006.

[22] Sven Koenig and Maxim Likhachev. Improved fast replanning for robot navigation in unknown terrain. Technical Report GIT-COGSCI-2002/3, Georgia Institute of Technology, 2002.

[23] Eric Krotkov, Scott Fish, Larry Jackel, Bill McBride, Mike Perschbacher, and Jim Pippine. The darpa perceptor evaluation experiments. In *Autonomous Robots*, 2006.

[24] D. Lieb, A. Lookingbill, and S. Thrun. Adaptive road following using self-supervised learning and reverse optical flow. In *Robotics Science and Systems (RSS 06)*, 2006.

[25] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV 99)*, 1999.

[26] R. Manduchi, A. Castano, A. Talukder, and L. Matthies. Obstacle detection and terrain classification for autonomous off-road navigation. In *Journal of Autonomous Robots*, 2005.

[27] J. Michels, A. Saxena, and A.Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning*, 2005.

[28] NASA. NASA SnakeBot, 2008. http://ti.arc.nasa.gov/ic/snakebot.

[29] J. Neufeld, J. Roberts, S. Walsh, M. Sokolski, A. Milstein, and M. Bowling. Autonomous geocaching: Navigation and goal finding in outdoor domains. In *Autonomous Agents and Multi-Agent Systems (AAMAS '08)*, 2008.

[30] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[31] Office of Science and Technology Policy National Security Council. U.s. global positioning system policy, 1996. http://clinton4.nara.gov/textonly/WH/EOP/OSTP/html/gps-factsheet.html.

[32] Jack W. Peters. *The Complete Idiot's Guide to Geocaching*. Marie Butler-Knight, 2004.

[33] Michael J. Procopio, Jane Mulligan, and Greg Grudic. Long-term learning using multiple models for outdoor autonomous robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 07)*, 2007.

[34] A Saxena, S. Chung, and AY Ng. Learning depth from single monocular images. In *Neural Information Processing Systems (NIPS 05)*, 2005.

[35] Segway. Segway rmp-can interface guide, 2005.

[36] R. Simmons and E. Krotkov. An integrated walking system for the ambler planetary rover. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1991.

[37] B. Sofman, E. Lin, J. Bagnell, N. Vandapel, and A. Stentz. Improving robot navigation through self-supervised online learning. In *Robotics Science and Systems (RSS)*, 2006.

[38] D. Stavens and S. Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. In *Uncertainty in Artificial Intelligence (UAI)*, 2006.

[39] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation (ICRA '94)*, 1994.

[40] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. In *IEEE/RSJ International Conference On Intelligent Robotic Systems (IROS 95)*, 1995.

[41] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley, the robot that won the DARPA Grand Challenge. In *Journal of Field Robotics*, 2006.

[42] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[43] Paul G. Trepagnier, Jorge Nagel, Powell M. Kinney, Cris Koutsougeras, and Matthew Dooner. Kat-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain. In *Journal of Field Robotics*, 2006.

[44] Christopher Urmson, Joshua Anhalt, Daniel Bartz, Michael Clark, Tugrul Galatali, Alexander Gutierrez, Sam Harbaugh, Joshua Johnston, Hiroki Kato, Phillip L Koon, William Messner, Nick Miller, Aaron Mosher, Kevin Peterson, Charlie Ragusa, David Ray, Bryon K Smith, Jarrod M Snider, Spencer Spiker, Joshua C Struble, Jason Ziglar, and William Red L. Whittaker. A robust approach to high-speed navigation. In *Journal of Field Robotics*, 2006.

[45] C. Wellington and A. Stentz. Online adaptive rough-terrain navigation in vegetation. In *IEEE Int. Conference on Robotics and Automation (ICRA 04)*, 2004.

[46] Carl Wellington and Anthony Stentz. Learning predictions of the load-bearing surface for autonomous rough-terrain navigation in vegetation. In *Proceedings of the Int. Conference on Field and Service Robotics (FSR 03)*, 2003.

[47] A. Yahja, S.Singhand, and A. Stentz. Recent results in path planning for mobile robots operating in vast outdoor environments. In *Symposium on Image, Speech, Signal Processing and Robotics*, 1998.